Simple Linear Regression (SLR)

Formula :

y = w * x + b

- y is the predicted **output**
- x is the **input**
- w is the **weight** or **slope** of the line
- b is the **bias**

MSE = 1/N   i=1∑N   (y[i]   −yhat[i]) ^ 2

- N is the number of data points.
- y[i] is the true value for the i-th data point.
- yhat[i] = w . x[i] + b is the predicted value for the i-th data point.

```python
# 1. Simple Linear Regression (SLR):
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = np.array([2.2, 4.1, 6.1, 8.2, 10.2])

# 1. Define the model
#yhat = w + b*x

# Initialize
w = np.random.rand()
b = np.random.rand()
learning_rate = 0.01
epochs = 1000

# Training loop
for epoch in range(epochs):
  yhat = b + w*x
  # 2. Define the loss
  loss = np.mean((yhat - y)**2)
  # 3. Fit the model
  dw = np.mean(2*(yhat - y)*x)
  db = np.mean(2*(yhat - y))
  w = w - learning_rate*dw
  b = b - learning_rate*db

  if epoch % 10 == 0:
    print(f"Epoch {epoch}, Loss: {loss}, w: {w}, b: {b}")
```

Logistic Regression:

Compute a weighted sum of input features, z = w * x + b, and apply the sigmoid function, yhat = 1 / 1+e^−z. The model is trained using cross-entropy loss to minimize error.

```python
# 2. Logistic Regression:
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = np.array([0, 1, 0, 0, 1])

# 1. Define the model
#yhat = w + b*x

def sigmoid(z):
    return 1/(1+np.exp(-z))

# Initialize
w = np.random.rand()
b = np.random.rand()
learning_rate = 0.01
epochs = 1000

# Training loop
for epoch in range(epochs):
    yhat = sigmoid( b + w*x )
    # 2. Define the loss
    loss = -np.mean(( y*np.log(yhat) )+ ( (1-y)*np.log(1 - yhat) ))
    # 3. Fit the model
    dw = np.mean(2*(yhat - y)*x)
    db = np.mean(2*(yhat - y))
    w = w - learning_rate*dw
    b = b - learning_rate*db

    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {loss}, w: {w}, b: {b}")
```

Multiple Linear Regression (MLR):

MLR handles multiple input features by computing a weighted sum of all input features: yhat = w[1] * x[1] + w[2] * x[2]+...+ w[n] * x[n] + b

```python
# 3. Multiple Linear Regression (MLR):
import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 0]])
y = np.array([2.2, 4.1, 6.1, 8.2, 10.2])

# 1. Define the model
#yhat = w1 * x1 + w2 * x2 + b => W . X + b

# Initialize
w = np.random.rand(x.shape[1])
print(w)
b = np.random.rand()
learning_rate = 0.01
epochs = 1000

# Training loop
for epoch in range(epochs):
  yhat = np.dot(x, w) + b
  # 2. Define the loss (M.S.E.)
  loss = np.mean((yhat - y)**2)
  # 3. Fit the model
  dw = np.mean(2*(yhat - y)[:, np.newaxis] * x, axis=0)
  db = np.mean(2*(yhat - y))
  w = w - learning_rate*dw
  b = b - learning_rate*db

  if epoch % 10 == 0:
    print(f"Epoch {epoch}, Loss: {loss}, w: {w}, b: {b}")
```