```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = ':https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F310%2F23498%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading , 69155672 bytes compressed
[==================================================] 69155672 bytes downloaded
Downloaded and uncompressed: 
Data source import complete.
```

```
# read & manipulate data
import pandas as pd
import numpy as np
import tensorflow as tf

# visualisations
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='whitegrid', context='notebook')
%matplotlib notebook

# misc
import random as rn

# load the dataset
df = pd.read_csv('../input/creditcard.csv')

# manual parameters
RANDOM_SEED = 42
TRAINING_SAMPLE = 200000
VALIDATE_SIZE = 0.2

# setting random seeds for libraries to ensure reproducibility
np.random.seed(RANDOM_SEED)
rn.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)  # Use tf.random.set_seed() instead of tf.set_random_seed()
```

```
# let's quickly convert the columns to lower case and rename the Class column
# so as to not cause syntax errors
df.columns = map(str.lower, df.columns)
df.rename(columns={'class': 'label'}, inplace=True)

# print first 5 rows to get an initial impression of the data we're dealing with
df.head()
```

	time	v1	v2	v3	v4	v5	v6	v7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

```
# add a negligible amount to avoid taking the log of 0
df['log10_amount'] = np.log10(df.amount + 0.00001)
```

```
# keep the label field at the back
df = df[
    [col for col in df if col not in ['label', 'log10_amount']] +
    ['log10_amount', 'label']
]
```

```
# manual parameter
RATIO_TO_FRAUD = 15

# dropping redundant columns
df = df.drop(['time', 'amount'], axis=1)

# splitting by class
fraud = df[df.label == 1]
clean = df[df.label == 0]

# undersample clean transactions
clean_undersampled = clean.sample(
    int(len(fraud) * RATIO_TO_FRAUD),
    random_state=RANDOM_SEED
)

# concatenate with fraud transactions into a single dataframe
visualisation_initial = pd.concat([fraud, clean_undersampled])
column_names = list(visualisation_initial.drop('label', axis=1).columns)

# isolate features from labels
features, labels = visualisation_initial.drop('label', axis=1).values, \
                   visualisation_initial.label.values


print(f"""The non-fraud dataset has been undersampled from {len(clean):,} to {len(clean_undersampled):,}.
This represents a ratio of {RATIO_TO_FRAUD}:1 to fraud.""")
```

```
The non-fraud dataset has been undersampled from 284,315 to 7,380.
This represents a ratio of 15:1 to fraud.
```

```
from sklearn.manifold import TSNE
from mpl_toolkits.mplot3d import Axes3D

def tsne_scatter(features, labels, dimensions=2, save_as='graph.png'):
    if dimensions not in (2, 3):
        raise ValueError('tsne_scatter can only plot in 2d or 3d (What are you? An alien that can visualise >3d?). Make sure the "dimensi

    # t-SNE dimensionality reduction
    features_embedded = TSNE(n_components=dimensions, random_state=RANDOM_SEED).fit_transform(features)

    # initialising the plot
    fig, ax = plt.subplots(figsize=(8,8))

    # counting dimensions
    if dimensions == 3: ax = fig.add_subplot(111, projection='3d')

    # plotting data
