```
# PACKAGE: DO NOT EDIT THIS CELL
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import numpy as np


from sklearn.datasets import fetch_olivetti_faces, fetch_lfw_people
from ipywidgets import interact
%matplotlib inline
image_shape = (64, 64)
# Load faces data
dataset = fetch_olivetti_faces('./')
faces = dataset.data
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-7b401ecb1f8a> in <module>
      4 image_shape = (64, 64)
      5 # Load faces data
----> 6 dataset = fetch_olivetti_faces('./')
      7 faces = dataset.data

TypeError: fetch_olivetti_faces() takes 0 positional arguments but 1 was given

SEARCH STACK OVERFLOW
```

```
import numpy.testing as np_test
def test_property_projection_matrix(P):
    """Test if the projection matrix satisfies certain properties.
    In particular, we should have P @ P = P, and P = P^T
    """
    np_test.assert_almost_equal(P, P @ P)
    np_test.assert_almost_equal(P, P.T)

def test_property_projection(x, p):
    """Test orthogonality of x and its projection p."""
    np_test.assert_almost_equal(np.dot(p-x, p), 0)


# GRADED FUNCTION: DO NOT EDIT THIS LINE
# Projection 1d

# ===YOU SHOULD EDIT THIS FUNCTION===
def projection_matrix_1d(b):
    """Compute the projection matrix onto the space spanned by `b`
    Args:
        b: ndarray of dimension (D,), the basis for the subspace

    Returns:
        P: the projection matrix
    """
    D, = b.shape
    b = b[:, np.newaxis]
    P = np.dot(b,b.T)/np.dot(b.T,b) # EDIT THIS
    return P

# ===YOU SHOULD EDIT THIS FUNCTION===
def project_1d(x, b):
    """Compute the projection matrix onto the space spanned by `b`
    Args:
        x: the vector to be projected
        b: ndarray of dimension (D,), the basis for the subspace

    Returns:
        y: projection of x in space spanned by b
    """
    p = np.matmul(projection_matrix_1d(b), x[:, np.newaxis]).ravel() # EDIT THIS
    return p

# Projection onto general subspace
# ===YOU SHOULD EDIT THIS FUNCTION===
def projection_matrix_general(B):
    """Compute the projection matrix onto the space spanned by `B`
    Args:
        B: ndarray of dimension (D, M), the basis for the subspace

    Returns:
        P: the projection matrix
    """
```

```
    #P = np.dot(np.dot(B,(np.dot(B.T,B)).I),B.T) # EDIT THIS
    P = np.dot(np.dot( B, np.linalg.pinv(np.dot(B.T,B))), B.T)
    return P

# ===YOU SHOULD EDIT THIS FUNCTION===
def project_general(x, B):
    """Compute the projection matrix onto the space spanned by `B`
    Args:
        B: ndarray of dimension (D, E), the basis for the subspace

    Returns:
        y: projection of x in space spanned by b
    """
    p = np.dot(projection_matrix_general(B),x) # EDIT THIS
    return p


# Orthogonal projection in 2d
# define basis vector for subspace
b = np.array([2,1]).reshape(-1,1)
# point to be projected later
x = np.array([1,2]).reshape(-1, 1)


# Test 1D
np_test.assert_almost_equal(projection_matrix_1d(np.array([1, 2, 2])),
                            np.array([[1,  2,  2],
                                      [2,  4,  4],
                                      [2,  4,  4]]) / 9)

np_test.assert_almost_equal(project_1d(np.ones(3),
                                       np.array([1, 2, 2])),
                            np.array([5, 10, 10]) / 9)

B = np.array([[1, 0],
              [1, 1],
              [1, 2]])

# Test General
np_test.assert_almost_equal(projection_matrix_general(B),
                            np.array([[5,  2, -1],
                                      [2,  2,  2],
                                      [-1, 2,  5]]) / 6)

np_test.assert_almost_equal(project_general(np.array([6, 0, 0]), B),
                            np.array([5, 2, -1]))
print('correct')
```

```
correct
```

```
from sklearn.datasets import fetch_olivetti_faces, fetch_lfw_people
from ipywidgets import interact
%matplotlib inline
image_shape = (64, 64)
# Load faces data
dataset = fetch_olivetti_faces('./')
faces = dataset.data
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-7b401ecb1f8a> in <module>
      4 image_shape = (64, 64)
      5 # Load faces data
----> 6 dataset = fetch_olivetti_faces('./')
      7 faces = dataset.data

TypeError: fetch_olivetti_faces() takes 0 positional arguments but 1 was given
```

SEARCH STACK OVERFLOW

```
plt.figure(figsize=(10,10))
plt.imshow(np.hstack(faces[:5].reshape(5,64,64)), cmap='gray');
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
# for numerical reasons we normalize the dataset
mean = faces.mean(axis=0)
std = faces.std(axis=0)
faces_normalized = (faces - mean) / std


B = np.load('eigenfaces.npy')[:50] # we use the first 50 basis vectors --- you should play around with this.
print("the eigenfaces have shape {}".format(B.shape))


plt.figure(figsize=(10,10))
plt.imshow(np.hstack(B[:5].reshape(-1, 64, 64)), cmap='gray');


# EDIT THIS FUNCTION
@interact(i=(0, 10))
def show_face_face_reconstruction(i):
    original_face = faces_normalized[i].reshape(64, 64)
    # reshape the data we loaded in variable `B`
    # so that we have a matrix representing the basis.
    B_basis = np.random.normal(size=(4096,50)) # <-- EDIT THIS
    face_reconstruction = project_general(faces_normalized[i], B_basis).reshape(64, 64)
    plt.figure()
    plt.imshow(np.hstack([original_face, face_reconstruction]), cmap='gray')
    plt.show()


x = np.linspace(0, 10, num=50)
theta = 2
def f(x):
    random = np.random.RandomState(42) # we use the same random seed so we get deterministic output
    return theta * x + random.normal(scale=1.0, size=len(x)) # our observations are corrupted by some noise, so that we do not get (x,y)

y = f(x)
plt.scatter(x, y);
plt.xlabel('x');
plt.ylabel('y');


X = x.reshape(-1,1) # size N x 1
Y = y.reshape(-1,1) # size N x 1

# maximum likelihood estimator
theta_hat = np.linalg.solve(X.T @ X, X.T @ Y)


fig, ax = plt.subplots()
ax.scatter(x, y);
xx = [0, 10]
yy = [0, 10 * theta_hat[0,0]]
ax.plot(xx, yy, 'red', alpha=.5);
ax.set(xlabel='x', ylabel='y');
print("theta = %f" % theta)
print("theta_hat = %f" % theta_hat)


N = np.arange(2, 10000, step=10)
# Your code comes here, which calculates \hat{\theta} for different dataset sizes.

theta_error = np.zeros(N.shape)

theta_error = np.ones(N.shape) # <-- EDIT THIS


plt.plot(theta_error)
plt.xlabel("dataset size")
plt.ylabel("parameter error");
```

0s completed at 6:31 PM