[ ]:
```
from keras.datasets import mnist
import matplotlib.pyplot as plt
import seaborn as sns ; sns.set()
from sklearn import metrics
import numpy as np
from numpy.linalg import svd, norm
from tqdm import tqdm
```

Fig 1

[ ]:
```
A = np.random.randint(0, 10, (100))
print("rank A: ", 1)
A_1 = A + np.random.normal(0, 0.01, (100, 100))
print("rank A after noise: ", np.linalg.matrix_rank(A_1))
ax = plt.gca()
ax.set_yscale('log')
_,e,_ = svd(A_1)
n = e.shape[0]
ax.scatter(np.arange(n), e, 5.0);
```

[ ]:
```
M = 28 * 28
N = 28
K = 10
```

[ ]:
```
(train_x , train_y), (test_x, test_y) = mnist.load_data()
train_x, test_x = train_x.reshape(-1, M), test_x.reshape(-1, M)
```

Fig 2

[ ]:
```
from matplotlib.gridspec import GridSpec
pics = train_x[np.random.choice(train_x.shape[0], 6)]

def format_axes(fig):
    for i, ax in enumerate(fig.axes):
        ax.imshow(pics[i].reshape(N, N), cmap='gray')
        ax.tick_params(labelbottom=False, labelleft=False)

fig = plt.figure(constrained_layout=True)
fig.set_size_inches(5, 5)

gs = GridSpec(3, 3, figure=fig)
ax1 = fig.add_subplot(gs[:2, :2])
ax2 = fig.add_subplot(gs[0, 2])
ax3 = fig.add_subplot(gs[1, 2])
ax4 = fig.add_subplot(gs[2, 2])
ax5 = fig.add_subplot(gs[2, 0])
ax6 = fig.add_subplot(gs[2, 1])
```

```
format_axes(fig)
plt.show()
```

Fig 3

[ ]:
```
fig, ax = plt.subplots(2, 5, sharey=True)
fig.set_size_inches(5, 2)
fig.subplots_adjust(wspace=0, hspace=0)

means = np.zeros(shape=(10, M))
for i in range(10) :
    t = train_x[train_y == i]
    means[i] = np.mean(t, axis=0)
    a = ax[i//5, i%5]
    a.imshow(means[i].reshape(N, N), cmap='gray')
    a.tick_params(labelbottom=False, labelleft=False)
    a.set_xticklabels([])
    a.set_yticklabels([])
```

[ ]:
```
# computing clustering accuracy
pred = np.argmin(norm(means - test_x.reshape(-1, 1 , M), 2, axis=2), axis=1)
print( "accuracy: ", test_accuracy(pred) , "%" )
```

[ ]:
```
bases = np.zeros(shape=(10, M, M))
for i in range(10) :
    t = train_x[train_y == i].T
    u, _, _ = svd(t, full_matrices=False)
    bases[i] = u
```

[ ]:
```
fig, ax = plt.subplots(10, 10, sharex=True, sharey=True)
fig.set_size_inches(20, 20)
fig.subplots_adjust(wspace=0, hspace=0)
for i in range(10) :
    for j in range(K) :
        ax[i, j].imshow(bases[i][:, j].reshape(N, N), cmap='gray')
        ax[i, j].tick_params(labelbottom=False, labelleft=False)
```

Fig 4

[ ]:
```
fig, ax = plt.subplots(2, 3, sharex=True, sharey=True)
fig.set_size_inches(3, 2)
fig.subplots_adjust(wspace=0, hspace=0)
for i, v in enumerate([3,6]):
    for j in range(3) :
        ax[i, j].imshow(bases[v][:, j].reshape(N, N), cmap='gray')
        ax[i, j].tick_params(labelbottom=False, labelleft=False)
        ax[i, j].set_xticklabels([])
```

```
        ax[i, j].set_yticklabels([])
```

Fig 5

[ ]:
```
fig, ax = plt.subplots(1, 2)
fig.set_size_inches(20, 7)
for i, d in enumerate([3, 7]) :
    res = residual(num_bases=10, digit=d)
    ax[i].plot(res.T)
    ax[i].set_xticks(np.arange(10));
    ax[i].set_xlabel("base")
    ax[i].set_ylabel("residual")
```