    ax.scatter(
        *zip(*features_embedded[np.where(labels==1)]),
        marker='o',
        color='r',
        s=2,
        alpha=0.7,
        label='Fraud'
    )
    ax.scatter(
        *zip(*features_embedded[np.where(labels==0)]),
        marker='o',
        color='g',
        s=2,
        alpha=0.3,
        label='Clean'
    )

    # storing it to be displayed later
    plt.legend(loc='best')
    plt.savefig(save_as);
    plt.show;


tsne_scatter(features, labels, dimensions=2, save_as='tsne_initial_2d.png')
```

```
print(f"""Shape of the datasets:
    clean (rows, cols) = {clean.shape}
    fraud (rows, cols) = {fraud.shape}""")
```

```
Shape of the datasets:
    clean (rows, cols) = (284315, 30)
    fraud (rows, cols) = (492, 30)
```

```
# shuffle our training set
clean = clean.sample(frac=1).reset_index(drop=True)

# training set: exlusively non-fraud transactions
X_train = clean.iloc[:TRAINING_SAMPLE].drop('label', axis=1)

# testing  set: the remaining non-fraud + all the fraud
X_test = clean.iloc[TRAINING_SAMPLE:].append(fraud).sample(frac=1)
```

```
<ipython-input-17-334e732d32cd>:8: FutureWarning: The frame.append method is deprecat
  X_test = clean.iloc[TRAINING_SAMPLE:].append(fraud).sample(frac=1)
```

```
print(f"""Our testing set is composed as follows:

{X_test.label.value_counts()}""")
```

```
Our testing set is composed as follows:

0    84315
1      492
Name: label, dtype: int64
```

```
from sklearn.model_selection import train_test_split

# train // validate - no labels since they're all clean anyway
X_train, X_validate = train_test_split(X_train,
                                       test_size=VALIDATE_SIZE,
                                       random_state=RANDOM_SEED)

# manually splitting the labels from the test df
X_test, y_test = X_test.drop('label', axis=1).values, X_test.label.values
```

```
print(f"""Shape of the datasets:
    training (rows, cols) = {X_train.shape}
    validate (rows, cols) = {X_validate.shape}
    holdout  (rows, cols) = {X_test.shape}""")
```

```
Shape of the datasets:
    training (rows, cols) = (160000, 29)
    validate (rows, cols) = (40000, 29)
    holdout  (rows, cols) = (84807, 29)
```

```
from sklearn.preprocessing import Normalizer, MinMaxScaler
from sklearn.pipeline import Pipeline

# configure our pipeline
pipeline = Pipeline([('normalizer', Normalizer()),
                     ('scaler', MinMaxScaler())])
```

```
# get normalization parameters by fitting to the training data
pipeline.fit(X_train);
```

```
# transform the training and validation data with these parameters
X_train_transformed = pipeline.transform(X_train)
X_validate_transformed = pipeline.transform(X_validate)
```

```
g = sns.PairGrid(X_train.iloc[:,:3].sample(600, random_state=RANDOM_SEED))
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Before:')
g.map_diag(sns.kdeplot)
g.map_offdiag(sns.kdeplot);
```

```
g = sns.PairGrid(pd.DataFrame(X_train_transformed, columns=column_names).iloc[:,:3].sample(600, random_state=RANDOM_SEED))
plt.subplots_adjust(top=0.9)
g.fig.suptitle('After:')
g.map_diag(sns.kdeplot)
g.map_offdiag(sns.kdeplot);
```

```
# Load the extension and start TensorBoard
%load_ext tensorboard
%tensorboard --logdir logs
```

