Aim: To implement Stochastic Gradient Descent

```
import numpy as np
def sgd(
    gradient, x, y, start, learn_rate=0.1, batch_size=1, n_iter=50,
    tolerance=1e-06, dtype="float64", random_state=None
):
    # Checking if the gradient is callable
    if not callable(gradient):
        raise TypeError("'gradient' must be callable")
    # Setting up the data type for NumPy arrays
    dtype_ = np.dtype(dtype)
    # Converting x and y to NumPy arrays
    x, y = np.array(x, dtype=dtype_), np.array(y, dtype=dtype_)
    n_obs = x.shape[0]
    if n_obs != y.shape[0]:
        raise ValueError("'x' and 'y' lengths do not match")
    xy = np.c_[x.reshape(n_obs, -1), y.reshape(n_obs, 1)]
    # Initializing the random number generator
    seed = None if random_state is None else int(random_state)
    rng = np.random.default_rng(seed=seed)
    # Initializing the values of the variables
    vector = np.array(start, dtype=dtype_)
    # Setting up and checking the learning rate
    learn_rate = np.array(learn_rate, dtype=dtype_)
    if np.any(learn_rate <= 0):
        raise ValueError("'learn_rate' must be greater than zero")
    # Setting up and checking the size of minibatches
    batch_size = int(batch_size)
    if not 0 < batch_size <= n_obs:
        raise ValueError(
            "'batch_size' must be greater than zero and less than "
            "or equal to the number of observations"
        )
    # Setting up and checking the maximal number of iterations
    n_iter = int(n_iter)
    if n_iter <= 0:
        raise ValueError("'n_iter' must be greater than zero")
    # Setting up and checking the tolerance
    tolerance = np.array(tolerance, dtype=dtype_)
    if np.any(tolerance <= 0):
        raise ValueError("'tolerance' must be greater than zero")
    # Performing the gradient descent loop
    for _ in range(n_iter):
        # Shuffle x and y
        rng.shuffle(xy)
        # Performing minibatch moves
        for start in range(0, n_obs, batch_size):
            stop = start + batch_size
            x_batch, y_batch = xy[start:stop, :-1], xy[start:stop, -
```

```
1:]
            # Recalculating the difference
            grad = np.array(gradient(x_batch, y_batch, vector),
dtype_)
            print("gradient ", grad)
            diff = -learn_rate * grad

            # Checking if the absolute difference is small enough
            if np.all(np.abs(diff) <= tolerance):
                break

            # Updating the values of the variables
            vector += diff
        print("epoch no : ",_, "  ",vector ,"  ",diff)
    return vector if vector.shape else vector.item()

x = np.array([5, 15, 25, 35, 45, 55])
y = np.array([5, 20, 14, 32, 22, 38])

def ssr_gradient(x, y, b):
    res = b[0] + b[1] * x - y
    return res.mean(), (res * x).mean()


def gradient_descent(
    gradient, x, y, start, learn_rate=0.1, n_iter=50, tolerance=1e-06
):
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * np.array(gradient(x, y, vector))
        if np.all(np.abs(diff) <= tolerance):
            break
        vector += diff
        print("epoch no : ",_, "  ",vector ," ",diff)
    return vector

gradient_descent(
     ssr_gradient, x, y, start=[0.5, 0.5], learn_rate=0.0008,
      n_iter=10
)
```

```
epoch no :  0    [0.50506667 0.66133333]   [0.00506667 0.16133333]
epoch no :  1    [0.50625728 0.66874062]   [0.00119061 0.00740729]
epoch no :  2    [0.50726917 0.66905772]   [0.00101189 0.0003171 ]
epoch no :  3    [0.50827263 0.66904823]   [ 1.00346603e-03 -
9.48731962e-06]
epoch no :  4    [0.50927552 0.66902371]   [ 1.00289095e-03 -
2.45259263e-05]
epoch no :  5    [0.5102782  0.66899849]   [ 1.00267726e-03 -
2.52139261e-05]
epoch no :  6    [0.51128068 0.66897325]   [ 1.00248026e-03 -
```

```
2.52409042e-05]
epoch no :  7    [0.51228296 0.66894802]   [ 1.00228405e-03 -
2.52374350e-05]
epoch no :  8    [0.51328505 0.66892278]   [ 1.00208792e-03 -
2.52325642e-05]
epoch no :  9    [0.51428694 0.66889756]   [ 1.00189183e-03 -
2.52276298e-05]

array([0.51428694, 0.66889756])


sgd(
     ssr_gradient, x, y, start=[0.5, 0.5], learn_rate=0.0008,
     batch_size=3, n_iter=10, random_state=0
)

gradient  [  -8.33333333 -355.        ]
gradient  [  1.82666667 167.17777778]
epoch no :  0    [0.50520533 0.65025778]    [-0.00146133 -0.13374222]
gradient  [ -2.740928   -27.76340148]
gradient  [ 0.10181386 29.41285707]
epoch no :  1    [0.50731662 0.64893821]    [-8.14510899e-05 -
2.35302857e-02]
gradient  [ -0.10610067 -12.34338434]
gradient  [ -3.296856   -24.14065244]
epoch no :  2    [0.51003899 0.67812544]    [0.00263748 0.01931252]
gradient  [  -6.05766123 -138.90427584]
gradient  [  8.73358776 371.11597989]
epoch no :  3    [0.50789825 0.49235608]    [-0.00698687 -0.29689278]
gradient  [  -3.64393177 -190.7973226 ]
gradient  [  -6.66429775 -155.17973513]
epoch no :  4    [0.51614483 0.76913773]    [0.00533144 0.12414379]
gradient  [  3.64171373 205.04775736]
gradient  [  -5.32528375 -199.81418802]
epoch no :  5    [0.51749169 0.76495087]    [0.00426023 0.15985135]
gradient  [ 1.84060838 76.21567881]
gradient  [1.02216290e-01 1.06697314e+02]
epoch no :  6    [0.51593743 0.61862048]    [-8.17730321e-05 -
8.53578513e-02]
gradient  [  -5.01855067 -116.77421148]
gradient  [  2.77468018 117.91646259]
epoch no :  7    [0.51773253 0.61770667]    [-0.00221974 -0.09433317]
gradient  [-1.76528952 41.31176656]
gradient  [  -5.06899354 -216.69806737]
epoch no :  8    [0.52319995 0.75801572]    [0.00405519 0.17335845]
gradient  [  5.80692618 233.97734521]
gradient  [  -9.50226121 -300.35678512]
epoch no :  9    [0.52615622 0.81111927]    [0.00760181 0.24028543]

array([0.52615622, 0.81111927])
```