Author - AYUSH CHHOKER

Linear Regression

1. DATA ANALYSIS

2. SIMPLE LINEAR REGRESSION

3. MULTI LINEAR REGRESSION

Importing needed libraries

In [2]:

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

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

In [2]:

```
dataset = pd.read_csv('datasets/kc_house_data.csv')
```

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

In [3]:

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

In [4]:

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

1. Data Analysis

Basic data discovery and analysis with `info`, `describe` and `head`

using pandas `.info()` we see we have 18 columns and 21613 records. Pretty much all the features given are already in numeric format.

In [5]:

```
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  
 12  yr_renovated   21613 non-null  int64  
 13  zipcode        21613 non-null  int64  
 14  lat            21613 non-null  float64
 15  long           21613 non-null  float64
 16  sqft_living15  21613 non-null  int64  
 17  sqft_lot15     21613 non-null  int64  
dtypes: float64(4), int64(14)
memory usage: 3.0 MB
```

In [6]:

```
#list our columns
columns = X.columns
columns
```

Out[6]:

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

In [7]:

```
#show first 5 records
X.head()
```

Out[7]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_
0	3	1.00	1180	5650	1.0	0	0	3	7	
1	3	2.25	2570	7242	2.0	0	0	3	7	
2	2	1.00	770	10000	1.0	0	0	3	6	
3	4	3.00	1960	5000	1.0	0	0	5	7	
4	3	2.00	1680	8080	1.0	0	0	3	8	

Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding `NaN` values

In [8]:

```
X.describe()
```

Out[8]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	
count	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21
mean	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	
std	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	
min	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	
25%	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	
50%	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	
75%	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	
max	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	

Compute correlation between variables and our predictor variable

In [9]:

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

In [10]:

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

Out[10]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
price	1.000000	0.308350	0.525138	0.702035	0.089661	0.256794	0.266369
bedrooms	0.308350	1.000000	0.515884	0.576671	0.031703	0.175429	-0.006582
bathrooms	0.525138	0.515884	1.000000	0.754665	0.087740	0.500653	0.063744
sqft_living	0.702035	0.576671	0.754665	1.000000	0.172826	0.353949	0.103818
sqft_lot	0.089661	0.031703	0.087740	0.172826	1.000000	-0.005201	0.021604
floors	0.256794	0.175429	0.500653	0.353949	-0.005201	1.000000	0.023698
waterfront	0.266369	-0.006582	0.063744	0.103818	0.021604	0.023698	1.000000
view	0.397293	0.079532	0.187737	0.284611	0.074710	0.029444	0.401857
condition	0.036362	0.028472	-0.124982	-0.058753	-0.008958	-0.263768	0.016653
grade	0.667434	0.356967	0.664983	0.762704	0.113621	0.458183	0.082775
sqft_above	0.605567	0.477600	0.685342	0.876597	0.183512	0.523885	0.072075
sqft_basement	0.323816	0.303093	0.283770	0.435043	0.015286	-0.245705	0.080588
yr_built	0.054012	0.154178	0.506019	0.318049	0.053080	0.489319	-0.026161
yr_renovated	0.126434	0.018841	0.050739	0.055363	0.007644	0.006338	0.092885
zipcode	-0.053203	-0.152668	-0.203866	-0.199430	-0.129574	-0.059121	0.030285
lat	0.307003	-0.008931	0.024573	0.052529	-0.085683	0.049614	-0.014274
long	0.021626	0.129473	0.223042	0.240223	0.229521	0.125419	-0.041910
sqft_living15	0.585379	0.391638	0.568634	0.756420	0.144608	0.279885	0.086463
sqft_lot15	0.082447	0.029244	0.087175	0.183286	0.718557	-0.011269	0.030703

