Q1a Code:

```
import numpy as np
import cv2
import os

# Define the path to the folder containing the images
notebook_path = os.path.abspath("Q1.ipynb")
path_to_folder = os.path.join(os.path.dirname(notebook_path), "data/ORL_dataset/")

# Initialize an empty numpy array to store the image data
data = np.empty((400, 2576))

# Initialize an empty list to store the gender labels
labels = []

# Loop through each image in the folder
for i, filename in enumerate(os.listdir(path_to_folder)):
    # Read the image using OpenCV
    img = cv2.imread(os.path.join(path_to_folder, filename), cv2.IMREAD_GRAYSCALE)

    # Flatten the image into a 1D numpy array
    img_flat = img.flatten()

    # Add the flattened image data to the data matrix
    data[i, :] = img_flat

    if filename.startswith('1_'):
        labels.append(0)
    elif filename.startswith('8_'):
        labels.append(0)
    elif filename.startswith('10_'):
        labels.append(0)
    elif filename.startswith('32_'):
        labels.append(0)
    else:
        labels.append(1)

# Convert the labels list to a numpy array and reshape to a column vector
labels = np.array(labels).reshape((400, 1))

# Concatenate the labels column to the data matrix
data_with_labels = np.concatenate((data, labels), axis=1)

print('Data shape:', data.shape)
print('Labels shape:', labels.shape)
print('Data with Labels', data_with_labels.shape)
```

Q1a Result:

```
Data shape: (400, 2576)
Labels shape: (400, 1)
Data with Labels (400, 2577)
```

Q1b-1 code:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import SequentialFeatureSelector

X = data_with_labels[:, :-1]
Y = data_with_labels[:, -1]

model = LinearRegression()
model.fit(X, Y)

print("R-squared:", model.score(X, Y))

import statsmodels.api as sm

# Split the data matrix into predictor variables (pixels) and response variable (gender label)
X = data_with_labels[:, :-1]
y = data_with_labels[:, -1]

# Add a constant term to the predictor variables to fit an intercept in the regression model
X = sm.add_constant(X)

# Fit a linear regression model
model = sm.OLS(y, X).fit()

# Print the summary statistics for the model
print(model.summary())
```

Q1b-1 Result:

```
OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                    nan
Method:                 Least Squares   F-statistic:                       nan
Date:                Sun, 12 Mar 2023   Prob (F-statistic):                nan
Time:                        18:24:08   Log-Likelihood:                 12603.
No. Observations:                 400   AIC:                        -2.441e+04
Df Residuals:                       0   BIC:                        -2.281e+04
Df Model:                         399
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      4.245e-06        inf          0        nan        nan        nan
x1            0.0001        inf          0        nan        nan        nan
x2            0.0001        inf          0        nan        nan        nan
x3            0.0001        inf          0        nan        nan        nan
x4            0.0001        inf          0        nan        nan        nan
x5          6.92e-05        inf          0        nan        nan        nan
x6         1.067e-05        inf          0        nan        nan        nan
x7        -9.023e-05        inf         -0        nan        nan        nan
x8        -9.828e-05        inf         -0        nan        nan        nan
.
.
```

Q1b-2 Code:

```
sfs = SequentialFeatureSelector(model, n_features_to_select=10, direction='forward',
scoring='r2', cv=5)
sfs.fit(X, Y)

important_pixels = sfs.get_support(indices=True)
print("Important pixels:", important_pixels)

canvas = np.zeros((46, 56))

for idx in important_pixels:
    row = idx // 56
    col = idx % 56
    canvas[row][col] = 255

cv2.imshow("Important Pixels", canvas)
cv2.waitKey(0)
```

Q1b-2 Result:

```
Important pixels: [ 156  224 1190 1469 1983 2116 2182 2221 2399 2449]
```

Q2 Code:

```
import pandas as pd
import numpy as np
import os
from sklearn.linear_model import LinearRegression

# Load the volcano dataset
notebook_path = os.path.abspath("Q2.ipynb")
data_file = os.path.join(os.path.dirname(notebook_path), "Volcano.csv")

volcano = pd.read_csv(data_file, header=None)

# Set the grid coordinates
x1 = np.arange(1, 88)
x2 = np.arange(1, 62)

# Initialize the starting point
current_point = np.array([87, 1])

# Set the domain size for each regression model
domain_size = 5
```

```
# Iterate until convergence
while True:
    # Get the current coordinates
    i = current_point[0] - 1
    j = current_point[1] - 1

