02_linear_regression_grad_desc

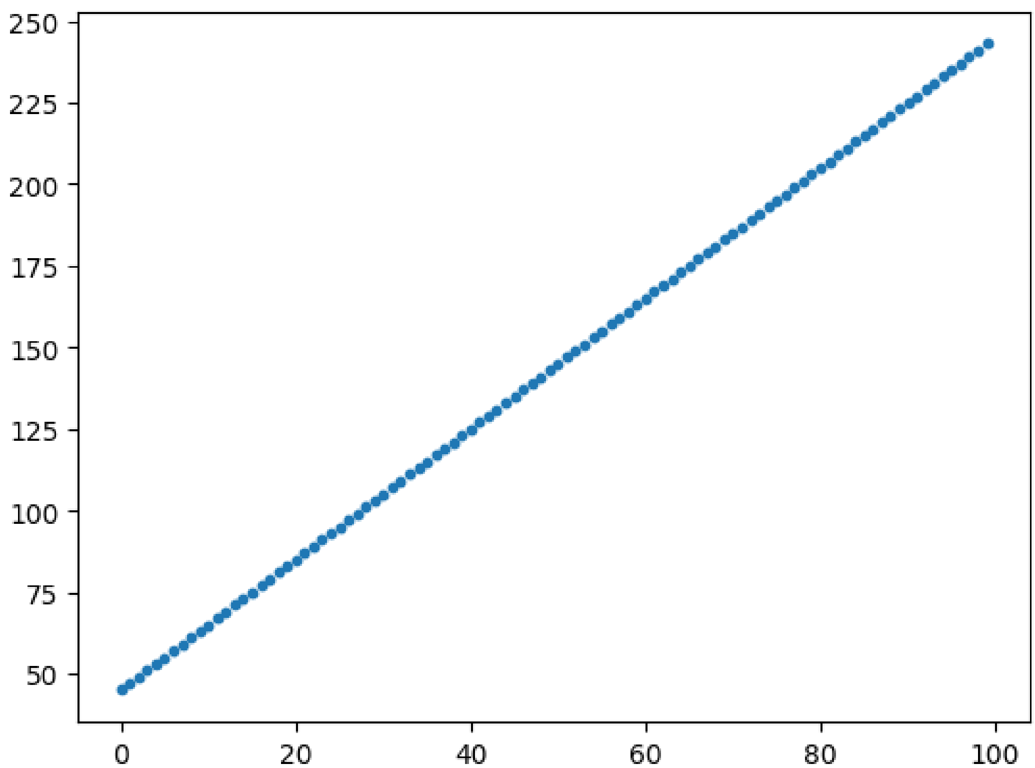
March 10, 2025

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
def linear_function(a, x, b):
    return a*x + b
```

[3]:
```
a = 2
x = np.arange(100)
b = 45
y = linear_function(a, x, b)
```

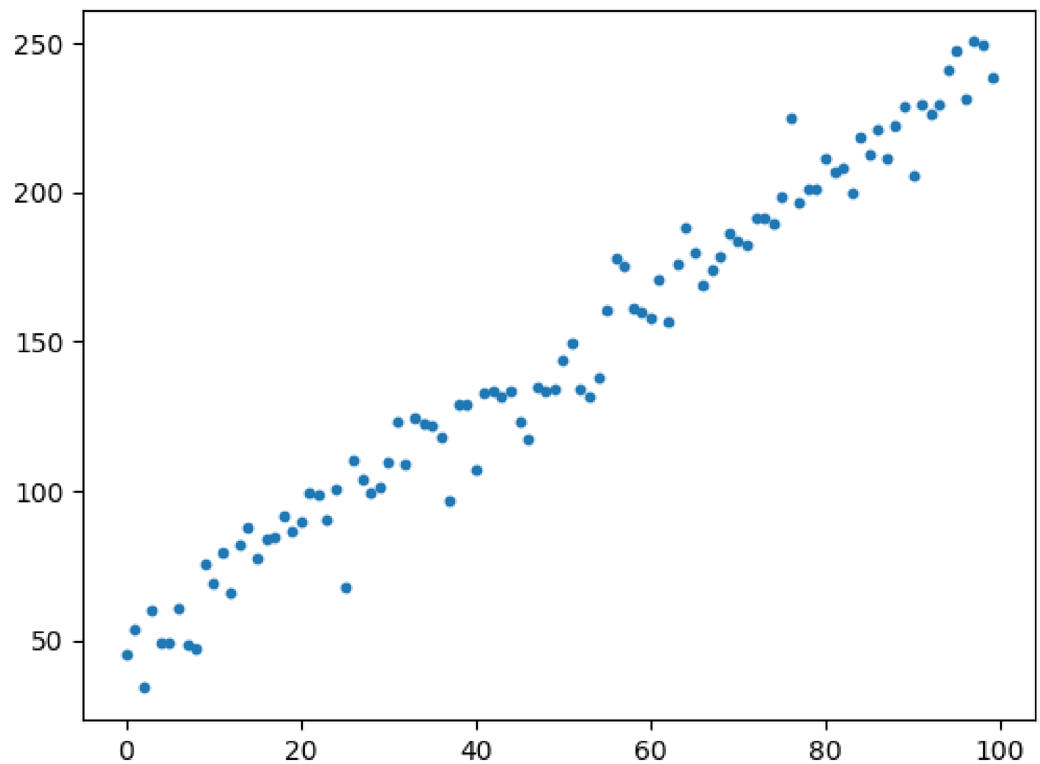
[4]:
```
plt.scatter(x, y, s=10)
plt.show()
```



[5]:
```
# https://medium.com/@ms_somanna/
↪guide-to-adding-noise-to-your-data-using-python-and-numpy-c8be815df524

noise = np.random.normal(0, 10, len(x))
y_noisy = y + noise
```

[6]:
```
plt.scatter(x, y_noisy, s=10)
plt.show()
```



1 Gradient Descent Derivation

The goal of gradient descent is to minimize the Mean Squared Error (MSE) loss function for a linear model:

$$y = \beta_1 x + \beta_0$$

1.1 Loss Function

The MSE loss function is defined as:

$$J(\beta_0, \beta_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (\beta_1 x_i + \beta_0))^2$$

where: - n is the number of data points, - y_i is the actual observed value, - x_i is the feature (independent variable), - β_0 is the intercept, - β_1 is the slope.

1.2 Gradient Computation

To update the parameters β_0 and β_1, we compute the partial derivatives of $J(\beta_0, \beta_1)$ with respect to each parameter.

1.2.1 Partial Derivative with Respect to β_1

$$\frac{\partial J}{\partial \beta_1} = \frac{2}{n} \sum_{i=1}^{n} (y_i - (\beta_1 x_i + \beta_0))(-x_i) = -\frac{2}{n} \sum_{i=1}^{n} x_i (y_i - (\beta_1 x_i + \beta_0))$$

1.2.2 Partial Derivative with Respect to β_0

$$\frac{\partial J}{\partial \beta_0} = \frac{2}{n} \sum_{i=1}^{n} (y_i - (\beta_1 x_i + \beta_0))(-1) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - (\beta_1 x_i + \beta_0))$$

1.3 Gradient Descent Updates

Using gradient descent, we update the parameters iteratively as follows:

$$\beta_1 := \beta_1 - \alpha \frac{\partial J}{\partial \beta_1}$$

$$\beta_0 := \beta_0 - \alpha \frac{\partial J}{\partial \beta_0}$$

where α is the learning rate, controlling the step size in the descent process.

[7]:
```
def gradient_descent(x, y, learning_rate=0.0001, epochs=100000):
    n = len(x)
    beta_0, beta_1 = 0, 0  # Initialize parameters

    for _ in range(epochs):
        y_pred = beta_1 * x + beta_0
        error = y_pred - y

        grad_beta_1 = (2/n) * np.sum(error * x)
        grad_beta_0 = (2/n) * np.sum(error)

        beta_1 -= learning_rate * grad_beta_1
        beta_0 -= learning_rate * grad_beta_0

    return beta_1, beta_0
```

```python
# Run gradient descent
beta_1, beta_0 = gradient_descent(x, y_noisy)
print(f"Estimated beta_1: {beta_1}, Estimated beta_0: {beta_0}")
```

```
Estimated beta_1: 1.9844676182020553, Estimated beta_0: 46.45394715662439
```

[8]:
```python
plt.scatter(x, y_noisy, s=10)
plt.plot(beta_0 + beta_1 * x)
plt.show()
```