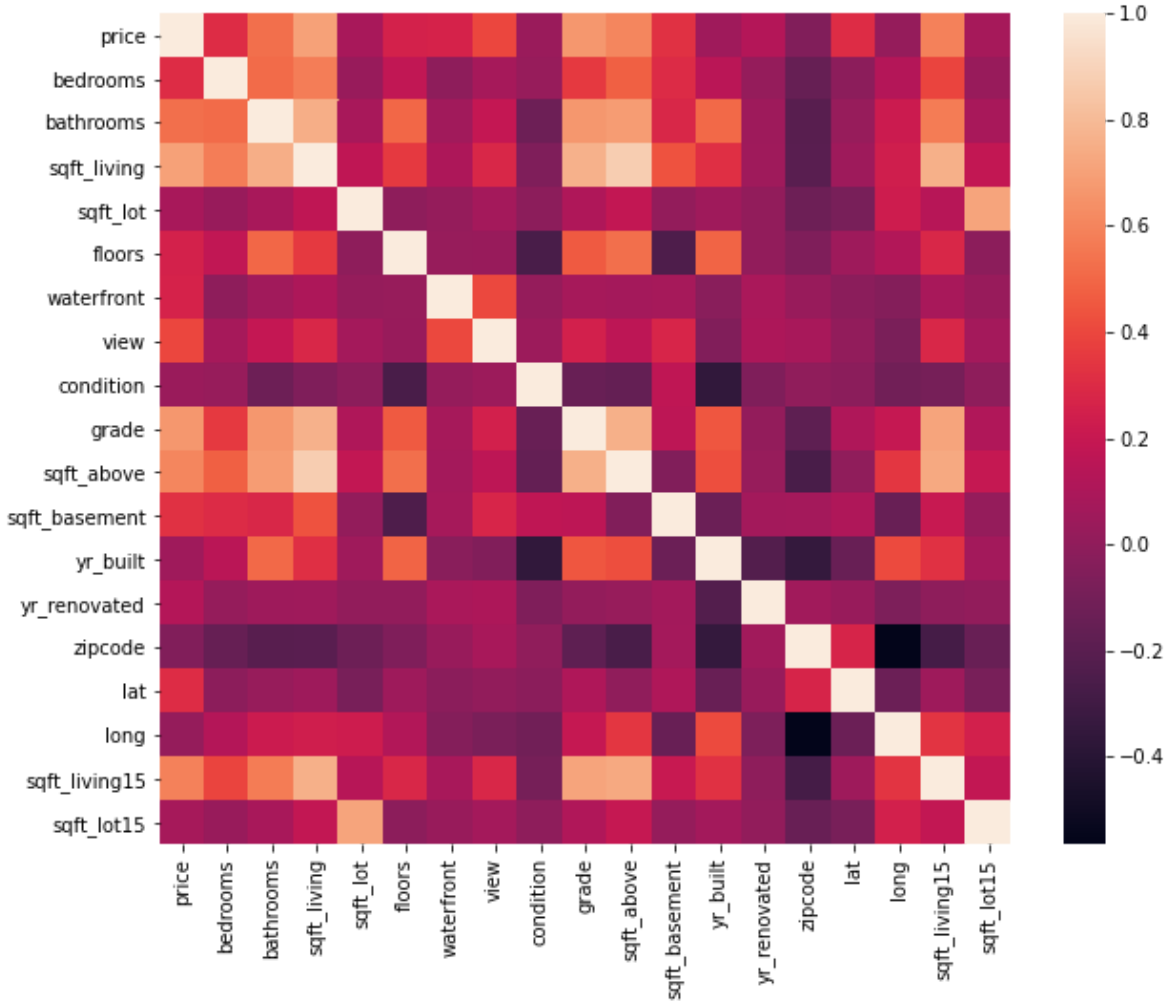
We can even visualize the table above

In [11]:

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

Out[11]:

<AxesSubplot:>



`statsmodel` package can also give us some great insight and summary statistics including `p-value`

The statsmodel can actually perform the regression modeling for us , but here I am mainly using it to help determine which variable I should focus on for my `Simple Linear Regression` (one independent variable) and get a feel of which values are statistically significant. There are techniques when dealing with `Multiple Linear Regression` (many variable) to narrow down to the most significant features/variables usiung **Step Wise Regression** which include techniques such as **Forward Selection** and **Backward Elimination**.

In [15]:

```
X
```

Out[15]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	s
0	3	1.00	1180	5650	1.0	0	0	3	7	
1	3	2.25	2570	7242	2.0	0	0	3	7	
2	2	1.00	770	10000	1.0	0	0	3	6	
3	4	3.00	1960	5000	1.0	0	0	5	7	
4	3	2.00	1680	8080	1.0	0	0	3	8	
...	...	...	...	...	...	...	...	...	...	
21608	3	2.50	1530	1131	3.0	0	0	3	8	
21609	4	2.50	2310	5813	2.0	0	0	3	8	
21610	2	0.75	1020	1350	2.0	0	0	3	7	
21611	3	2.50	1600	2388	2.0	0	0	3	8	
21612	2	0.75	1020	1076	2.0	0	0	3	7	

21613 rows × 18 columns

In [16]:

```
import statsmodels.api as sml
from statsmodels import tools

X_new = tools.add_constant(X)

regressor_OLS = sml.OLS(endog = Y,exog =  X_new).fit()

regressor_OLS.summary()
```

Out[16]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.700
Model:	OLS	**Adj. R-squared:**	0.700
Method:	Least Squares	**F-statistic:**	2960.
Date:	Sun, 10 Jan 2021	**Prob (F-statistic):**	0.00
Time:	11:51:59	**Log-Likelihood:**	-2.9460e+05
No. Observations:	21613	**AIC:**	5.892e+05
Df Residuals:	21595	**BIC:**	5.894e+05
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.69e+06	2.93e+06	2.282	0.022	9.44e+05	1.24e+07
bedrooms	-3.577e+04	1891.843	-18.906	0.000	-3.95e+04	-3.21e+04
bathrooms	4.114e+04	3253.678	12.645	0.000	3.48e+04	4.75e+04
sqft_living	110.4422	2.270	48.661	0.000	105.994	114.891
sqft_lot	0.1286	0.048	2.683	0.007	0.035	0.223
floors	6689.5501	3595.859	1.860	0.063	-358.599	1.37e+04
waterfront	5.83e+05	1.74e+04	33.580	0.000	5.49e+05	6.17e+05
view	5.287e+04	2140.055	24.705	0.000	4.87e+04	5.71e+04
condition	2.639e+04	2351.461	11.221	0.000	2.18e+04	3.1e+04
grade	9.589e+04	2152.789	44.542	0.000	9.17e+04	1e+05
sqft_above	70.7859	2.253	31.413	0.000	66.369	75.203
sqft_basement	39.6583	2.646	14.985	0.000	34.471	44.846
yr_built	-2620.2232	72.659	-36.062	0.000	-2762.640	-2477.806
yr_renovated	19.8126	3.656	5.420	0.000	12.647	26.978
zipcode	-582.4199	32.986	-17.657	0.000	-647.074	-517.765
lat	6.027e+05	1.07e+04	56.149	0.000	5.82e+05	6.24e+05
long	-2.147e+05	1.31e+04	-16.349	0.000	-2.4e+05	-1.89e+05
sqft_living15	21.6814	3.448	6.289	0.000	14.924	28.439
sqft_lot15	-0.3826	0.073	-5.222	0.000	-0.526	-0.239

Omnibus:	18384.201	**Durbin-Watson:**	1.990
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1868224.491
Skew:	3.566	**Prob(JB):**	0.00
Kurtosis:	47.985	**Cond. No.**	4.77e+17

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 9.63e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

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 $\hat{y} = h(x) = x\theta_1 + \theta_0$

We first start by creating our `x` and `y` variables. Then plotting to gain an intuition on how the data looks like.

In [17]:

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

In [18]:

```
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')
```

Out[18]:

```
<matplotlib.collections.PathCollection at 0x7f9ecab3d430>
```

Simple Linear Regression Implementations:

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

In [20]:

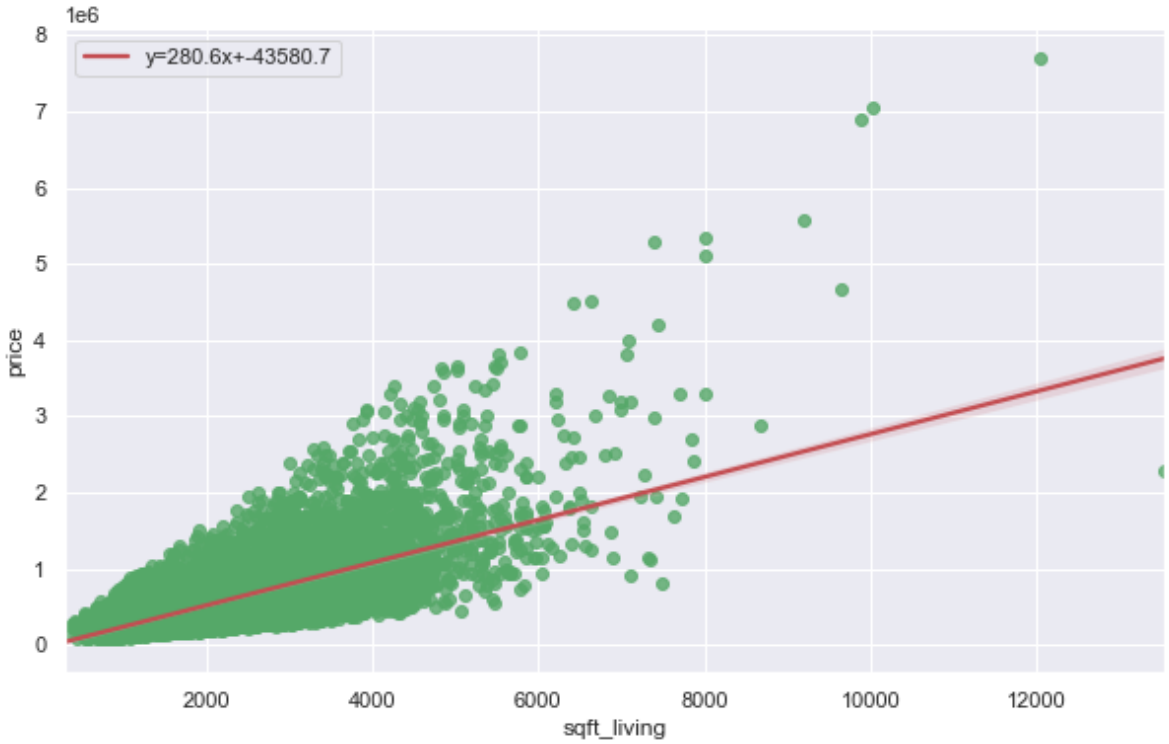
```
from scipy import stats
sns.set(color_codes=True)

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

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()
```

Out[20]:

```
<matplotlib.legend.Legend at 0x7f9ecdfc81c0>
```



In [24]:

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

```
280.6235678974483 -43580.74309447408
```

In [21]:

```
print(std_err)
```

```
1.9363985519989133
```

2. Manual Method : Gradient Descent Implementation

Top

Equations

Objective of Linear Regression is to minimize the cost function:

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

$\Large J(\theta) we called cost function or lost function here.

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

In batch gradient descent, each iteration performs the update:

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

In [168]:

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

In [169]:

```
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

In [170]:

```
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
```

In [171]:

```
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
```

In [172]:

```
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
```

```
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 encountere
d 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

In [173]:

