In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
import time
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
```

Univariate Linear Regression

In [2]:
```
data=pd.read_csv("Linear Regression/ex1data1.txt", header=None)
data.head()
```

Out[2]:

	0	1
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233

In [3]:
```
data.describe()
```

Out[3]:

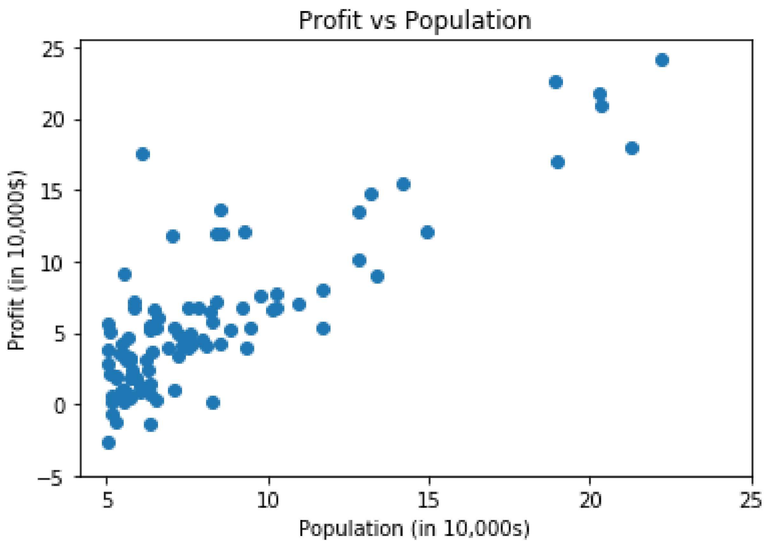
	0	1
count	97.000000	97.000000
mean	8.159800	5.839135
std	3.869884	5.510262
min	5.026900	-2.680700
25%	5.707700	1.986900
50%	6.589400	4.562300
75%	8.578100	7.046700
max	22.203000	24.147000

In [4]:
```
data.columns = ['Population', 'Profit'] #assigning column names
```

Profit Vs Population Graph

In [5]:
```
plt.scatter(data['Population'], data['Profit'])
plt.xticks(np.arange(5,30,step=5))
plt.yticks(np.arange(-5,30,step=5))
plt.xlabel('Population (in 10,000s)')
plt.ylabel('Profit (in 10,000$)')
plt.title('Profit vs Population')
```

Out[5]:
```
Text(0.5, 1.0, 'Profit vs Population')
```



Cost Function J(θ)

In [6]:
```
def computeCost(X,y,theta):
    """
    Take in a numpy arary X,y,theta and get cost function using theta as param
eter in a linear regression model
    """
    m = len(y)
    predictions = X.dot(theta)
    square_err = (predictions - y)**2

    return 1/(2*m)*np.sum(square_err)
```

Assigning X, y, θ and Reshaping the data

In [7]:
```
data['x0'] = 1
data_val = data.values
m =  len(data_val[:-1])
X = data[['x0','Population']].iloc[:-1].values
y = data['Profit'][:-1].values.reshape(m,1)
theta = np.zeros((2,1))

m, X.shape, y.shape, theta.shape
```

Out[7]:
```
(96, (96, 2), (96, 1), (2, 1))
```

$$h(\theta) = x_0\theta_0 + x_1\theta_1 \ldots (x_0 = 1)$$

In [8]:
```
computeCost(X,y,theta)
```

Out[8]:
```
32.40484177877031
```

Gradient Descent

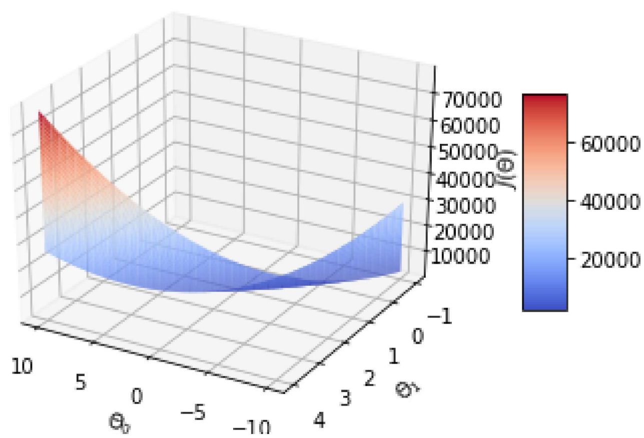
In [9]:
```
def gradientDescent(X,y,theta,alpha,num_iters):
    """
    Take numpy array for X,y,theta and update theta for every iteration of gra
dient steps
    Return theta and the list of cost of theta during each iteration
    """

