```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Example 3.7 Multiclass confusion matrix for the MINST data set

Developed for Machine Learning for Mechanical Engineers at the University of
South Carolina

@author: austin_downey
"""

import IPython as IP
IP.get_ipython().run_line_magic('reset', '-sf')

import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk

cc = plt.rcParams['axes.prop_cycle'].by_key()['color']
plt.close('all')

#%% Load your data

# Fetch the MNIST dataset from openml
mnist = sk.datasets.fetch_openml('mnist_784',as_frame=False,parser='auto')
X = np.asarray(mnist['data'])     # load the data and convert to np array
Y = np.asarray(mnist['target'],dtype=int)  # load the target

# Split the data set up into a training and testing data set
X_train = X[0:60000,:]
X_test = X[60000:,:]
Y_train = Y[0:60000]
Y_test = Y[60000:]

#%% Confusion Matrix for a Multiclass classifier.

# Use the one-vs-one classifier that uses Stochastic Gradient Descent as this is
# faster for this specific data set

# here we test a
ovo_clf = sk.multiclass.OneVsOneClassifier(sk.linear_model.SGDClassifier())

# make a prediction for every case using the k-fold method.
Y_train_pred = sk.model_selection.cross_val_predict(ovo_clf, X_train, Y_train, cv=3)
conf_mx = sk.metrics.confusion_matrix(Y_train, Y_train_pred)
print(conf_mx)

# plot the results
fig = plt.figure(figsize=(4,4))
pos = plt.imshow(conf_mx) #, cmap=plt.cm.gray)
cbar = plt.colorbar(pos)
cbar.set_label('number of classified digits')
plt.ylabel('actual digit')
plt.xlabel('estimated digit')
plt.savefig('confusion_matrix',dpi=300)

# next look at only the noise in the system, to do this, divide each value in
# the confusion matrix by the number of images in the corresponding class, so
# you can compare error rates instead of the absolute number of errors (which would
# make abundant classes look unfairly bad):
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums


# Next, we remove the high values along the diagonal. This is done by converting
# the confusion matrix to a float data type, and replacing # everything on the
# diagonal with NaNs.
```

```python
conf_mx_noise = np.asarray(norm_conf_mx,dtype=np.float32)
np.fill_diagonal(conf_mx_noise, np.NaN)


# plot the results but only consider the noise
fig = plt.figure(figsize=(4,4))
pos = plt.imshow(conf_mx_noise) #, cmap=plt.cm.gray)
cbar = plt.colorbar(pos)
cbar.set_label('normalized classification error')
plt.ylabel('actual digit')
plt.xlabel('estimated digit')
plt.savefig('confusion_matrix_error',dpi=300)



```