OOP Concept in Machine Learning 2

Simple Linear Regression

- `Gradient Descent` Python Implementation
- `Scipy` Implementation
- `Scikit-Learn` Implementation
- `Statsmodel` Implementation
- Multi-step visual of Gradient Descent
- Animating the Gradient Descent

Importing needed libraries

```
[ ]
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from google.colab import files
uploaded = files.upload()
```

Loading our housing dataset

We will load our data on house sales in King County to predict house prices using simple (one input) linear regression

```
[ ]
dataset = pd.read_csv('kc_house_data.csv')
```

We want to be able to predict `Y` which is our price variable.

```
[ ]
Y = dataset[['price']]
```

```
[ ]
X = dataset.drop(['price', 'id', 'date'],  axis=1)
```

```python
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       21613 non-null  int64  
 1   bathrooms      21613 non-null  float64
 2   sqft_living    21613 non-null  int64  
 3   sqft_lot       21613 non-null  int64  
 4   floors         21613 non-null  float64
 5   waterfront     21613 non-null  int64  
 6   view           21613 non-null  int64  
 7   condition      21613 non-null  int64  
 8   grade          21613 non-null  int64  
 9   sqft_above     21613 non-null  int64  
 10  sqft_basement  21613 non-null  int64  
 11  yr_built       21613 non-null  int64  
```

```
[ ] 1 #list our columns
    2 columns = X.columns
    3 columns
```

```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

```
[ ] 1 #show first 5 records
    2 X.head()
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built
0	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	1955
1	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	1951
2	2	1.00	770	10000	1.0	0	0	3	6	770	0	1933

```python
1 X.describe()
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
count	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000
mean	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303
std	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318
min	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000
25%	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000
50%	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000
75%	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000
max	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000

```
1 dataset = dataset.drop(['id', 'date'], axis=1)
```

```
1 dataset.corr(method='pearson')
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
price	1.000000	0.308350	0.525138	0.702035	0.089661	0.256794	0.266369	0.397293
bedrooms	0.308350	1.000000	0.515884	0.576671	0.031703	0.175429	-0.006582	0.079532
bathrooms	0.525138	0.515884	1.000000	0.754665	0.087740	0.500653	0.063744	0.187737
sqft_living	0.702035	0.576671	0.754665	1.000000	0.172826	0.353949	0.103818	0.284611
sqft_lot	0.089661	0.031703	0.087740	0.172826	1.000000	-0.005201	0.021604	0.074710
floors	0.256794	0.175429	0.500653	0.353949	-0.005201	1.000000	0.023698	0.029444
waterfront	0.266369	-0.006582	0.063744	0.103818	0.021604	0.023698	1.000000	0.401857
view	0.397293	0.079532	0.187737	0.284611	0.074710	0.029444	0.401857	1.000000
condition	0.036362	0.028472	-0.124982	-0.058753	-0.008958	-0.263768	0.016653	0.045990

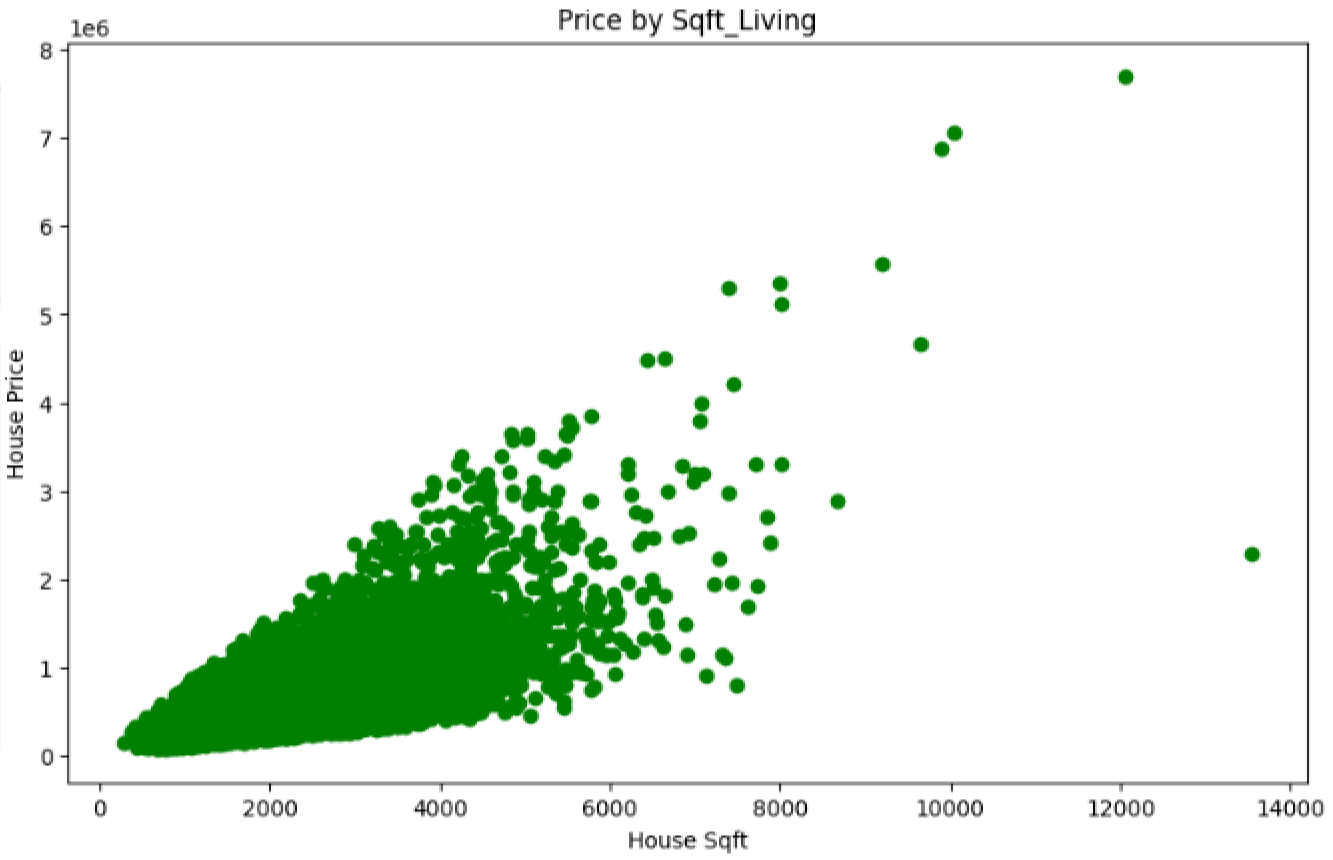
```
plt.subplots(figsize=(10,8))
sns.heatmap(dataset.corr())
```

It is **Simple Linear Regression** when we have one dependent variable (feature) and one independent variable. Here we will pick `sqft_living` as our independent variable `x`.

Our goal is to estimate $\hat{y} = x\theta_1 + \theta_0$, where θ_1 is our coefficient and θ_0 is our `Y` intercept. To estimate $\hat{y}$ we need to find a function such as $\hat{y} = h(x) = x\theta_1 + \theta_0$

```python
x = X[['sqft_living']]
y = Y
```