    m = len(y)
    J_history = []
    for i in range(num_iters):
        predictions = X.dot(theta)
        error = np.dot(X.transpose(), (predictions-y))
        descent = alpha * 1/m * error
        theta-=descent
        J_history.append(computeCost(X,y,theta))

    return theta, J_history
```

In [10]:
```
theta, J_history = gradientDescent(X,y,theta,0.001, 2000)
```

In [11]:
```
print(f"h(x) = {str(round(theta[0,0],2))} + {str(round(theta[1,0],2))}x1")
```

```
h(x) = -1.11 + 0.92x1
```

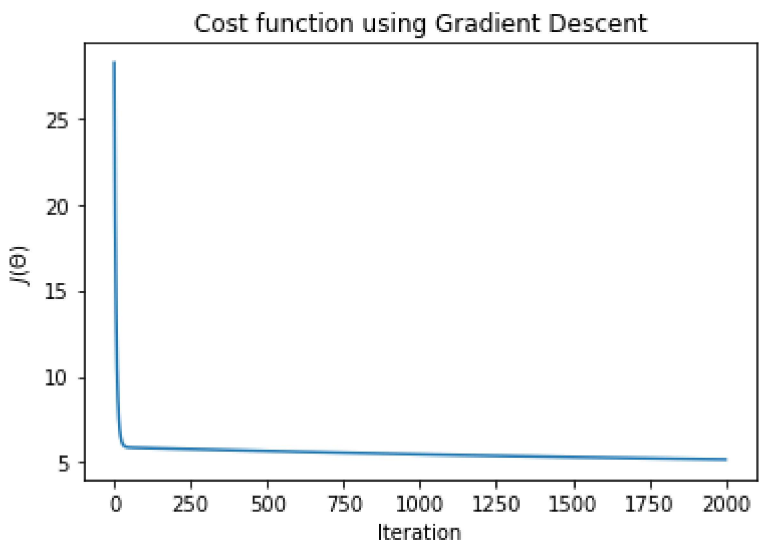
In [12]:
```
from mpl_toolkits.mplot3d import Axes3D
#Generating values for theta0, theta1 and the resulting cost value
theta0_vals=np.linspace(-10,10,100)
theta1_vals=np.linspace(-1,4,100)
J_vals=np.zeros((len(theta0_vals),len(theta1_vals)))
for i in range(len(theta0_vals)):
    for j in range(len(theta1_vals)):
        t=np.array([theta0_vals[i],theta1_vals[j]])
        J_vals[i,j]=computeCost(X,y,t)
#Generating the surface plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf=ax.plot_surface(theta0_vals,theta1_vals,J_vals,cmap="coolwarm")
fig.colorbar(surf, shrink=0.5, aspect=5)
ax.set_xlabel("$\Theta_0$")
ax.set_ylabel("$\Theta_1$")
ax.set_zlabel("$J(\Theta)$")
#rotate for better angle
ax.view_init(30,120)
```



Cost Graph

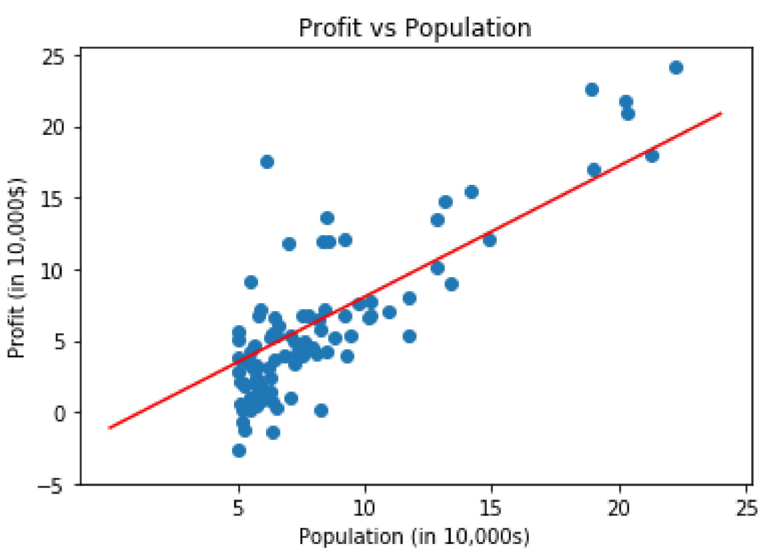
In [13]:
```
plt.plot(J_history)
plt.xlabel("Iteration")
plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
```

Out[13]: `Text(0.5, 1.0, 'Cost function using Gradient Descent')`



In [14]:
```
plt.scatter(data['Population'], data['Profit'])
x_value = [x for x in range(25)]
y_value = [x*theta[1] + theta[0] for x in x_value]
plt.plot(x_value, y_value, color = 'r')
plt.xticks(np.arange(5,30,step=5))
plt.yticks(np.arange(-5,30,step=5))
plt.xlabel('Population (in 10,000s)')
plt.ylabel('Profit (in 10,000$)')
plt.title('Profit vs Population')
```

Out[14]: `Text(0.5, 1.0, 'Profit vs Population')`



Function to predict

In [15]:
```
def predict(x,theta):
    """
    Takes in numpy array x and theta and returns predicted value of y
    """
    predictions = np.dot(theta.transpose(),x)
    return predictions[0]
```

In [16]:
```
data.tail(1)
```

Out[16]:

	Population	Profit	x0
96	5.4369	0.61705	1

In [17]:
```
predict1 = predict(data[['x0','Population']].iloc[-1].values, theta)*10000
print(f'For a population of 6170 the predicted profit is ${predict1}')
```

```
For a population of 6170 the predicted profit is $38686.246103378166
```

TO DO: Configure code for

Multivariate Linear Regression

In [18]:
```
data2=pd.read_csv("Linear Regression/ex1data2.txt",header=None)
data2.head()
```

Out[18]:

	0	1	2
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

In [19]:
```
data2.columns=['Size of house','No. of bedrooms','Price']
data2.head()
```

Out[19]:

	Size of house	No. of bedrooms	Price
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

In [20]:
```
data2.describe()
```

Out[20]:

	Size of house	No. of bedrooms	Price
count	47.000000	47.000000	47.000000
mean	2000.680851	3.170213	340412.659574
std	794.702354	0.760982	125039.899586
min	852.000000	1.000000	169900.000000
25%	1432.000000	3.000000	249900.000000
50%	1888.000000	3.000000	299900.000000
75%	2269.000000	4.000000	384450.000000
max	4478.000000	5.000000	699900.000000

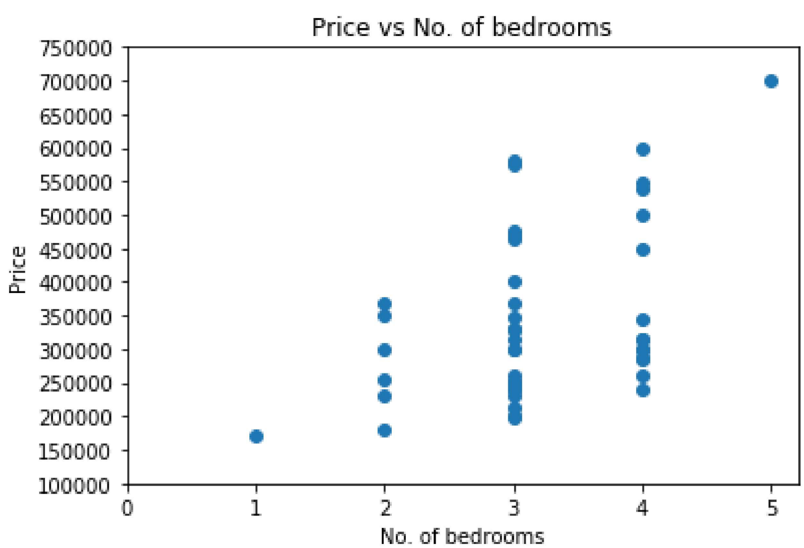
Price Vs Size of house & Price Vs No. of bedrooms

In [21]:
```
plt.scatter(data2['Size of house'], data2['Price'])
plt.xticks(np.arange (500,5000, step=500))
plt.yticks (np.arange (100000, 800000, step=50000))
plt.xlabel('Size of house')
plt.ylabel('Price')
plt.title('Price vs Size of house')
```

Out[21]:
```
Text(0.5, 1.0, 'Price vs Size of house')
```

In [22]:
```
plt.scatter(data2['No. of bedrooms'], data2['Price'])
plt.xticks(np.arange(0,6, step=1))
plt.yticks (np.arange(100000, 800000, step=50000))
plt.xlabel('No. of bedrooms')
plt.ylabel('Price')
plt.title('Price vs No. of bedrooms')
```

Out[22]:
```
Text(0.5, 1.0, 'Price vs No. of bedrooms')
```



Since column "size of house" has very high values we need to normalize it

Scaling column "size of house"

In [23]:
```
col = ['Size of house', 'Price']
scaler=MinMaxScaler()
data2[col]=pd.DataFrame(scaler.fit_transform(data2[col]), columns=data2 [col].
columns)
```

Splitting X and y from original dataset $(also\ into\ train\ and\ test)$

In [24]:
```
y = np.array(data2['Price'][:-1])
X = np.array(data2.drop('Price', axis =1)[:-1])
X.shape, y.shape
```

Out[24]:
```
((46, 2), (46,))
```

We need y shape as (46,1) and add a column of 1's in X

Reshaping X,y

In [25]:
```
y=y.reshape(y.shape[0],1)
X=np.c_[np.ones(X.shape[0]),X]
X.shape,y.shape
```

Out[25]:
```
((46, 3), (46, 1))
```

Configuring the functions $(costCompute, gradientDescent, predict)$ for Multivariate

I am writing code for costCompute and gradientDescent as taught by Siba Sir.

In [26]:
```
def computeCost(X,Y,theta):
    """
    Take in a numpy arary X,y,theta and get cost function using theta as param
eter in a linear regression model
    """
    m=Y.size
    h_theta=np.dot(X,theta) #Predictions
    error=h_theta-Y
    cost=(1/(2*m))*np.dot(error.T,error)
    return cost
```

In [27]:
```
def gradientDescent(X,Y,theta,alpha,num_iters):
    """
    Take numpy array for X,y,theta and update theta for every iteration of gra
dient steps
    Return theta and the list of cost of theta during each iteration
    """
    m = Y.size
    past_cost = []
    past_theta=[theta]
    for i in range(num_iters):
        h_theta = np.dot(X,theta)
        error=h_theta-Y
        past_cost.append(computeCost(X,Y,theta))
        theta=theta-alpha*(1/m)*np.dot(X.T,error)
        past_theta.append(theta)
        if(past_theta[i]==past_theta[i+1]).all() : break
    return past_theta, past_cost,i
```

In [28]:
```
def predict(X,theta):
    """
    Takes in numpy array x and theta and returns predicted value of y
    """
    predictions = np.dot(X,theta)
    return predictions
```

Assigning random values for θ and values for α & number of iterations we want

In [29]:
```
np.random.seed(123)
theta=np.random.rand(X.shape[1],1)
alpha=0.05
num_iters=50000
```

Performing Linear Regression and displaying results

In [30]:
```
start=time.time()
past_theta,past_cost,stop=gradientDescent(X,y,theta,alpha,num_iters)
timetaken=time.time()-start
```

In [31]:
```
best_theta=past_theta[-1]
best_cost=past_cost[-1]
```

In [32]:
```
print(f'Our model performed {stop} iterations out of {num_iters} iterations be
fore converging (the previous θ was equal to current θ)')
```

```
Our model performed 21483 iterations out of 50000 iterations before convergin
g (the previous θ was equal to current θ)
```

In [33]:
```
print(f'Best Theta: \n {best_theta}')
print(f'Best Cost: \n {best_cost}')
```

```
Best Theta:
 [[ 0.07198606]
 [ 0.95458358]
 [-0.01672784]]
Best Cost:
 [[0.00742914]]
```

In [34]:
```
print(f'h(θ) = {str(round(best_theta[0,0],5))} + {str(round(best_theta[1,0],
5))}x1 + {str(round(best_theta[2,0],5))}x2')
print(f'\nTime taken :{timetaken}')
print(f'\nAccuracy: {round(r2_score(y,predict(X,best_theta)),4)*100}%')
```

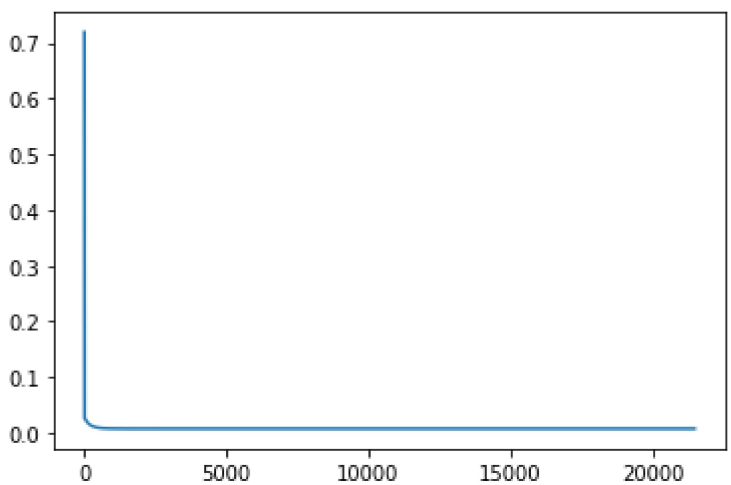
```
h(θ) = 0.07199 + 0.95458x1 + -0.01673x2

Time taken :0.501678466796875

Accuracy: 72.91%
```

Plotting Cost

In [35]:
```
cost = np.asarray(past_cost)
cost = cost.reshape((cost.shape[0],1))
cost.shape
```

Out[35]: (21484, 1)

In [36]:
```
plt.plot(cost)
plt.show()
```



Predicting for test data

In [37]:
```
data2.tail(1)
```

Out[37]:

	Size of house	No. of bedrooms	Price
46	0.096801	3	0.131321

In [38]:
```
predict2=predict(X[-1], best_theta)*1000000
print(f'For house of size 968 sq.ft. the predicted price is :${predict2}')
```

```
For house of size 968 sq.ft. the predicted price is :$[268335.49803214]
```

Comparing parameters

In [39]:
```
print(f'Parameters from StatsModels -> {sm.OLS(y,X).fit() .params}')
print(f'Parameters from ScikitLearn -> {LinearRegression().fit(X,y).coef_}')
print(f'Parameters from Gradient Descent -> {best_theta.reshape(3,)}')
```

```
Parameters from StatsModels -> [ 0.07198606  0.95458358 -0.01672784]
Parameters from ScikitLearn -> [[ 0.          0.95458358 -0.01672784]]
Parameters from Gradient Descent -> [ 0.07198606  0.95458358 -0.01672784]
```

Here we can see that the results match

Code for Linear Regression

if we want to put the splitting data into X,y and its reshaping into a function too

In [40]:
```
def LinearReg_GD (data, pred_col, alpha, num_iters, dec = 5):
    #Declaring X,y and theta
    y1 = np.array(data[pred_col]).reshape((-1,1))
    X1 = np.array(data.drop(pred_col, axis = 1))
    X1 = np.c_[np.ones (X1.shape[0]), X1]
    theta = np.zeros((X1.shape[1],1))
    return gradientDescent(X1,y1,theta,alpha,num_iters)
```

In [41]:
```
np.random.seed(123)
theta1=np.random.rand(X.shape[1],1)
alpha=0.05
num_iters=50000
past_theta1,past_cost1,stop1=LinearReg_GD(data2,'Price',alpha,num_iters)
```

In [42]:
```
best_theta1=past_theta1[-1]
best_cost1=past_cost1[-1]

print(f'Our model performed {stop1} iterations out of {num_iters} iterations b
efore converging (the previous θ was equal to current θ)\n\n')

print(f'Best Theta: \n {best_theta1}')
print(f'Best Cost: \n {best_cost1}')

print(f'\n\nh(θ) = {str(round(best_theta1[0,0],5))} + {str(round(best_theta1
[1,0],5))}x1 + {str(round(best_theta1[2,0],5))}x2')
```

```
Our model performed 22195 iterations out of 50000 iterations before convergin
g (the previous θ was equal to current θ)


Best Theta:
 [[ 0.07227435]
 [ 0.95241114]
 [-0.01648683]]
Best Cost:
 [[0.00727405]]


h(θ) = 0.07227 + 0.95241x1 + -0.01649x2
```