Fig 6

[ ]:
```
z = train_x[train_y == 3][0]
plt.imshow(z.reshape(N, N), cmap='gray')
```

[ ]:
```
fig, ax = plt.subplots(1, 6, sharex=True, sharey=True)
fig.subplots_adjust(wspace=0, hspace=0)

u = bases[5]
for i, j in enumerate([3, 10, 50, 125, 250, 500]) :
    u_k = u[:, :j]
    a = u_k.T @ z
    ax[i].imshow((u_k @ a).reshape(N, N), cmap='gray')
    ax[i].set_xlabel(f'k={j}')
    ax[i].set_xticklabels([])
    ax[i].set_yticklabels([])
```

Fig 7

[ ]:
```
z = train_x[train_y == 4][0]
plt.imshow(z.reshape(N, N), cmap='gray')
```

[ ]:
```
fig, ax = plt.subplots(1, 6, sharex=True, sharey=True)
fig.subplots_adjust(wspace=0, hspace=0)

u = bases[4]
for i, j in enumerate([3, 10, 125, 250, 500, 784]) :
    u_k = u[:, :j]
    a = u_k.T @ z
    ax[i].imshow((u_k @ a).reshape(N, N), cmap='gray')
    ax[i].set_xlabel(f'k={j}')
    ax[i].set_xticklabels([])
    ax[i].set_yticklabels([])
```

Fig 8

[ ]:
```
fig, ax = plt.subplots(1, 2)
fig.set_size_inches(15, 5)
ax[0].plot(np.cumsum(e) / np.sum(e) * 100)
ax[0].set_ylabel("explained variance")
ax[1].set_yscale('log')
ax[1].set_ylabel("eigen value")
ax[1].plot(np.arange(len(e)), e)
```

[ ]:
```
history = []
b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] + [20, 30, 40, 50] + [100, 300, 500]
for K in tqdm(b) :
    res = residual(num_bases=K)
    y = np.argmin(res, axis=1).reshape(-1)
    history += [test_accuracy(y)]
```

[ ]:
```
plt.xticks(np.arange(len(b)), labels=b)
plt.plot(history, 'bx--');
plt.xlabel("number of bases (k)")
plt.ylabel("accuracy")
```

Fig 9

[ ]:
```
history = []
b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] + [20, 30, 40, 50] + [100, 300, 500]
for K in tqdm([500, 784]) :
    res = residual(num_bases=K)
    y = np.argmin(res, axis=1).reshape(-1)
    history += [test_accuracy(y)]
```

[ ]:
```
plt.xticks(np.arange(len(b)), b)
plt.plot(history, 'bx--');
```

Fig 10

[ ]:
```
fig, ax = plt.subplots(1, 2)
fig.set_size_inches(10, 7)
for i, b in enumerate([50, 100]) :
    ax[i].imshow(bases[4][:, b].reshape(N, N), cmap="gray") ;
    ax[i].set_xlabel(f'k={b}')
    ax[i].set_xticklabels([])
    ax[i].set_yticklabels([])
```

Fig 11

[ ]:
```
res = residual(num_bases=30)
y = np.argmin(res, axis=1).reshape(-1)
z = metrics.confusion_matrix(test_y, y, normalize='true')
sns.heatmap(z, annot=True)
```

```
plt.rcParams["figure.figsize"] = (10, 5)
plt.xlabel("predictions")
plt.ylabel("true values")
plt.show()
```

[ ]:
```
def test_accuracy(y) :
    return np.sum(y == test_y) / len(test_y) * 100
```

[ ]:
```
def residual(num_bases, digit=None) :
    """
    get_residual returns residual given by :
                    min || z - u @ a || / ||z||
    where `z` is the target vector, `u` is the basis which is
    used to estimate `z`, and `a` are multipliers.
    ---------------
    parameters :
        num_bases : number of bases used to approximate m by n
                    matrix A in SVD decomposition.
        digit : use only digits equal to digit in test set if
                specified, default None
    """
    u = bases[:, :, :num_bases]
    z = test_x if (digit == None) else test_x[test_y == digit]
    z = z.reshape(-1, 1, M, 1)
    res = (np.eye(M) - u @ np.transpose(u, (0, 2, 1))) @ z
    res = norm(res, 2, axis=2) / norm(z, 2, axis=2)
    res = res.reshape(-1, 10)
    return res
```