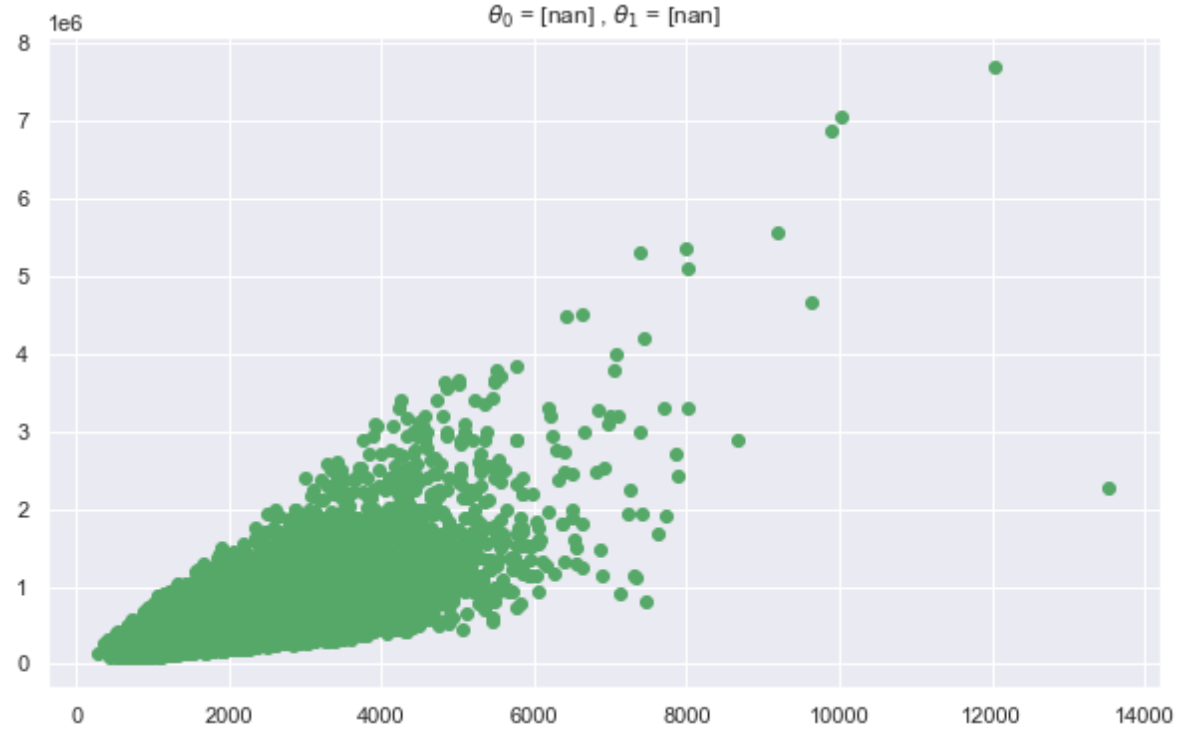
```
theta.shape
```

Out[173]:

```
(2, 1)
```

In [174]:

```
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()
```

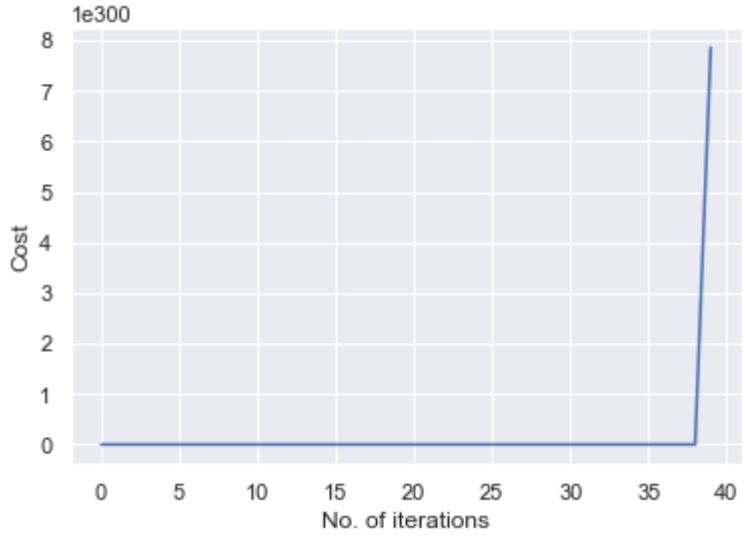


In [175]:

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

Out[175]:

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



3. Manual Method: Compute Slope and Intercept using a Formula (Gradient = 0)

Slope $a = \frac{\sum(x_i - \bar{x})(y_i - \bar{Y})}{\sum(x_i - \bar{x})^2}$

Intercept $b = \bar{y} - a\bar{x}$

In [176]:

```
def slr(X, Y):
    mean_x = X.mean()
    mean_y = Y.mean()

    delta_x = X - mean_x
    delta_y = Y - mean_y

    slope = (delta_x * delta_y).sum()/(delta_x**2).sum()
    intercept = mean_y - slope*mean_x

    return (slope, intercept)
```

In [177]:

```
xf = x.values.reshape(-1,1)
yf = y.values.reshape(-1,1)

slope, intercept = slr(xf, yf)
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
```

4. Implement with using Scipy

In [178]:

```
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)
```

In [179]:

```
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
```

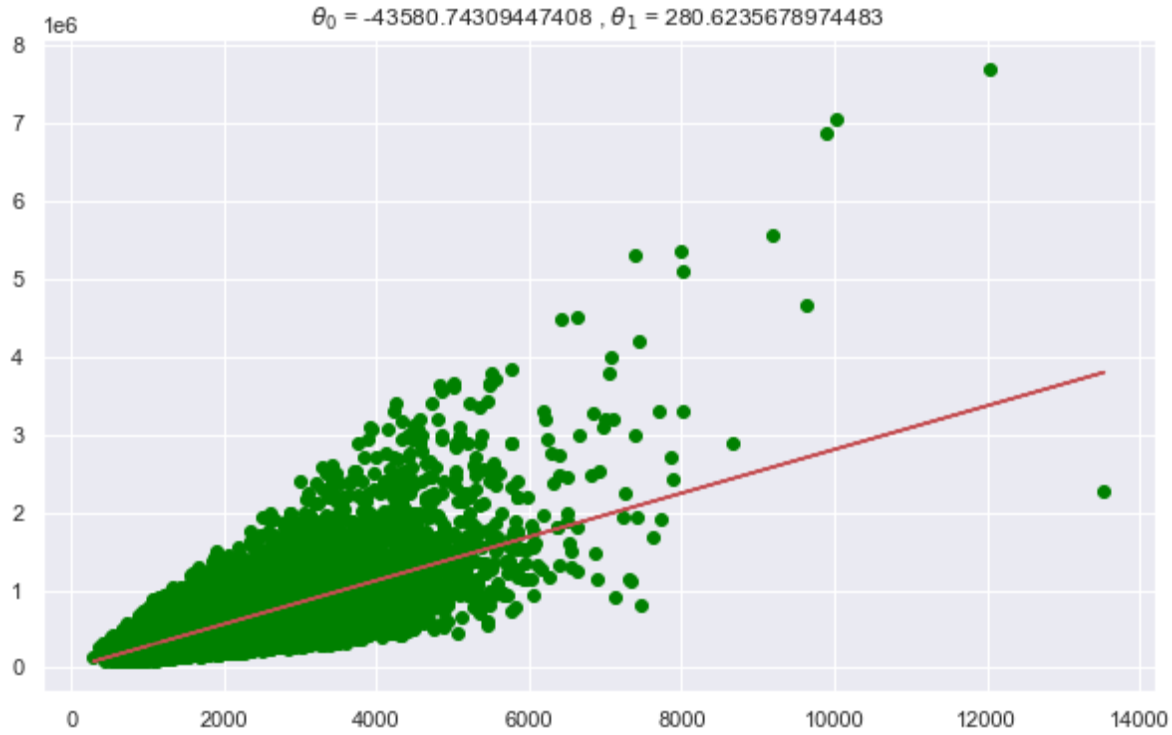
Plot the linear fit using the slop and intercept values from scipy

In [180]:

```
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')
```

Out[180]:

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



5. Implement using Scikit-Learn

In [181]:

```
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()
```

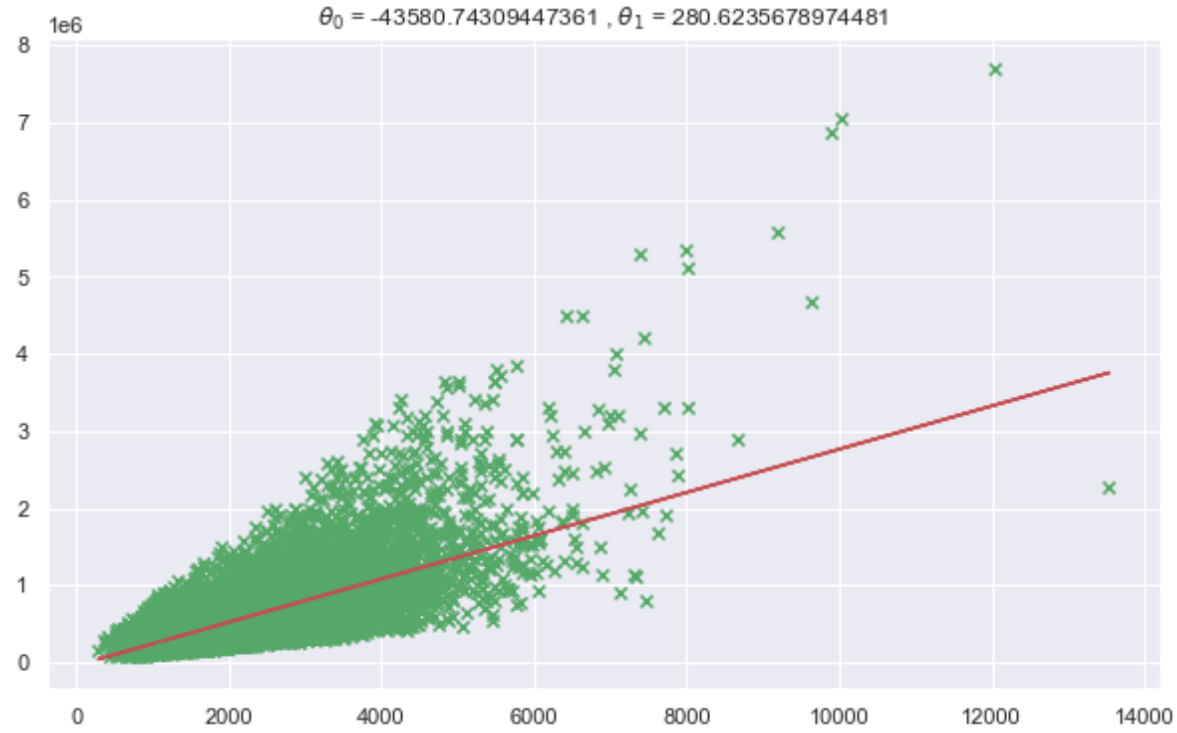
```
theta[0] =  [-43580.74309447]
theta[1] =  [[280.6235679]]
```

In [182]:

```
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')
```

Out[182]:

```
[<matplotlib.lines.Line2D at 0x7f9eb7479340>]
```



Implement using Statsmodel

In [183]:

```
xsm = x.values.reshape(-1,1)
ysm = y.values.reshape(-1,1)
xsm = np.concatenate((np.ones(len(x)).reshape(-1,1), xsm), axis=1)

import statsmodels.api as sm

results = sm.OLS(ysm, xsm).fit()
results.summary()
```

Out[183]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.493
Model:	OLS	**Adj. R-squared:**	0.493
Method:	Least Squares	**F-statistic:**	2.100e+04
Date:	Sun, 10 Jan 2021	**Prob (F-statistic):**	0.00
Time:	12:36:24	**Log-Likelihood:**	-3.0027e+05
No. Observations:	21613	**AIC:**	6.005e+05
Df Residuals:	21611	**BIC:**	6.006e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-4.358e+04	4402.690	-9.899	0.000	-5.22e+04	-3.5e+04
x1	280.6236	1.936	144.920	0.000	276.828	284.419

Omnibus:	14832.490	**Durbin-Watson:**	1.983
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	546444.713
Skew:	2.824	**Prob(JB):**	0.00
Kurtosis:	26.977	**Cond. No.**	5.63e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

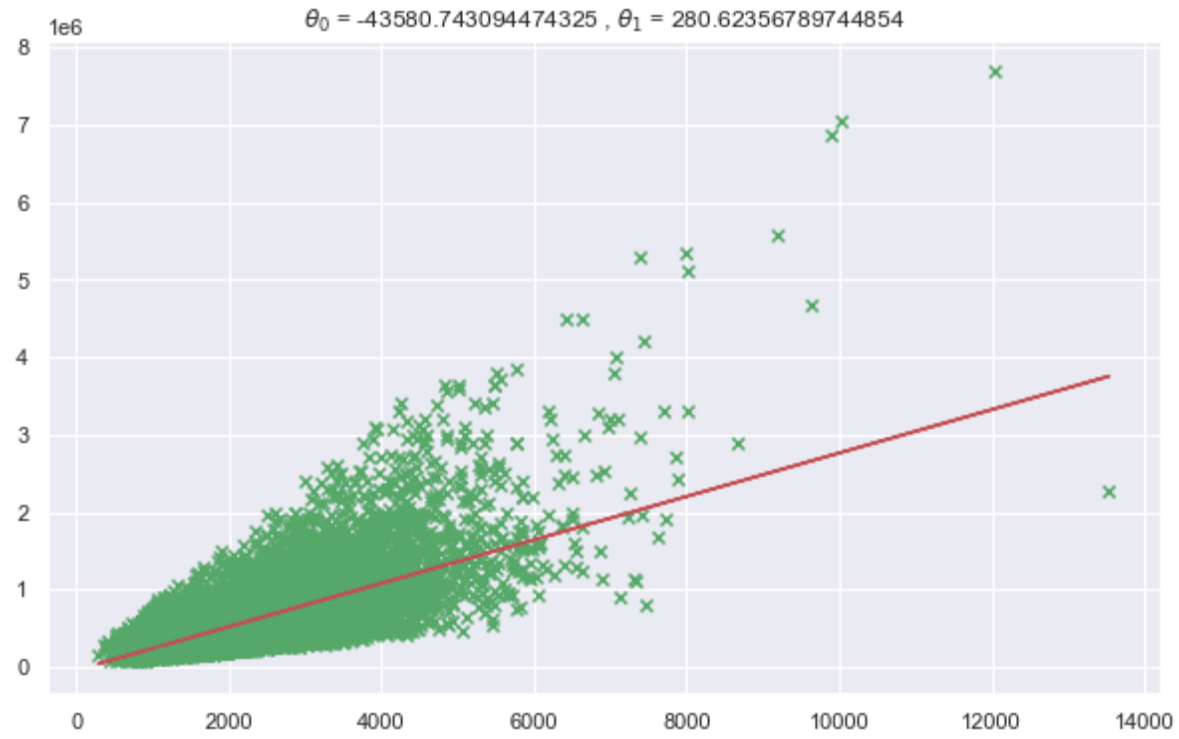
In [184]:

```
thetas = results.params

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

Out[184]:

```
[<matplotlib.lines.Line2D at 0x7f9eb1214790>]
```



Multiple Linear Regression

Top

In [185]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

dataset = pd.read_csv('datasets/kc_house_data.csv')
Y = dataset[['price']]
X = dataset.drop(['price', 'id', 'date'],  axis=1)
```

In [186]:

```
dataset.head()
```

Out[186]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0

5 rows × 21 columns

Feature Normalization

In [187]:

```
x = X.values
y = Y.values
```

In [188]:

```
def featureNormalize(x_m):
    mu = np.zeros((1,x_m.shape[1]))
    sigma = np.zeros((1,x_m.shape[1]))
    x_norm = x_m.astype(float)

    for i in range(0,len(mu)+1):
        mu[:,i] = x_m[:,i].mean()
        sigma[:,i] = x_m[:,i].std()
        x_norm[:,i] = (x_m[:,i] - mu[:,i])/sigma[:,i]
    return (x_norm, mu, sigma)
```

Normalized

In [189]:

```
x_norm, mu, sigma = featureNormalize(x)
x_norm = np.concatenate((np.ones(len(x_norm)).reshape(-1,1), x_norm), axis=1)
```

In [190]:

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

In [193]:

```
theta_init = np.zeros((19, 1))
computeCost_m(x_norm, Y, theta_init)
```

Out[193]:

```
price    2.132357e+11
dtype: float64
```

Manual - Gradient Descent

In [197]:

```
def gradientDescentMulti(X, Y, theta, alpha, num_iters):
    m = len(Y)
    p = np.copy(X)
    t = np.copy(theta)
    j = []
    print('Running Gradient Descent')
    for i in range(0,num_iters+1):
        cost = computeCost_m(p, Y, t)
        j.append(cost)
        h_x = np.dot(p, t)
        err = h_x - Y
        for f in range(theta.size):
            t[f] = t[f] - alpha/m *(np.sum((np.dot(p[:,f].T, err))))
    return j, t
```

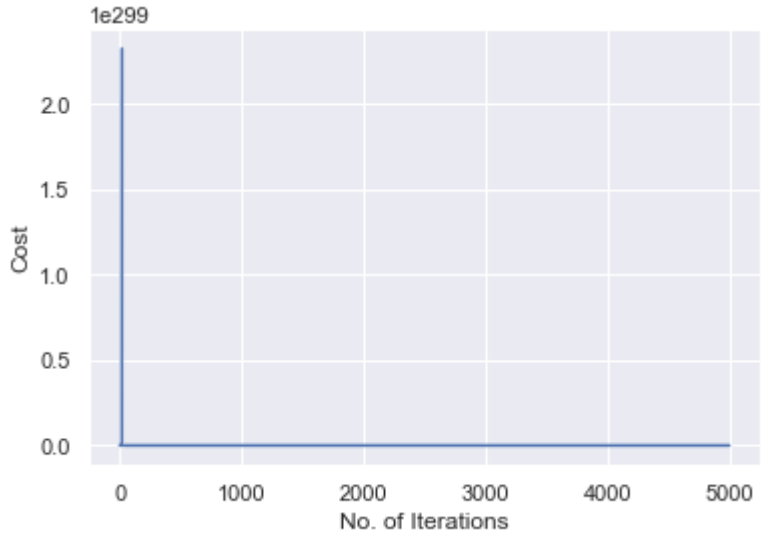
In [199]:

```
# theta_init = np.zeros((3, 1))
alpha = 0.01
num_iters = 5000
theta_init = np.zeros((19, 1))
cost, theta_final = gradientDescentMulti(x_norm, Y, theta_init, alpha, num_iters)

plt.figure()
plt.plot(cost)
plt.xlabel('No. of Iterations')
plt.ylabel('Cost')
plt.show()
```

```
Running Gradient Descent

<ipython-input-190-a9b84bc00e96>:4: RuntimeWarning: overflow encountered in
square
  j = np.sum(np.square(h_x - y))/(2*m)
```



In [200]:

```
theta_final
```

Out[200]:

```
array([[nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan],
       [nan]])
```

In [201]:

```
theta_init
```

Out[201]:

```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

Multi-variable Regression with Scikit-Learn

In [202]:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

mlr = LinearRegression()
sc_x = StandardScaler()
X_new = sc_x.fit_transform(x)
```

In [203]:

```
mlr.fit(X_new, Y)
y_hat = mlr.predict(X_new)
```

In [204]:

```
mlr.score(X_new, Y)
```

Out[204]:

```
0.6997471890109157
```

In [205]:

```
mlr.intercept_
```

Out[205]:

```
array([540088.14176653])
```

In [206]:

```
mlr.coef_
```

Out[206]:

```
array([[-33264.32545782,  31687.07437464,  81812.50710964,
          5326.46629074,   3612.19920732,  50434.93843527,
         40514.99473763,  17169.88047514, 112712.65585307,
         76305.51997634,  27005.67955588, -76963.1122356 ,
          7958.11945582, -31161.66927787,  83517.09842808,
        -30239.34618464,  14859.89955603, -10447.48009171]])
```

Solving with Normal Equation

In [207]:

```
x_neq = np.concatenate((np.ones(len(x)).reshape(-1,1), x), axis=1)
a = np.linalg.inv(np.dot(x_neq.T, x_neq))
b = np.dot(x_neq.T, y)
theta_neq = np.dot(a,b)
```

In [208]:

```
theta_neq
```

Out[208]:

```
array([[-8.87182378e+06],
       [ 5.04750191e+06],
       [-7.81287005e+05],
       [ 9.94709813e+03],
       [-1.80933539e-01],
       [ 4.03583721e+05],
       [-2.74465540e+06],
       [ 3.26413834e+05],
       [ 3.92312640e+04],
       [-3.64827988e+04],
       [-9.86376221e+03],
       [-9.66659353e+03],
       [-2.62022321e+03],
       [ 1.98125837e+01],
       [-5.82419866e+02],
       [ 6.02748227e+05],
       [-2.14729829e+05],
       [ 2.16814005e+01],
       [-3.82641849e-01]])
```

Scipy implementation

In [ ]:

```
from scipy import stats

slope, intercept, r_value, p_value, std_err = stats.linregress(X.values.reshape(-1,1),
                                                               Y.values)

print('theta[0] = ', intercept)
print('theta[1] = ', slope)
```