    # Extract the domain for regression
    i_start = int(max(i - domain_size, 0))
    i_end = int(min(i + domain_size, 86))
    j_start = int(max(j - domain_size, 0))
    j_end = int(min(j + domain_size, 60))
    X1, X2 = np.meshgrid(x1[i_start:i_end+1], x2[j_start:j_end+1])
    Y = volcano.values[j_start:j_end+1, i_start:i_end+1].ravel()

    # Fit the regression model
    model = LinearRegression()
    X = np.column_stack((X1.ravel(), X2.ravel()))
    model.fit(X, Y)

    # Find the direction of improvement
    gradient = model.coef_
    direction = np.sign(gradient)

    # Update the current point
    new_point = current_point + direction
    if np.all(new_point == current_point):
        # Converged to a local maximum
        break
    else:
        current_point = new_point

# Print the final result
print("The highest point is:", tuple(current_point))
```

Q2 Code Result:

```
The highest point is: (20,31)
```

Q3:

為了模擬具有兩個預測變量的 multiple regression，我們可以使用 Python 中 sklearn.datasets 中的 make_regression 函數。下面是使用 statsmodels 包生成數據的 code：

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_regression
import statsmodels.api as sm

# Generate data
X, y = make_regression(n_samples=50000, n_features=2, noise=10)
```

```
# Add intercept to X
X = sm.add_constant(X)

# Fit linear regression model
model = sm.OLS(y, X).fit()

# Print model summary
print(model.summary())
```

Result:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.988
Model:                            OLS   Adj. R-squared:                  0.988
Method:                 Least Squares   F-statistic:                 2.040e+06
Date:                Sun, 12 Mar 2023   Prob (F-statistic):               0.00
Time:                        18:55:22   Log-Likelihood:            -1.8605e+05
No. Observations:               50000   AIC:                         3.721e+05
Df Residuals:                   49997   BIC:                         3.721e+05
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0040      0.045     -0.089      0.929      -0.092       0.084
x1            70.4707      0.045   1573.566      0.000      70.383      70.558
x2            56.4484      0.045   1263.579      0.000      56.361      56.536
==============================================================================
Omnibus:                        0.430   Durbin-Watson:                   2.013
Prob(Omnibus):                  0.807   Jarque-Bera (JB):                0.420
Skew:                           0.006   Prob(JB):                        0.811
Kurtosis:                       3.007   Cond. No.                         1.00
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

這將生成一個 50,000 個樣本、兩個預測變量和正態分佈噪聲的數據集。然後，我們向預測變量添加一個 intercept，並使用來自 statsmodels 的 OLS 函數擬合線性回歸模型。最後，我們 print 出模型的摘要。

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_regression

# Generate data
X, y = make_regression(n_samples=50000, n_features=2, noise=10)

# Add intercept to X
X = np.c_[np.ones(X.shape[0]), X]

# Define learning rate and number of iterations
learning_rate = 0.01
n_iterations = 1000

# Initialize coefficients
beta = np.zeros(X.shape[1])
```

```
# Define error function and gradient function
def error(X, y, beta):
    return np.mean((y - X @ beta)**2)

def gradient(X, y, beta):
    return -2*np.mean((y - X @ beta)*X.T, axis=1)

# Perform gradient descent
errors = []
betas = [beta]
for i in range(n_iterations):
    beta = beta - learning_rate * gradient(X, y, beta)
    errors.append(error(X, y, beta))
    betas.append(beta)

# Print coefficients
print("Coefficients:", beta)
print("Regression package:", model.params)
print("Gradient descent:", beta)
```

```
Coefficients: [-2.88158447e-02  5.09783802e+01  5.04664600e+00]
Regression package: [-3.96912328e-03  7.04706923e+01  5.64484188e+01]
Gradient descent: [-2.88158447e-02  5.09783802e+01  5.04664600e+00]
```

我們先使用 make_regression 函數生成數據並向預測變量添加 intercept。然後我們定義梯度下降算法的學習率和迭代次數，並將係數初始化為零。我們還定義了誤差函數和梯度函數，它們分別計算均方誤差及其梯度。梯度函數使用均方誤差梯度的矩陣形式。然後我們進入一個循環，為指定的迭代次數執行梯度下降。在每次迭代中，我們使用梯度下降規則更新係數，計算誤差，並將誤差和係數附加到它們各自的列表中。最後，我們打印通過梯度下降獲得的最終係數。為了比較線性回歸包和梯度下降得到的結果，我們可以print 出兩種方法得到的係數。