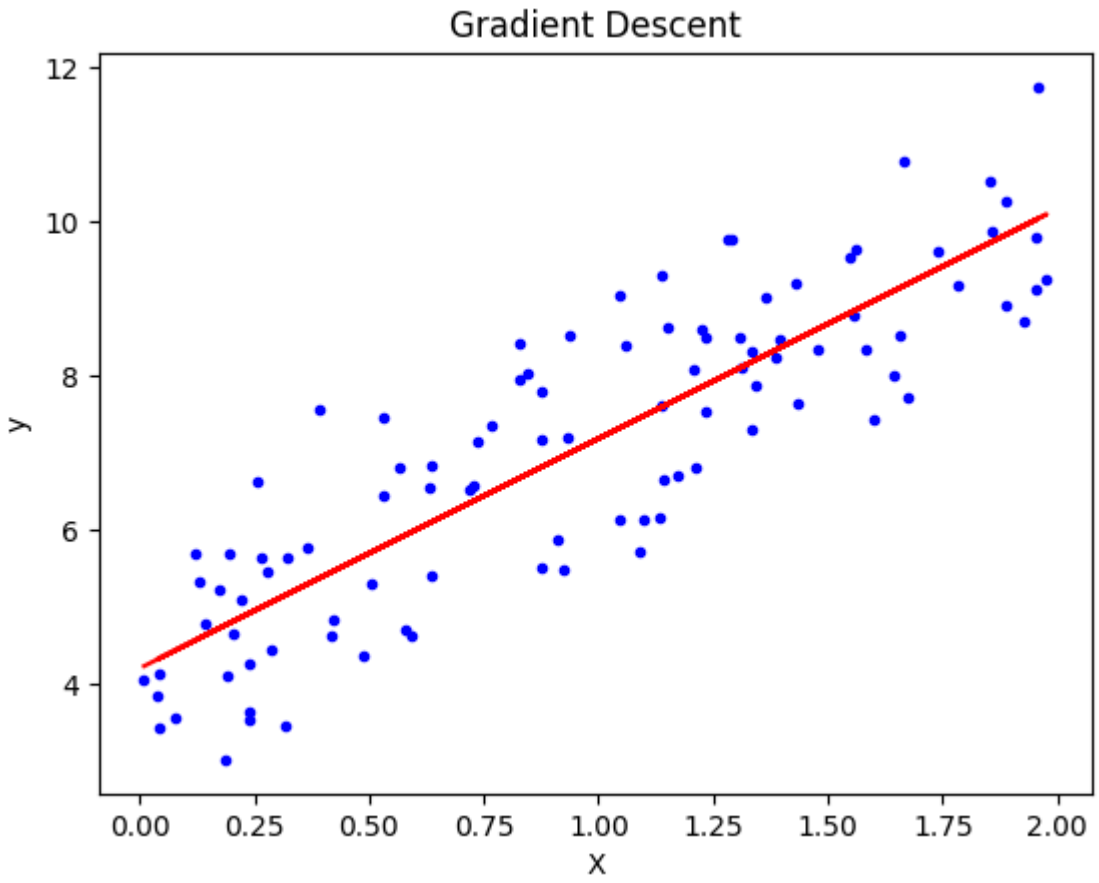
Start coding or generate with AI.

13

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
X_b = np.c_[np.ones((100, 1)), X]

def gradient_descent(X, y, lr=0.01, n_iterations=1000):
    m = len(X)
    theta = np.random.randn(2, 1)
    for iteration in range(n_iterations):
        gradients = 2/m * X.T.dot(X.dot(theta) - y)
        theta -= lr * gradients
    return theta
theta = gradient_descent(X_b, y)
plt.plot(X, y, 'b.')
plt.plot(X, X_b.dot(theta), 'r-')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Gradient Descent')
plt.show()

print("Theta:", theta)
```

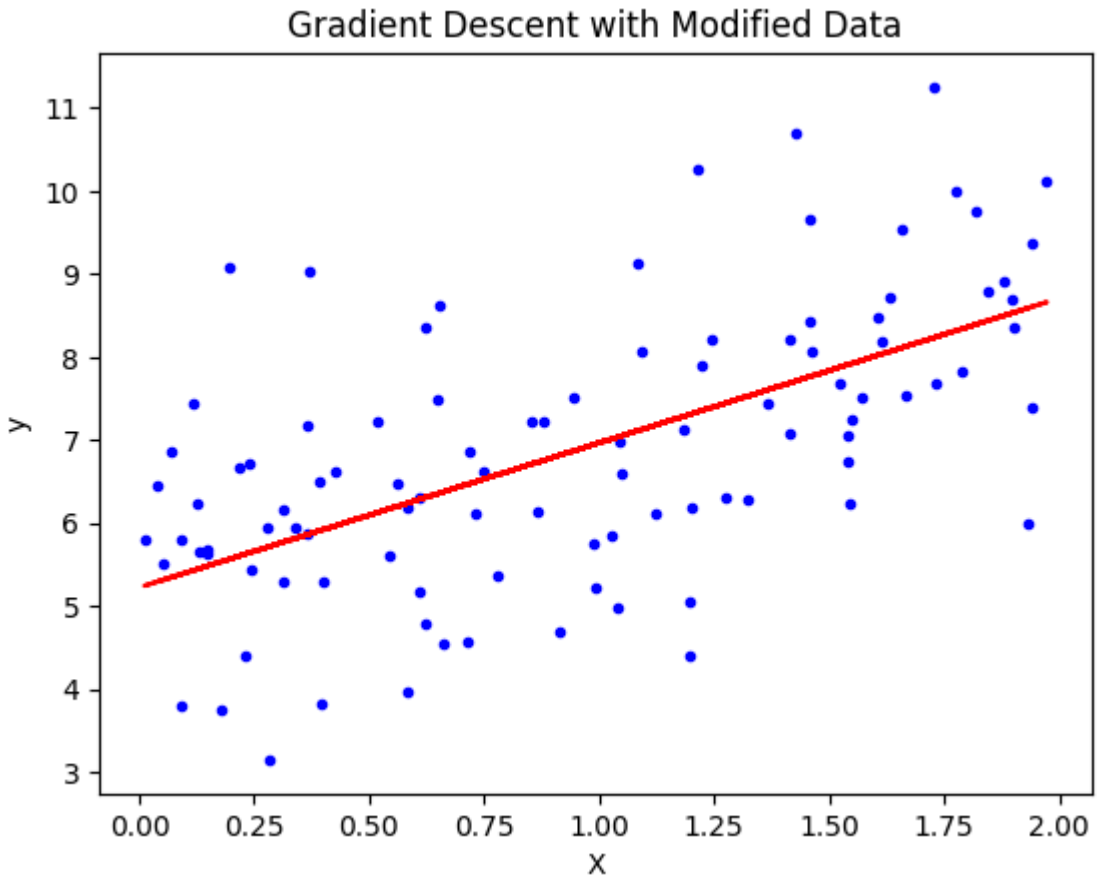



```
Theta: [[4.20607718]
 [2.98273036]]
```

14

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 5 + 2 * X + np.random.randn(100, 1) * 1.5
X_b = np.c_[np.ones((100, 1)), X]
def gradient_descent(X, y, lr=0.01, n_iterations=1000):
    m = len(X)
    theta = np.random.randn(2, 1)
    for iteration in range(n_iterations):
        gradients = 2/m * X.T.dot(X.dot(theta) - y)
        theta -= lr * gradients
    return theta
theta = gradient_descent(X_b, y)
plt.plot(X, y, 'b.')
plt.plot(X, X_b.dot(theta), 'r-')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Gradient Descent with Modified Data')
plt.show()

print("Theta:", theta)
```




```
Theta: [[5.2264107 ]
 [1.74013906]]
```

15

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread(r'/content/dora.jpeg')
b, g, r = cv2.split(img)
rgb_img = cv2.merge([r, g, b])

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

kernel = np.ones((2, 2), np.uint8)
closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=2)
sure_bg = cv2.dilate(closing, kernel, iterations=3)
dist_transform = cv2.distanceTransform(closing, cv2.DIST_L2, 3)
ret, sure_fg = cv2.threshold(dist_transform, 0.1 * dist_transform.max(), 255, 0)
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

ret, markers = cv2.connectedComponents(sure_fg)
markers = markers + 1
markers[unknown == 255] = 0
markers = cv2.watershed(img, markers)
img[markers == -1] = [255, 0, 0]

plt.subplot(211)
plt.imshow(cv2.cvtColor(rgb_img, cv2.COLOR_BGR2RGB))
plt.title('Input Image')
plt.xticks([]), plt.yticks([])

plt.subplot(212)
plt.imshow(thresh, cmap='gray')
plt.title("Otsu's Binary Threshold")
plt.xticks([]), plt.yticks([])

plt.imsave(r'/content/dora.jpeg', thresh, cmap='gray')

plt.tight_layout()
plt.show()
```

16

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, color
from skimage.filters import sobel
from skimage.segmentation import watershed
from skimage.measure import label
image = color.rgb2gray(data.astronaut())
elevation_map = sobel(image)
markers = np.zeros_like(image, dtype=np.int32)
markers[image < 0.3] = 1
markers[image > 0.7] = 2
segmentation = watershed(elevation_map, markers, mask=image)
labels = label(segmentation)

fig, axs = plt.subplots(1, 2, figsize=(12, 6))
axs[0].imshow(image, cmap='gray')
axs[0].set_title('Original Image')
axs[0].axis('off')
axs[1].imshow(color.label2rgb(labels, image=image))
axs[1].set_title('Segmented Image')
axs[1].axis('off')
plt.show()
```