Statsmodel implementation

Statsmodel WITHOUT Feature Scaling/Normalization

In [558]:

```
import statsmodels.api as sm
#without feature scaling
results = sm.OLS(Y, x_neq).fit()
results.summary()
```

Out[558]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.700
Model:	OLS	**Adj. R-squared:**	0.700
Method:	Least Squares	**F-statistic:**	2960.
Date:	Wed, 22 Aug 2018	**Prob (F-statistic):**	0.00
Time:	23:31:12	**Log-Likelihood:**	-2.9460e+05
No. Observations:	21613	**AIC:**	5.892e+05
Df Residuals:	21595	**BIC:**	5.894e+05
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.69e+06	2.93e+06	2.282	0.022	9.44e+05	1.24e+07
x1	-3.577e+04	1891.843	-18.906	0.000	-3.95e+04	-3.21e+04
x2	4.114e+04	3253.678	12.645	0.000	3.48e+04	4.75e+04
x3	110.4380	2.270	48.661	0.000	105.990	114.886
x4	0.1286	0.048	2.683	0.007	0.035	0.223
x5	6689.5501	3595.859	1.860	0.063	-358.599	1.37e+04
x6	5.83e+05	1.74e+04	33.580	0.000	5.49e+05	6.17e+05
x7	5.287e+04	2140.055	24.705	0.000	4.87e+04	5.71e+04
x8	2.639e+04	2351.461	11.221	0.000	2.18e+04	3.1e+04
x9	9.589e+04	2152.789	44.542	0.000	9.17e+04	1e+05
x10	70.7901	2.253	31.418	0.000	66.374	75.207
x11	39.6625	2.647	14.985	0.000	34.475	44.850
x12	-2620.2232	72.659	-36.062	0.000	-2762.640	-2477.806
x13	19.8126	3.656	5.420	0.000	12.647	26.978
x14	-582.4199	32.986	-17.657	0.000	-647.074	-517.765
x15	6.027e+05	1.07e+04	56.149	0.000	5.82e+05	6.24e+05
x16	-2.147e+05	1.31e+04	-16.349	0.000	-2.4e+05	-1.89e+05
x17	21.6814	3.448	6.289	0.000	14.924	28.439
x18	-0.3826	0.073	-5.222	0.000	-0.526	-0.239

Omnibus:	18384.201	**Durbin-Watson:**	1.990
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1868224.491
Skew:	3.566	**Prob(JB):**	0.00

Kurtosis:	47.985	**Cond. No.**	2.52e+17

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.45e-21. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [559]:

```
tt = results.params

#predict house for 1650 and 3 bedroom
tt[0] + (tt[1] * 1650) + (tt[2] * 3)
```

Out[559]:

```
-52201035.94329346
```

Statsmodel using Feature Scaling/Normalization

In [560]:

```
import statsmodels.api as sm
#with feature scaling
d = sm.add_constant(X_new)
results = sm.OLS(Y, d).fit()
results.summary()
```

Out[560]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.700
Model:	OLS	**Adj. R-squared:**	0.700
Method:	Least Squares	**F-statistic:**	2960.
Date:	Wed, 22 Aug 2018	**Prob (F-statistic):**	0.00
Time:	23:31:33	**Log-Likelihood:**	-2.9460e+05
No. Observations:	21613	**AIC:**	5.892e+05
Df Residuals:	21595	**BIC:**	5.894e+05
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.401e+05	1368.905	394.540	0.000	5.37e+05	5.43e+05
x1	-3.326e+04	1759.490	-18.906	0.000	-3.67e+04	-2.98e+04
x2	3.169e+04	2505.805	12.645	0.000	2.68e+04	3.66e+04
x3	8.181e+04	1588.340	51.508	0.000	7.87e+04	8.49e+04
x4	5326.4663	1984.923	2.683	0.007	1435.870	9217.062
x5	3612.1992	1941.679	1.860	0.063	-193.635	7418.034
x6	5.043e+04	1501.912	33.580	0.000	4.75e+04	5.34e+04
x7	4.051e+04	1639.923	24.705	0.000	3.73e+04	4.37e+04
x8	1.717e+04	1530.162	11.221	0.000	1.42e+04	2.02e+04
x9	1.127e+05	2530.456	44.542	0.000	1.08e+05	1.18e+05
x10	7.631e+04	1696.376	44.981	0.000	7.3e+04	7.96e+04
x11	2.701e+04	1559.700	17.315	0.000	2.39e+04	3.01e+04
x12	-7.696e+04	2134.197	-36.062	0.000	-8.11e+04	-7.28e+04
x13	7958.1195	1468.341	5.420	0.000	5080.062	1.08e+04
x14	-3.116e+04	1764.866	-17.657	0.000	-3.46e+04	-2.77e+04
x15	8.352e+04	1487.409	56.149	0.000	8.06e+04	8.64e+04
x16	-3.024e+04	1849.583	-16.349	0.000	-3.39e+04	-2.66e+04
x17	1.486e+04	2362.983	6.289	0.000	1.02e+04	1.95e+04
x18	-1.045e+04	2000.508	-5.222	0.000	-1.44e+04	-6526.336

Omnibus:	18384.201	**Durbin-Watson:**	1.990
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1868224.491

Skew:	3.566	**Prob(JB):**	0.00
Kurtosis:	47.985	**Cond. No.**	8.31e+15

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.63e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [ ]:

In [ ]: