1. (1%) Linear regression function by **Gradient Descent.**

```
from sys import argv
import math
import numpy as np

def F(B, C, X):
    return (C * X).sum() + B

def Loss(B, C, X):
    W = 0.0
    for x, y in X:
        Y = F(B, C, x)
        W += (y - Y) ** 2
    return W / len(X)

def Gradient(B, C, x, y):
    Y = F(B, C, x)
    return (Y - y), (Y - y) * x

def main():
    X = loadTrainingData(argv[1])
    B, C = loadCoefficient(argv[2])
    B = np.random.random() - 0.5
    C = np.random.random(C.shape) - 0.5
    TestSets = loadTestData(argv[3])

    Alpha = 0.000001
    for Iteration in range(20000):
        if Iteration % 100 == 0:
            print (Iteration, Loss(B, C, X))
        GradB = 0.0
        GradC = np.zeros((9, 18))
        for x, y in X:
            g = Gradient(B, C, x, y)
            GradB += g[0]
            GradC += g[1]
        GradB /= len(X)
        GradC /= len(X)
        B -= Alpha * GradB
```

```
        C -= Alpha * GradC

    print (Iteration+1, Loss(B, C, X))

    if len(argv) > 4:
        Filename = argv[4]
    else:
        Filename = "linear_regression.csv"
    OutputFile = open(Filename, "w")
    OutputFile.write("id,value\n")
    for (Id, X) in TestSets:
        Y = F(B, C, X)
        OutputFile.write("%s,%f\n" % (Id, Y))
    OutputFile.close()
```

2. (1%) Describe your method.

使用最近 9 個小時的所有觀測數值當作 feature，並且假設預測的 PM2.5 數值是所有 feature 的線性組合。因此 feature 就是一個 9x18 = 162 維向量，並且用 gradient descent 去搜尋最佳的線性函數係數(C)和 bias(B)。

3. (1%) Discussion on regularization.

No regularization, RMSE(public set) = 6.02327
smoothness = 0.1, RMSE(public set) = 5.93850
smoothness = 1, RMSE(public set) = 5.94504
smoothness = 10, RMSE(public set) = 6.20823
結論：感覺 regularization 的幫助並不大。

4. (1%) Discussion on learning rate.

When using vanilla gradient descent:
Learning rate = 0.000001, Iterations = 20000
Error(Loss function, training set) = 73.651549714979325

Learning rate = 0.0000001, Iterations = 20000
Error(Loss function, training set) = 181.09477263867058

Use Adagrad to tune each variable's learning rate dynamically:

```
At iteration 1400, error has converged to 71.73358799010559.
At iteration 20000, error converged to 34.584058764315643.
```

結論：

1. 用 Adagrad 收斂的速度快非常多。
2. 把 training data random shuffle 過後用 stochastic gradient descent 收斂速度雖然有加快，但是收斂到的結果會被 learning rate 影響很大；learning rate 大則沒辦法收斂到 local minimum，learning rate 小則能收斂到的 local minimum 但是 training 的速度會慢到失去使用 SGD 的好處。
3. 另外我也有試過把所有 feature 先標準化後再 train，但是對收斂的速度還有收斂到的結果的影響並不大。