TensorBoard

INACTIVE
SCALARS

Show data download links
Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing

0.6

Horizontal Axis

STEP RELATIVE

WALL

Runs

Write a regex to filter runs

TOGGLE ALL RUNS

logs

No scalar data was found.

Probable causes:

- You haven't written any scalar data to your event files.
- TensorBoard can't find your event files.

If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the README and perhaps the TensorBoard tutorial.

If you think TensorBoard is configured properly, please see the section of the README devoted to missing data problems and consider filing an issue on GitHub.

```
# data dimensions // hyperparameters
input_dim = X_train_transformed.shape[1]
BATCH_SIZE = 256
EPOCHS = 100

# https://keras.io/layers/core/
autoencoder = tf.keras.models.Sequential([

    # deconstruct / encode
    tf.keras.layers.Dense(input_dim, activation='elu', input_shape=(input_dim, )),
    tf.keras.layers.Dense(16, activation='elu'),
    tf.keras.layers.Dense(8, activation='elu'),
    tf.keras.layers.Dense(4, activation='elu'),
    tf.keras.layers.Dense(2, activation='elu'),

    # reconstruction / decode
    tf.keras.layers.Dense(4, activation='elu'),
    tf.keras.layers.Dense(8, activation='elu'),
    tf.keras.layers.Dense(16, activation='elu'),
    tf.keras.layers.Dense(input_dim, activation='elu')

])

# https://keras.io/api/models/model_training_apis/
autoencoder.compile(optimizer="adam",
                    loss="mse",
                    metrics=["acc"])

# print an overview of our model
autoencoder.summary();
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 29)                870

 dense_1 (Dense)             (None, 16)                480

 dense_2 (Dense)             (None, 8)                 136

 dense_3 (Dense)             (None, 4)                 36

 dense_4 (Dense)             (None, 2)                 10

 dense_5 (Dense)             (None, 4)                 12

 dense_6 (Dense)             (None, 8)                 40

 dense_7 (Dense)             (None, 16)                144

 dense_8 (Dense)             (None, 29)                493

=================================================================
Total params: 2221 (8.68 KB)
Trainable params: 2221 (8.68 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from datetime import datetime

# current date and time
yyyymmddHHMM = datetime.now().strftime('%Y%m%d%H%M')

# new folder for a new run
log_subdir = f'{yyyymmddHHMM}_batch{BATCH_SIZE}_layers{len(autoencoder.layers)}'

# define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True
)

save_model = tf.keras.callbacks.ModelCheckpoint(
    filepath='autoencoder_best_weights.hdf5',
    save_best_only=True,
    monitor='val_loss',
    verbose=0,
    mode='min'
)

tensorboard = tf.keras.callbacks.TensorBoard(
    f'logs/{log_subdir}',
    batch_size=BATCH_SIZE,
    update_freq='batch'
)

# callbacks argument only takes a list
cb = [early_stop, save_model, tensorboard]
```

Training

```
history = autoencoder.fit(
    X_train_transformed, X_train_transformed,
    shuffle=True,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=cb,
    validation_data=(X_validate_transformed, X_validate_transformed)
);
```

```
Epoch 1/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0111 - acc: 0.373
Epoch 2/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0110 - acc: 0.377
Epoch 3/100
625/625 [==============================] - 10s 16ms/step - loss: 0.0110 - acc: 0.3
Epoch 4/100
625/625 [==============================] - 5s 8ms/step - loss: 0.0109 - acc: 0.380
Epoch 5/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0109 - acc: 0.381
Epoch 6/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0108 - acc: 0.381
Epoch 7/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0108 - acc: 0.381
Epoch 8/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0108 - acc: 0.381
Epoch 9/100
625/625 [==============================] - 4s 7ms/step - loss: 0.0107 - acc: 0.382
Epoch 10/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0107 - acc: 0.384
Epoch 11/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0107 - acc: 0.385
Epoch 12/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0106 - acc: 0.386
Epoch 13/100
625/625 [==============================] - 6s 10ms/step - loss: 0.0106 - acc: 0.38
Epoch 14/100
625/625 [==============================] - 4s 7ms/step - loss: 0.0106 - acc: 0.387
Epoch 15/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0105 - acc: 0.387
Epoch 16/100
625/625 [==============================] - 5s 9ms/step - loss: 0.0105 - acc: 0.387
Epoch 17/100
625/625 [==============================] - 5s 8ms/step - loss: 0.0105 - acc: 0.388
Epoch 18/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0105 - acc: 0.388
Epoch 19/100
625/625 [==============================] - 6s 10ms/step - loss: 0.0104 - acc: 0.38
Epoch 20/100
625/625 [==============================] - 5s 9ms/step - loss: 0.0104 - acc: 0.389
Epoch 21/100
625/625 [==============================] - 6s 10ms/step - loss: 0.0104 - acc: 0.39
Epoch 22/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0104 - acc: 0.391
Epoch 23/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0103 - acc: 0.392
Epoch 24/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0103 - acc: 0.393
Epoch 25/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0103 - acc: 0.394
Epoch 26/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0103 - acc: 0.396
Epoch 27/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0103 - acc: 0.397
Epoch 28/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0102 - acc: 0.399
Epoch 29/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0102 - acc: 0.400
Epoch 30/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0102 - acc: 0.401
Epoch 31/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0102 - acc: 0.402
Epoch 32/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0102 - acc: 0.403
Epoch 33/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0102 - acc: 0.404
Epoch 34/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0101 - acc: 0.404
Epoch 35/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0101 - acc: 0.405
Epoch 36/100
625/625 [==============================] - 4s 7ms/step - loss: 0.0101 - acc: 0.406
Epoch 37/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0101 - acc: 0.408
Epoch 38/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0101 - acc: 0.407
Epoch 39/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0100 - acc: 0.409
Epoch 40/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0100 - acc: 0.410
Epoch 41/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0100 - acc: 0.411
Epoch 42/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0100 - acc: 0.413
Epoch 43/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0099 - acc: 0.413
Epoch 44/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0099 - acc: 0.415
Epoch 45/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0099 - acc: 0.415
```

```
Epoch 46/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0099 - acc: 0.416
Epoch 47/100
625/625 [==============================] - 4s 7ms/step - loss: 0.0098 - acc: 0.417
Epoch 48/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0098 - acc: 0.418
Epoch 49/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0098 - acc: 0.417
Epoch 50/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0098 - acc: 0.419
Epoch 51/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0097 - acc: 0.419
Epoch 52/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0097 - acc: 0.420
Epoch 53/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0097 - acc: 0.422
Epoch 54/100
625/625 [==============================] - 4s 7ms/step - loss: 0.0097 - acc: 0.423
Epoch 55/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0096 - acc: 0.424
Epoch 56/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0096 - acc: 0.423
Epoch 57/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0096 - acc: 0.424
Epoch 58/100
625/625 [==============================] - 4s 7ms/step - loss: 0.0096 - acc: 0.425
Epoch 59/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0096 - acc: 0.425
Epoch 60/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0095 - acc: 0.425
Epoch 61/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0095 - acc: 0.425
Epoch 62/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0095 - acc: 0.425
Epoch 63/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0095 - acc: 0.425
Epoch 64/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0095 - acc: 0.428
Epoch 65/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0095 - acc: 0.427
Epoch 66/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0095 - acc: 0.428
Epoch 67/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0094 - acc: 0.428
Epoch 68/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0094 - acc: 0.429
Epoch 69/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0094 - acc: 0.429
Epoch 70/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0094 - acc: 0.429
Epoch 71/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0094 - acc: 0.429
Epoch 72/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0094 - acc: 0.430
Epoch 73/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0094 - acc: 0.430
Epoch 74/100
625/625 [==============================] - 4s 6ms/step - loss: 0.0094 - acc: 0.430
Epoch 75/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0094 - acc: 0.430
Epoch 76/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0093 - acc: 0.431
Epoch 77/100
625/625 [==============================] - 4s 7ms/step - loss: 0.0093 - acc: 0.430
Epoch 78/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0093 - acc: 0.431
Epoch 79/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0093 - acc: 0.431
Epoch 80/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0093 - acc: 0.431
Epoch 81/100
625/625 [==============================] - 4s 7ms/step - loss: 0.0093 - acc: 0.431
Epoch 82/100
625/625 [==============================] - 3s 5ms/step - loss: 0.0093 - acc: 0.431
Epoch 83/100
621/625 [============================>.] - ETA: 0s - loss: 0.0093 - acc: 0.4316Res
625/625 [==============================] - 3s 5ms/step - loss: 0.0093 - acc: 0.431
```

```
# transform the test set with the pipeline fitted to the training set
X_test_transformed = pipeline.transform(X_test)

# pass the transformed test set through the autoencoder to get the reconstructed result
reconstructions = autoencoder.predict(X_test_transformed)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
  warnings.warn(
Exception ignored in: <function _xla_gc_callback at 0x786b05726f80>
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/jax/_src/lib/__init__.py", line 97, i
    def _xla_gc_callback(*args):
KeyboardInterrupt:
2651/2651 [==============================] - 4s 1ms/step
```

```python
# calculating the mean squared error reconstruction loss per row in the numpy array
mse = np.mean(np.power(X_test_transformed - reconstructions, 2), axis=1)
```

```python
clean = mse[y_test==0]
fraud = mse[y_test==1]

fig, ax = plt.subplots(figsize=(6,6))

ax.hist(clean, bins=50, density=True, label="clean", alpha=.6, color="green")
ax.hist(fraud, bins=50, density=True, label="fraud", alpha=.6, color="red")

plt.title("(Normalized) Distribution of the Reconstruction Loss")
plt.legend()
plt.show()
```

```python
THRESHOLD = 3

def mad_score(points):
    """https://www.itl.nist.gov/div898/handbook/eda/section3/eda35h.htm """
    m = np.median(points)
    ad = np.abs(points - m)
    mad = np.median(ad)

    return 0.6745 * ad / mad

z_scores = mad_score(mse)
outliers = z_scores > THRESHOLD
```

```python
print(f"Detected {np.sum(outliers):,} outliers in a total of {np.size(z_scores):,} transactions [{np.sum(outliers)/np.size(z_scores):.2
```

```
Detected 1,892 outliers in a total of 84,807 transactions [2.23%].
```

```python
from sklearn.metrics import (confusion_matrix,
                             precision_recall_curve)

# get (mis)classification
cm = confusion_matrix(y_test, outliers)

# true/false positives/negatives
(tn, fp,
 fn, tp) = cm.flatten()
```

```python
print(f"""The classifications using the MAD method with threshold={THRESHOLD} are as follows:
{cm}

% of transactions labeled as fraud that were correct (precision): {tp}/({fp}+{tp}) = {tp/(fp+tp):.2%}
% of fraudulent transactions were caught succesfully (recall):    {tp}/({fn}+{tp}) = {tp/(fn+tp):.2%}""")
```

```
The classifications using the MAD method with threshold=3 are as follows:
[[82784  1531]
 [  131   361]]

% of transactions labeled as fraud that were correct (precision): 361/(1531+361) = 19
```

```python
clean = z_scores[y_test==0]
fraud = z_scores[y_test==1]

fig, ax = plt.subplots(figsize=(6,6))

ax.hist(clean, bins=50, density=True, label="clean", alpha=.6, color="green")
ax.hist(fraud, bins=50, density=True, label="fraud", alpha=.6, color="red")

plt.title("Distribution of the modified z-scores")
plt.legend()
plt.show()
```

```
encoder = tf.keras.models.Sequential(autoencoder.layers[:5])
encoder.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 29)                870

 dense_1 (Dense)             (None, 16)                480

 dense_2 (Dense)             (None, 8)                 136

 dense_3 (Dense)             (None, 4)                 36

 dense_4 (Dense)             (None, 2)                 10

=================================================================
Total params: 1532 (5.98 KB)
Trainable params: 1532 (5.98 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# taking all the fraud, undersampling clean
fraud = X_test_transformed[y_test==1]
clean = X_test_transformed[y_test==0][:len(fraud) * RATIO_TO_FRAUD, ]

# combining arrays & building labels
features = np.append(fraud, clean, axis=0)
labels = np.append(np.ones(len(fraud)),
                   np.zeros(len(clean)))

# getting latent space representation
latent_representation = encoder.predict(features)

print(f'Clean transactions downsampled from {len(X_test_transformed[y_test==0]):,} to {len(clean):,}.')
print('Shape of latent representation:', latent_representation.shape)
```

```
246/246 [==============================] - 0s 1ms/step
Clean transactions downsampled from 84,315 to 7,380.
```