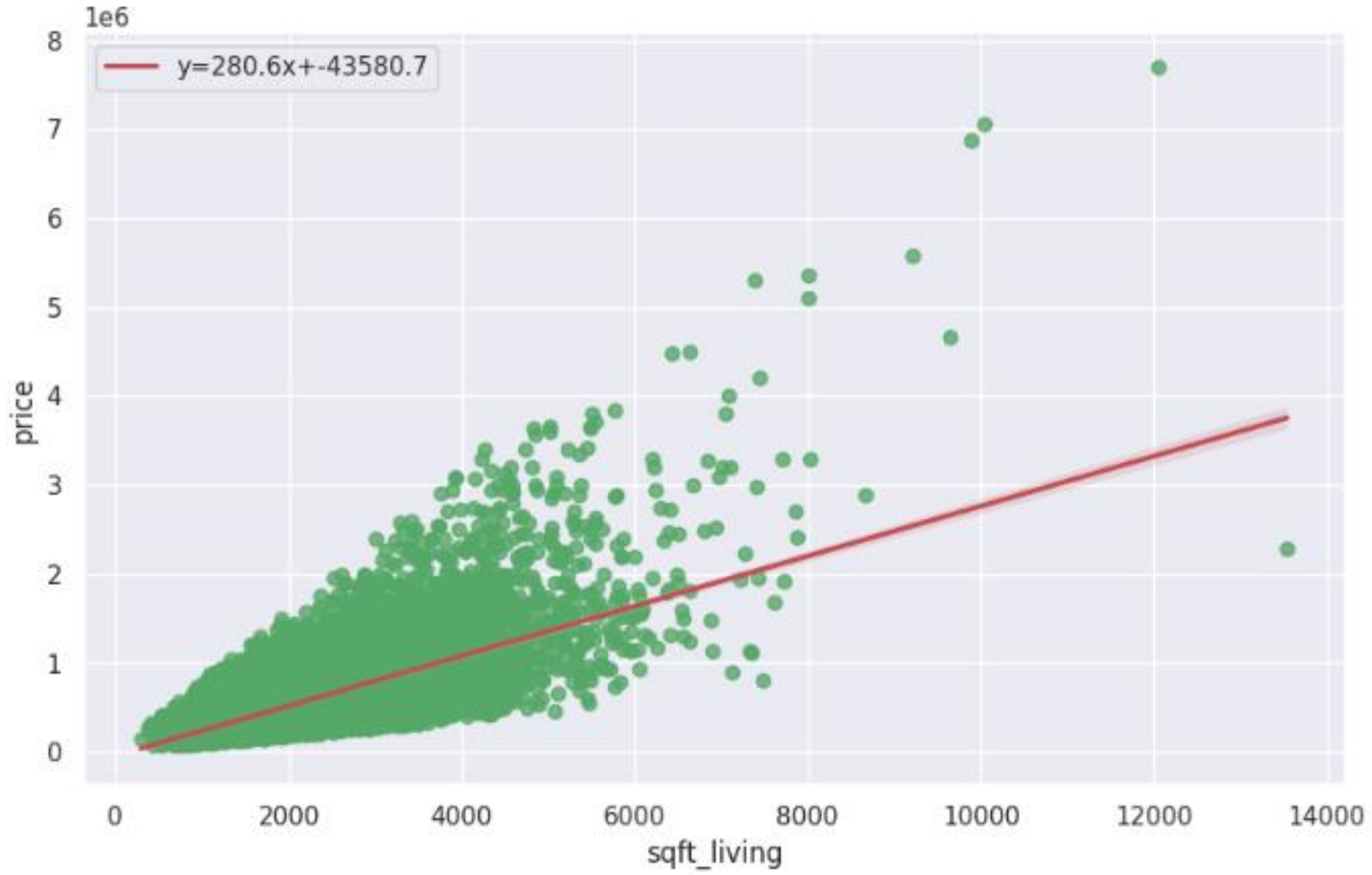
```python
plt.figure(figsize=(10,6))
plt.xlabel('House Sqft')
plt.ylabel('House Price')
plt.title('Price by Sqft_Living')
plt.scatter(x,y, marker='o', color='g')
```



Simple Linear Regression Implementations:

1. Using `seaborn.regplot()` and `scipy.stats`

```
[ ]
from scipy import stats
sns.set(color_codes=True)

slope, intercept, r_value, p_value, std_err = stats.linregress(dataset['sqft_living'],dataset['price'])

f = plt.figure(figsize=(10,6))
data = dataset[['price','sqft_living']]
ax = sns.regplot(x='sqft_living', y='price', data=data,
                  scatter_kws={"color": "g"},
                 line_kws={'color': 'r', 'label':"y={0:.1f}x+{1:.1f}".format(slope,intercept)})
ax.legend()
```




```
1 print(slope, intercept)
```

```
280.6235678974483 -43580.74309447408
```

```
1 print(std_err)
```

```
1.9363985519989133
```

2. Manual Method : Gradient Descent Implementation

Top

Equations

Objective of Linear Regression is to minimize the cost function:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

where the hypothesis $h_\theta(x)$ is given by the lienar model:

$$h_\theta(x) = \theta^T X = \theta_1 X_1 + \theta_0$$

In batch gradient descent, each iteration performs the update:

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

```python
x = X[['sqft_living']]
y = Y
```

```python
xg = x.values.reshape(-1,1)
yg = y.values.reshape(-1,1)
xg = np.concatenate((np.ones(len(x)).reshape(-1,1), x), axis=1)
```

Implementing the Cost Function $J(\theta)$ in python

```python
def computeCost(x, y, theta):
    m = len(y)
    h_x = x.dot(theta)
    j = np.sum(np.square(h_x - y))*(1/(2*m))
    return j
```

```python
def gradientDescent(x, y, theta, alpha, iteration):
    print('Running Gradient Descent...')
    j_hist = []
    m = len(y)
    for i in range(iteration):
        j_hist.append(computeCost(x, y, theta))
        h_x = x.dot(theta)
        theta = theta - ((alpha/m) *((np.dot(x.T, (h_x-y) ))))
        #theta[0] = theta[0] - ((alpha/m) *(np.sum((h_x-y))))
    return theta, j_hist
```

```python
[ ]
theta = np.zeros((2,1))
iteration = 2000
alpha = 0.001

theta, cost = gradientDescent(xg, yg, theta, alpha, iteration)
print('Theta found by Gradient Descent: slope = {} and intercept {}'.format(theta[1], theta[0]))
```

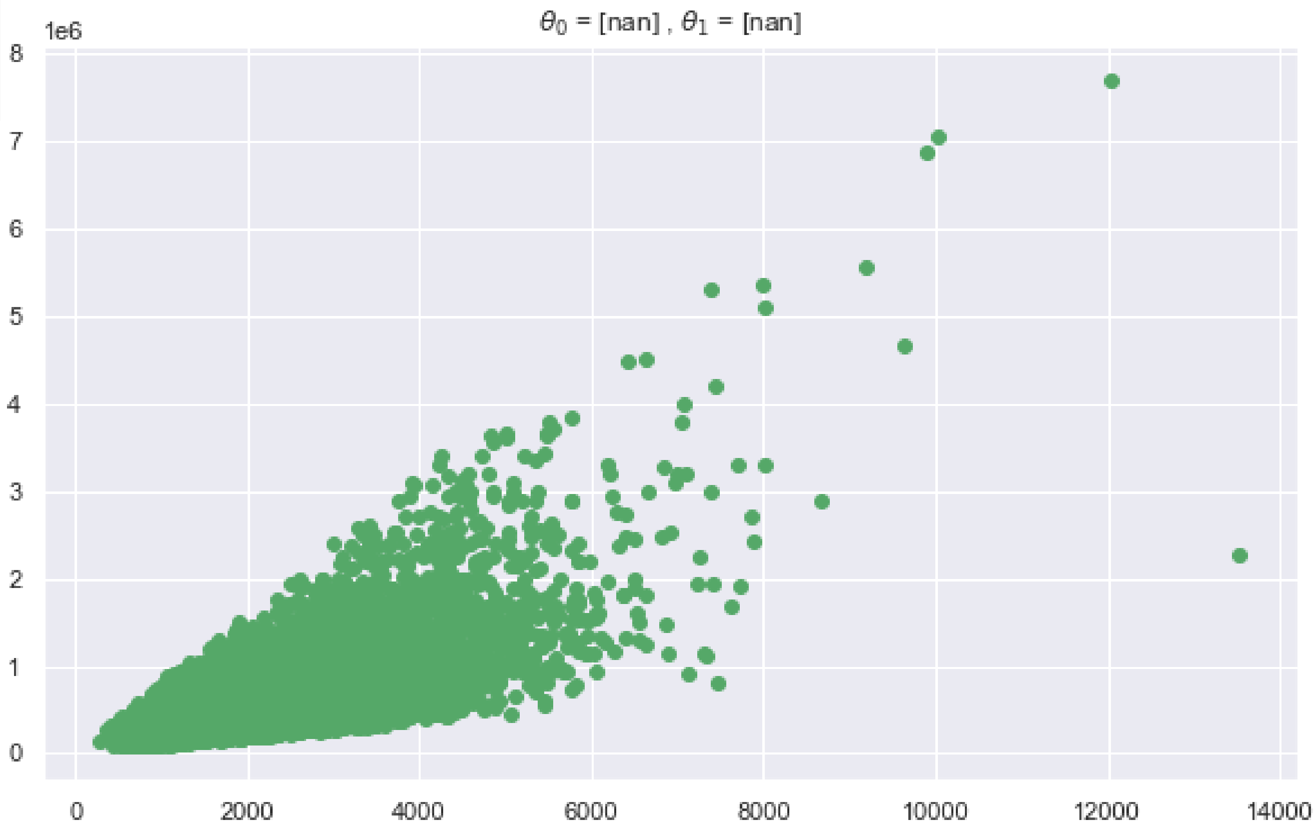
```
Running Gradient Descent...
<ipython-input-170-4c6aec17be90>:4: RuntimeWarning: overflow encountered in square
  j = np.sum(np.square(h_x - y))*(1/(2*m))
<ipython-input-171-e81683f4f12d>:8: RuntimeWarning: invalid value encountered in subtract
  theta = theta - ((alpha/m) *((np.dot(x.T, (h_x-y) ))))
Theta found by Gradient Descent: slope = [nan] and intercept [nan]
```

Plotting the linear fit

```
[ ]  theta.shape
```

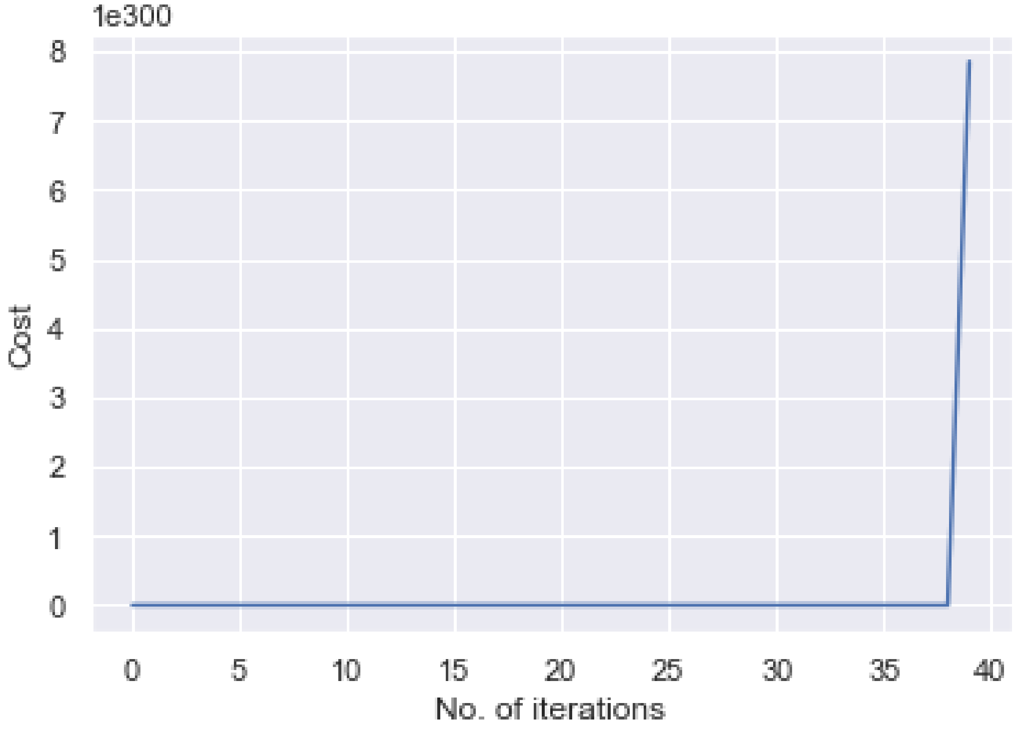
```
(2, 1)
```

```
[ ]  plt.figure(figsize=(10,6))
     plt.title('$\\theta_0$ = {} , $\\theta_1$ = {}'.format(theta[0], theta[1]))
     plt.scatter(x,y, marker='o', color='g')
     plt.plot(x,np.dot(x.values, theta.T))
     plt.show()
```



```
[ ]
plt.plot(cost)
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
```

```
Text(0, 0.5, 'Cost')
```



4. Implement with using Scipy

```
[ ]
from scipy import stats

xs = x.iloc[:,0]
ys = y.iloc[:,0]
#xs = np.concatenate((np.ones(len(x)).reshape(-1,1), x), axis=1)

slope, intercept, r_value, p_value, std_err = stats.linregress(xs, ys)
```

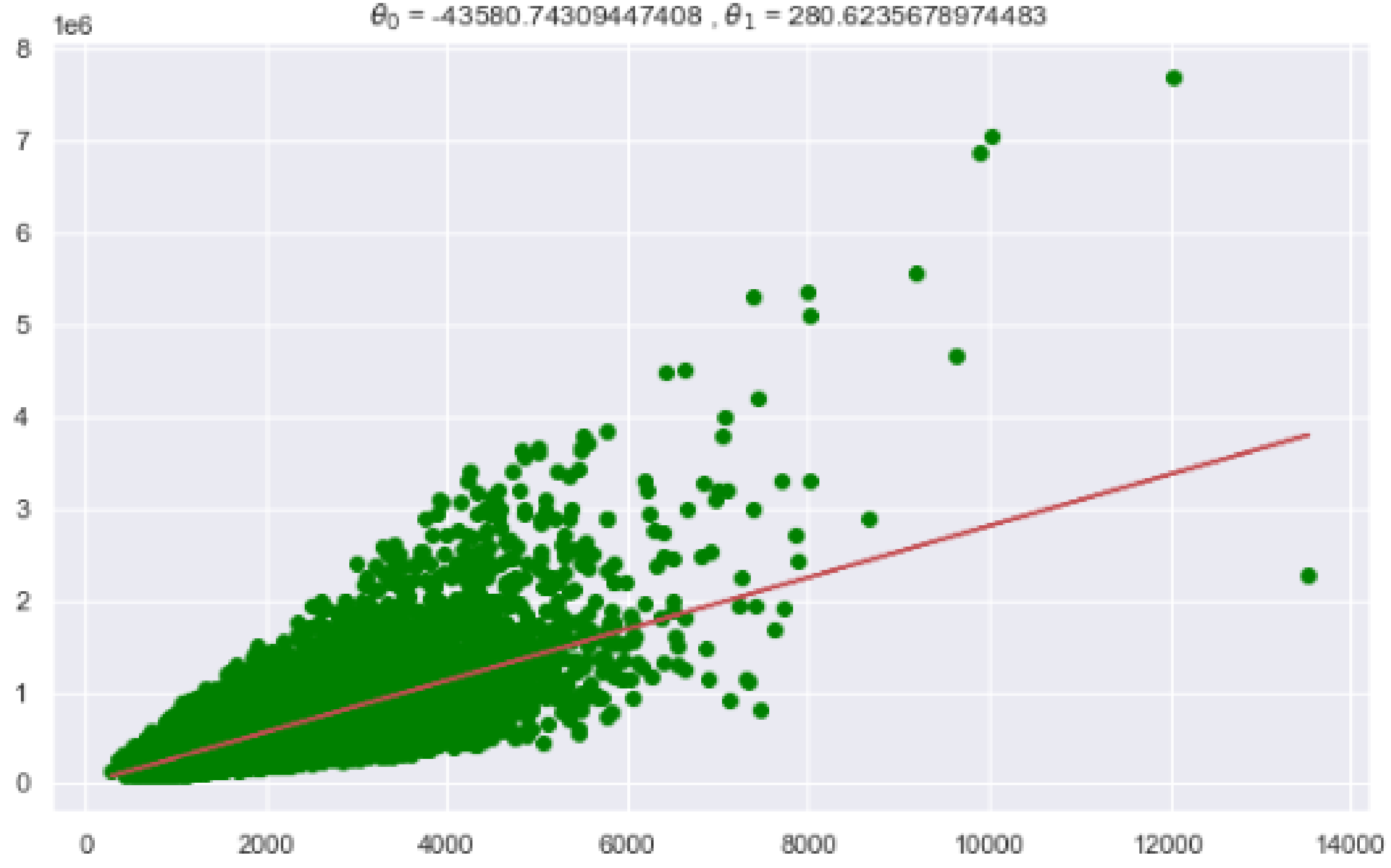
```
[ ]
print('Slope = {} and Intercept = {}'.format(slope, intercept))
print('y = x({}) + {}'.format(slope, intercept))
```

```
Slope = 280.6235678974483 and Intercept = -43580.74309447408
y = x(280.6235678974483) + -43580.74309447408
```

Plot the linear fit using the slop and intercept values from scipy

```
[ ]  plt.figure(figsize=(10,6))
     plt.title('$\\theta_0$ = {} , $\\theta_1$ = {}'.format(intercept, slope))
     plt.scatter(xs,y, marker='o', color='green')
     plt.plot(xs, np.dot(x, slope), 'r')
```

```
[<matplotlib.lines.Line2D at 0x7f9eb6b8af10>]
```

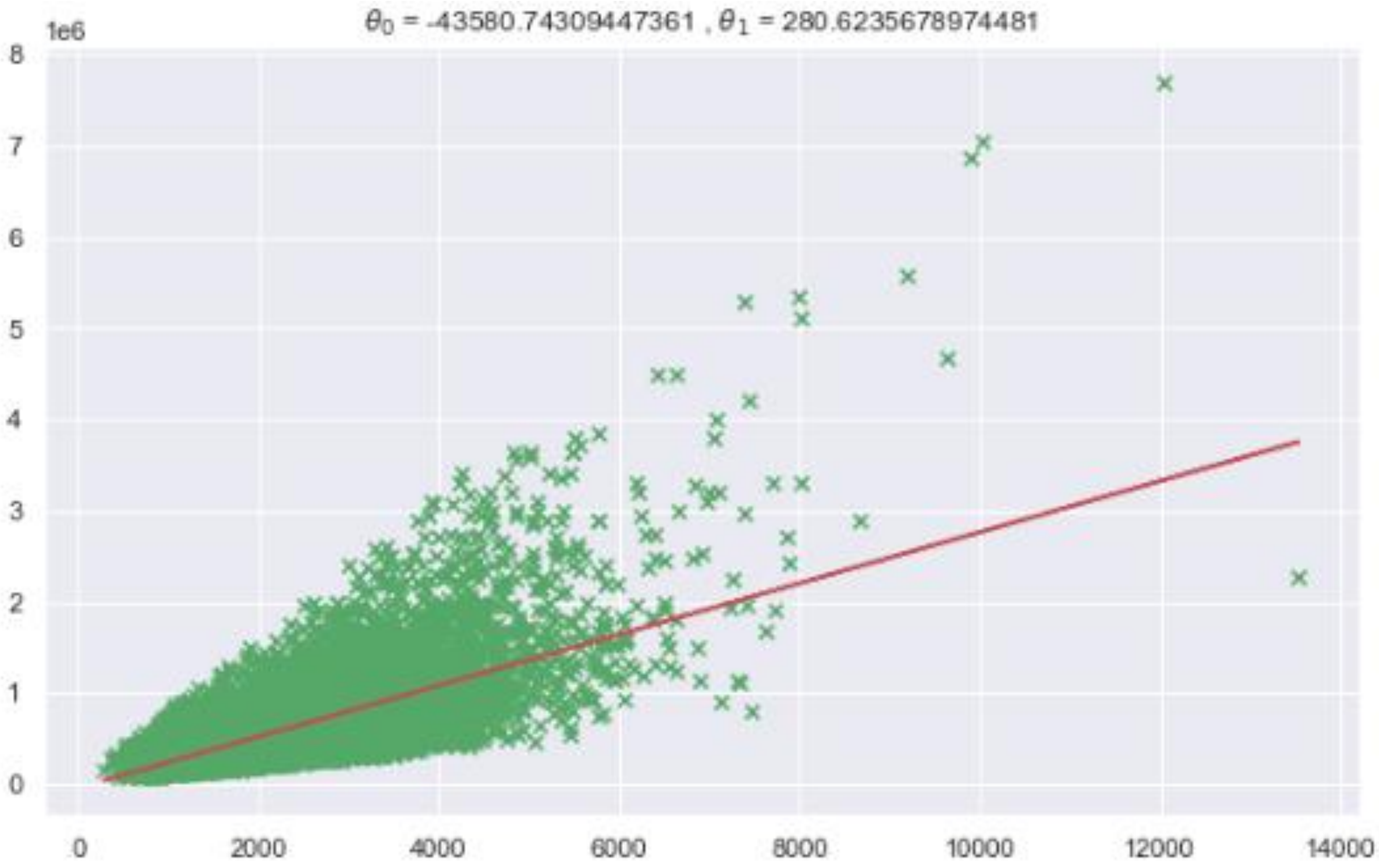


5. Implement using Scikit-Learn

```
[ ]
xsl = x.values.reshape(-1,1)
ysl = y.values.reshape(-1,1)
xsl = np.concatenate((np.ones(len(xsl)).reshape(-1,1), xsl), axis=1)

from sklearn.linear_model import LinearRegression

slr = LinearRegression()
slr.fit(xsl[:,1].reshape(-1,1), ysl.reshape(-1,1))
y_hat = slr.predict(xsl[:,1].reshape(-1,1))

print('theta[0] = ', slr.intercept_)
print('theta[1] = ', slr.coef_)

thetas = np.array((slr.intercept_, slr.coef_)).squeeze()
```

```
theta[0] =  [-43580.74309447]
theta[1] =  [[280.6235679]]
```

```
[ ]
plt.figure(figsize=(10,6))
plt.title('$\\theta_0$ = {} , $\\theta_1$ = {}'.format(thetas[0], thetas[1]))
plt.scatter(xsl[:,1],y, marker='x', color='g')
plt.plot(xsl[:,1], np.dot(xsl, thetas), 'r')
```



θ_0 = -43580.74309447361 , θ_1 = 280.6235678974481
1e6
8
7
6
5
4
3
2
1
0
0
2000
4000
6000
8000
10000
12000
14000