Writing my own class

```
class My_Stochastic Gradient Descent:
    def __init__(self, learningrate, epochs):
        self.coef_ = None
        self.intercept = None
        self.learningrate = learningrate
        self.epochs = epochs

    def fit(self, X_train, Y_train):
        self.intercept = 0
        self.coef_ = np.ones(X_train[1])
        for i in range(self.epochs):
            for j in range(X_train.shape[0]):
                # Calculating the Y_predicted, Y_intercept and random row..
#random      idx = np.random.randint(0, X_train.shape[0])
row selection
                np.dot(
#Y hat       Y_Predicted = X_train[idx], self.coef_) + self.intercept_
#Y_intercept intercept_der = -2 * np.dot(Y_train[idx] - Y_Predicted)
derivate                OR
            intercept_der = -2 * (Y_train[idx], Y_Predicted)
#Y_Upda-     self.intercept = (self.intercept - (self.learningrate * intercept_der))
 tion
#coef derivate coef_der = -2 * np.dot((Y_train[idx] - Y_Predicted), X_train[idx])
#X Updation  self.coef = (self.coef - (self.learningrate * coef_der))
```

Spiral

Date.....

```
    print(self.coef, self.intercept)
def predict(self, X_test):

    return ~~self~~ np.dot(self.coef_, X_train) + self.intercept_
    #As per the formula :- y = mx + b
```