Ex6: Gradient Descent - SAT

** Link: http://onlinestatbook.com/2/regression/intro.html (http://onlinestatbook.com/2/regression/intro.html) **

- Case study: "SAT and College GPA" có điểm của Trung học và Đại học cho 105 SV vào học chuyên ngành Computer Science ở một trường công lập địa phương. Chúng ta dự đoán *university GPA* nếu chúng ta biết *high school GPA* của sinh viên.

Cho dữ liệu sat.xls

- Đọc dữ liệu trên vào dataframe
- Trực quan hóa dữ liệu theo High School GPA, University GPA
- X = High School GPA đã chuyển theo định dạng chuẩn, y = University GPA
- Với y = mx + b (University GPA = m * High School GPA + b), gọi hàm tính m, b: theta = gradient_descent_2(alpha, X, y, 1000)
- Từ m, b (m = theta[1], b = theta[0]) => dự đoán University GPA theo m, b
- Trực quan hóa dữ liệu
- Với High School GPA là 2.3, 2.8,3.3, 3.8 thì University GPA lần lượt là bao nhiêu?

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
from sklearn.datasets.samples_generator import make_regression
from scipy import stats
```

In [3]:
```
data = pd.read_excel("sat.xls")
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105 entries, 0 to 104
Data columns (total 5 columns):
high_GPA    105 non-null float64
math_SAT    105 non-null int64
verb_SAT    105 non-null int64
comp_GPA    105 non-null float64
univ_GPA    105 non-null float64
dtypes: float64(3), int64(2)
memory usage: 4.2 KB
```

In [7]:
```
high_GPA = data.high_GPA.values
high_GPA.size
```

Out[7]:
```
105
```

In [8]:
```
high_GPA[0:5]
```

Out[8]:
```
array([3.45, 2.78, 2.52, 3.67, 3.24])
```

In [9]:
```
univ_GPA = data.univ_GPA.values
univ_GPA.size
```

Out[9]:
```
105
```

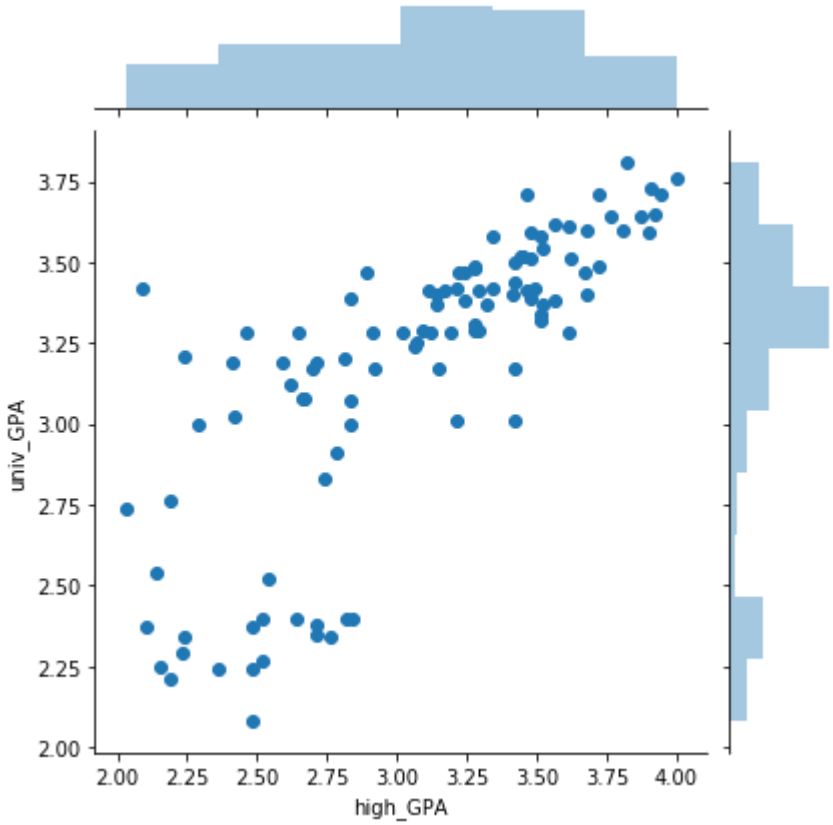
In [10]:
```
univ_GPA[0:5]
```

Out[10]:
```
array([3.52, 2.91, 2.4 , 3.47, 3.47])
```

In [12]:
```
import seaborn as sns
```

In [15]:
```
plt.figure(figsize=(12,12))
sns.jointplot(x='high_GPA', y = 'univ_GPA', data=data)
plt.show()
```

```
<Figure size 864x864 with 0 Axes>
```



In [17]:
```
from chapter4_lib import *
```

In [18]:
```
# y = mx + b
m = high_GPA.size
X = np.c_[ np.ones(m), high_GPA] # insert column
y = univ_GPA
alpha = 0.01 # learning rate
theta = gradient_descent_2(alpha, X, y, 1000)
```

```
iter 0 | J: 0.461
iter 1 | J: 0.376
iter 2 | J: 0.307
iter 3 | J: 0.253
iter 4 | J: 0.210
iter 5 | J: 0.175
iter 6 | J: 0.148
iter 7 | J: 0.126
iter 8 | J: 0.108
iter 9 | J: 0.094
iter 10 | J: 0.083
iter 11 | J: 0.074
iter 12 | J: 0.067
iter 13 | J: 0.062
iter 14 | J: 0.057
iter 15 | J: 0.053
iter 16 | J: 0.051
iter 17 | J: 0.048
iter 18 | J: 0.046
```

In [19]:
```
X[0:5]
```

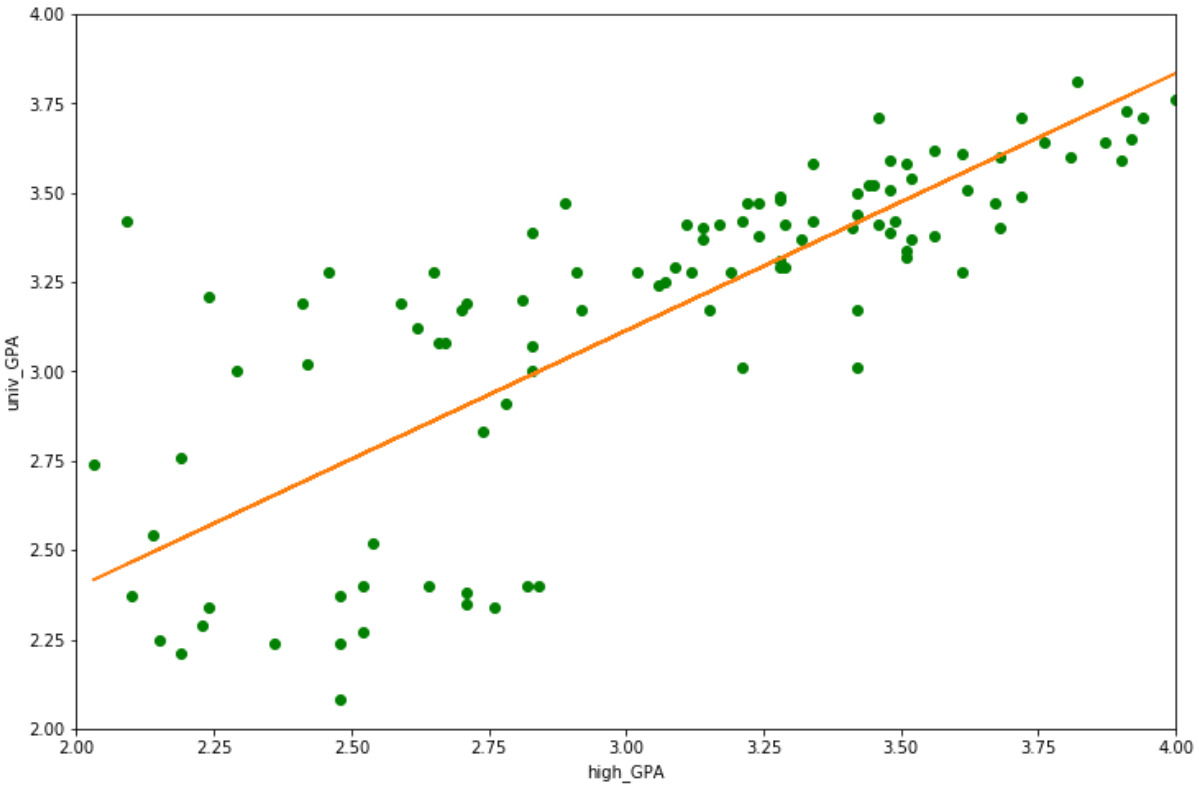
Out[19]:
```
array([[1.  , 3.45],
       [1.  , 2.78],
       [1.  , 2.52],
       [1.  , 3.67],
       [1.  , 3.24]])
```

In [20]:
```
print("m = ", theta[1], "b = ", theta[0])
```

```
m =  0.7198164351685973 b =  0.9549233210815301
```

In [21]:
```
for i in range(X.shape[1]):
    univ_GPA_predict = theta[1]* X + theta[0]
```

In [23]:
```
plt.figure(figsize=(12,8))
plt.xlim(2.0,4.0)
plt.ylim(2.0,4.0)
plt.scatter(X[:,1], univ_GPA, color="green")
plt.plot(X, univ_GPA_predict)
plt.xlabel("high_GPA")
plt.ylabel("univ_GPA")
plt.show()
```



In [25]:
```
high_GPA_new= np.array([2.3, 2.8,3.3, 3.8])
univ_GPA_new = theta[1]*high_GPA_new + theta[0]
univ_GPA_new
```

Out[25]:
```
array([2.61050112, 2.97040934, 3.33031756, 3.69022577])
```

In [ ]: