Lets import some librires

In [1]:

```
from random import seed
from random import randrange
from csv import reader
from math import sqrt
```

Lets create a class for linear regression

In [2]:

```
class Wine_quality_prediction:

    #load csv file
    def load_csv(self,filename):
        dataset=list()
        with open(filename,'r') as file:
            csv_reader = reader(file,delimiter=';')
            next(csv_reader)
            for row in csv_reader:
                if not row:
                    continue
                dataset.append(row)
        return dataset


    #convert string column to float
    def str_column_to_float(self,dataset,column):
        for row in dataset:
            row[column]=float(row[column].strip())


    #Find min and max value for each column
    def dataset_minmax(self,dataset):
        minmax=list()
        for i in range(len(dataset[0])):
            col_values=[row[i] for row in dataset]
            value_min=min(col_values)
            value_max=max(col_values)
            minmax.append([value_min,value_max])
        return minmax


    #Rescale dataset column to the range 0 and 1
    def normalize_dataset(self,dataset,minmax):
        for row in dataset:
            for i in range(len(row)):
                row[i]=(row[i]-minmax[i][0])/(minmax[i][1]-minmax[i][0])


    #Split a dataset into K-folds
    def cross_validation_split(self,dataset, n_folds):
        dataset_split = list()
        dataset_copy = list(dataset)
        fold_size = int(len(dataset) / n_folds)
        for i in range(n_folds):
            fold = list()
            while len(fold) < fold_size:
                index = randrange(len(dataset_copy))
                fold.append(dataset_copy.pop(index))
            dataset_split.append(fold)
        return dataset_split


    #calculate root mean square
    def rmse_metric(self,actual,predicted):
        sum_error=0.0
        for i in range(len(actual)):
```

```
            predicted_error=predicted[i] - actual[i]
            sum_error+=(predicted_error**2)
        mean_error=sum_error/ float(len(actual))
        return sqrt(mean_error)


    #Evaluate an algorithm using an cross validation split
    def evaluate_algorithm(self,dataset, algorithm, n_folds, *args):
        folds = self.cross_validation_split(dataset, n_folds)
        scores = list()
        for fold in folds:
            train_set = list(folds)
            train_set.remove(fold)
            train_set = sum(train_set, [])
            test_set = list()
            for row in fold:
                row_copy = list(row)
                test_set.append(row_copy)
                row_copy[-1] = None
            predicted = algorithm(train_set, test_set, *args)
            actual = [row[-1] for row in fold]
            rmse = self.rmse_metric(actual, predicted)
            scores.append(rmse)
        return scores


    #Make a prediction with coefficients
    def predict(self,row,coefficients):
        yhat=coefficients[0]
        for i in range(len(row)-1):
            yhat += coefficients[i+1]*row[i]
        return yhat


    #Estimate linear regression coefficients using stochastic gradient descent
    def coefficients_sgd(self,train, l_rate, n_epoch):
        coef = [0.0 for i in range(len(train[0]))]
        for epoch in range(n_epoch):
            for row in train:
                yhat = self.predict(row, coef)
                error = yhat - row[-1]
                coef[0] = coef[0] - l_rate * error
                for i in range(len(row)-1):
                    coef[i + 1] = coef[i + 1] - l_rate * error * row[i]
                # print(l_rate, n_epoch, error)
        return coef

    # Linear Regression Algorithm With Stochastic Gradient Descent
    def linear_regression_sgd(self,train, test, l_rate, n_epoch):
        predictions = list()
        coef = self.coefficients_sgd(train, l_rate, n_epoch)
        for row in test:
            yhat = self.predict(row, coef)
            predictions.append(yhat)
        return(predictions)
```

Linear Regression on wine quality dataset for white wine

In [3]:

```
dd=Wine_quality_prediction()
seed(1)

# load and prepare data
filename = 'winequality-white.csv'
dataset = dd.load_csv(filename)
for i in range(len(dataset[0])):
    dd.str_column_to_float(dataset, i)

# normalize
minmax = dd.dataset_minmax(dataset)
dd.normalize_dataset(dataset, minmax)

# evaluate algorithm
n_folds = 5
l_rate = 0.001
n_epoch = 500
scores = dd.evaluate_algorithm(dataset, dd.linear_regression_sgd, n_folds, l_rate, n_epoch)

print('Scores: %s' % scores)
print('Mean RMSE: %.3f' % (sum(scores)/float(len(scores))))
```

```
Scores: [0.12236392134227936, 0.12960987850071848, 0.12574808693656167, 0.12
851414761930938, 0.12433417974178929]
Mean RMSE: 0.126
```

Linear Regression on wine quality dataset for red wine

In [4]:

```
dd=Wine_quality_prediction()
seed(1)

# load and prepare data
filename = 'winequality-red.csv'
dataset = dd.load_csv(filename)
for i in range(len(dataset[0])):
    dd.str_column_to_float(dataset, i)

# normalize
minmax = dd.dataset_minmax(dataset)
dd.normalize_dataset(dataset, minmax)

# evaluate algorithm
n_folds = 5
l_rate = 0.001
n_epoch = 500
scores = dd.evaluate_algorithm(dataset, dd.linear_regression_sgd, n_folds, l_rate, n_epoch)

print('Scores: %s' % scores)
print('Mean RMSE: %.3f' % (sum(scores)/float(len(scores))))
```

```
Scores: [0.11793622579831198, 0.1365221170785699, 0.13547687308183987, 0.135
4676437425366, 0.125794606367393]
Mean RMSE: 0.130
```

In [ ]: