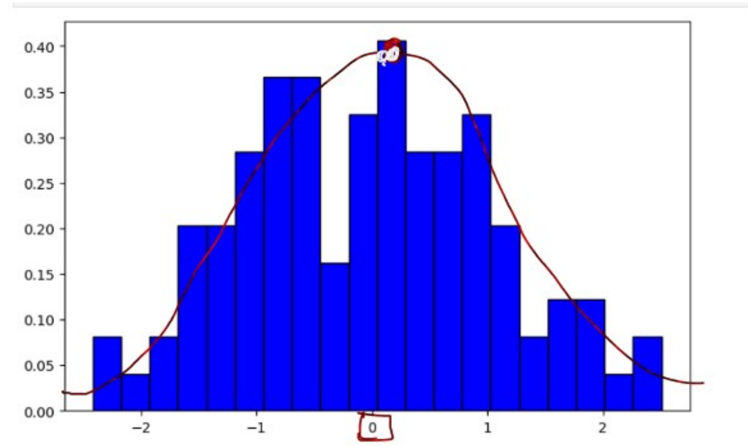
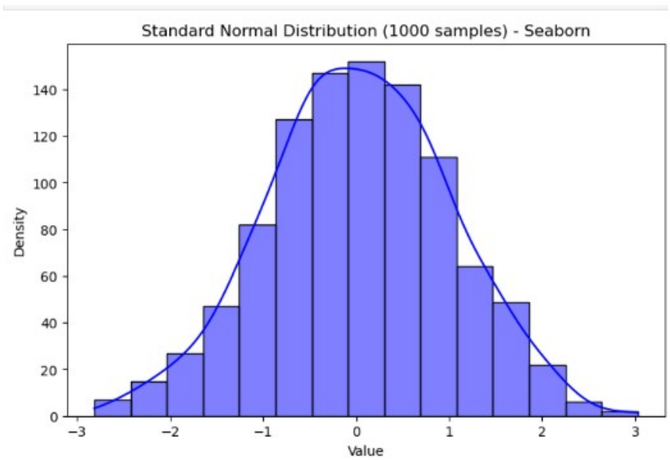
GDA Code Explanation

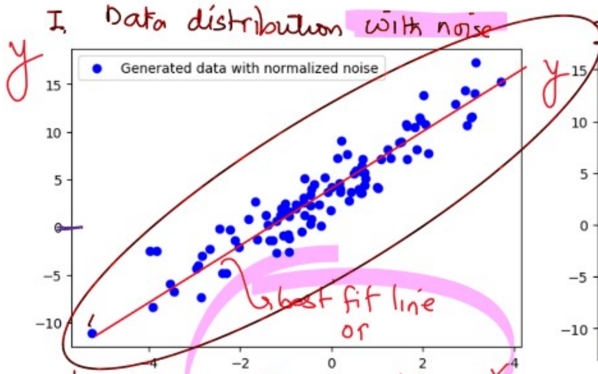
27 September 2025 11:55



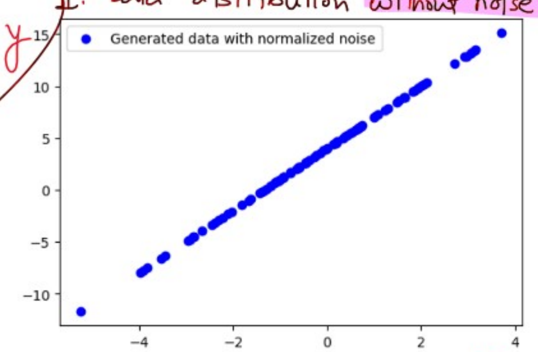


I. Data distribution with noise

II. Data distribution without noise



best fit line or regression line



x

need GDA

do you need GDA

No.

since the distribution is not around $y=x$ hence definitely some bias.

number of data points to be generated

```python
def generate_data( n_samples = 100, noise = 0.1, seed =42):
    """
    Generate random samples (synthetic) linear data: y=4 + 3*X + noise
    """
    np.random.seed(seed) #to ensure the reproducibility of the same random data -->to fix the random numbers generated
    X = 2 * np.random.randn(n_samples, 1) #generates random numbers from `standard normal distribution`
    y = 4 + 3*X + noise*np.random.randn(n_samples, 1) # Creates a random distribution around a line having some random noise added to it as well

    return X, y
```

ensures reproducibility

constant noise → 0.1 (default)

0.1 * (100 random values coming from std. normal distribution)

2 * random std. normal values

y → 4 + 3 * X + noise

→ 4 + ( ) + ( )

100 values

$$y = 4 + 3x$$

4 → intercept (bias)

3 → coefficient / slope (weight)

random values

same

```python
    X = 2 * np.random.randn(n_samples, 1) #generates random numbers from `standard normal distribution`
    y = 4 + 3*X + noise*np.random.randn(n_samples, 1) # Creates a random distribution around a line having so

    return X, y
```

```python
generate_data( n_samples =10, noise =0, seed =42)
```

```
(array([[ 0.99342831],
        [-0.2765286 ],
        [ 1.29537708],
        [ 3.04605971],
        [-0.46830675],
        [-0.46827391],
        [ 3.15842563],
        [ 1.53486946],
        [-0.93894877],
        [ 1.08512009]]),
 array([[ 6.98028492],
```

random (same)

X

$x_1 = 0.9934$

$y_1 = 4 + 3 * 0.9934 + 0$

y

y

4 + 3*0.99342831 = 6.98028493

Mean Squared Error - Cost Function for regression problems

For m training examples:

$$\underbrace{J}_{\text{cost function}}(\overset{\text{weight}}{w},\overset{\text{bias}}{b}) = \frac{1}{2m}\sum_{i=1}^{m}\Big(\overbrace{\underbrace{\hat{y}^{(i)}}_{\text{Predicted}} - \underbrace{y^{(i)}}_{\text{actual}}}^{\text{error ①}}\Big)^{2}$$

$\frac{2}{2m}[\,\_\_\,]$

Mean Squared Error

③ ← ② ← ①

$(P-A)^2$

$(\text{error})^2$ → why square ??

- ⇒ to prevent positive and negative errors cancelling / nullifying each other
  - ↓
- ⇒ squaring makes the error +ve
- → penalizes large errors more strongly than small errors

where:

- $y^{(i)}$ = actual output for sample i.
- $\hat{y}^{(i)}$ = predicted output.
- w, b = parameters (weights, bias).
- m = number of samples.
- i → ith row / sample

→ Division by m gives the mean / avg. making it independent of dataset size

↓

to get the avg. model error.

→ Factor $\frac{1}{2}$ is to simplify the derivative output (2 cancels when differentiating)

$y = x^2$	$y = \frac{1}{2}x^2$
$\frac{dy}{dx} = 2x$	$\frac{dy}{dx} = \frac{1}{\cancel{2}}(\cancel{2}x) = x$

Task

Do the below derivations:

$$\left.\begin{aligned}\frac{\partial J}{\partial w} &= \frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)\cdot x^{(i)} \\ \frac{\partial J}{\partial b} &= \frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)\end{aligned}\right\}\ \text{gradient}$$

[Cost Function] → MSE

[Cost Function] → MSE