Name : Sameeksha Kini

Roll No. : J029

B.Tech (Data Science) - 3rd Year

ML - Linear Regression

In [1]:
```
%matplotlib inline
import pandas as pd
import numpy as np
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]:
```
data1=pd.read_csv("C:\\Users\\samee\\Downloads\\ex1data1.txt", header=None)
data1.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]:
```
data1.describe()
```

Out[3]:

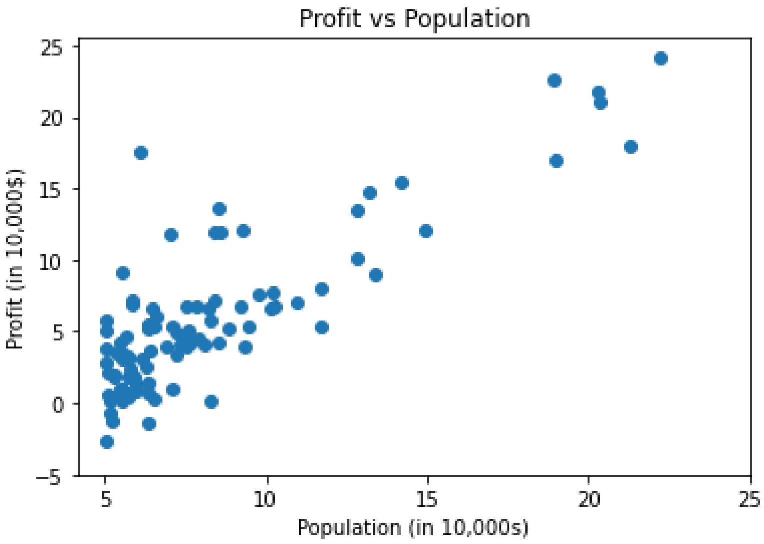
	0	1
count	97.000000	97.000000
mean	8.159800	5.839135
std	3.869884	5.510262
min	5.026900	-2.680700
25%	5.707700	1.986900
50%	6.589400	4.562300
75%	8.578100	7.046700
max	22.203000	24.147000

In [4]:
```
data1.columns = ['Population', 'Profit'] #assigning column names
```

Profit vs. Population graph

In [5]:
```
plt.scatter(data1['Population'], data1['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 parameter i
    """
    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]:
```
data1['x0'] = 1
data_val = data1.values
m =  len(data_val[:-1])
X = data1[['x0','Population']].iloc[:-1].values
y = data1['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))`

$$h(\theta) = x_0\theta_0 + x_1\theta_1 \ldots .(x0 = 1)$$

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

Out[8]: `32.40484177877031`

Gradient Descent Function

```
def GradientDescent(X,y,theta,alpha,num_iters):
```

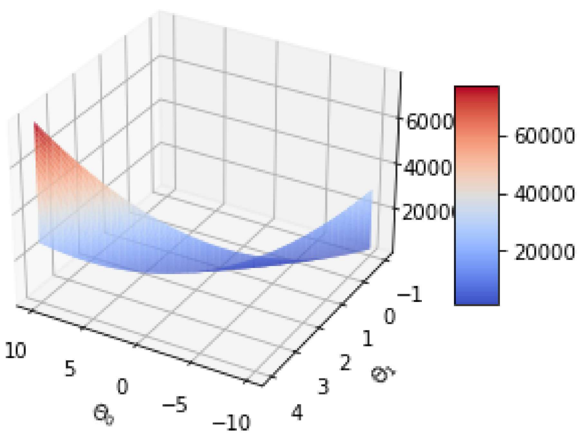
In [9]:
```
    """
    Take numpy array for X,y,theta and update theta for every iteration of gradient

    return theta and the list of cost of theta during each iteration
    """

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

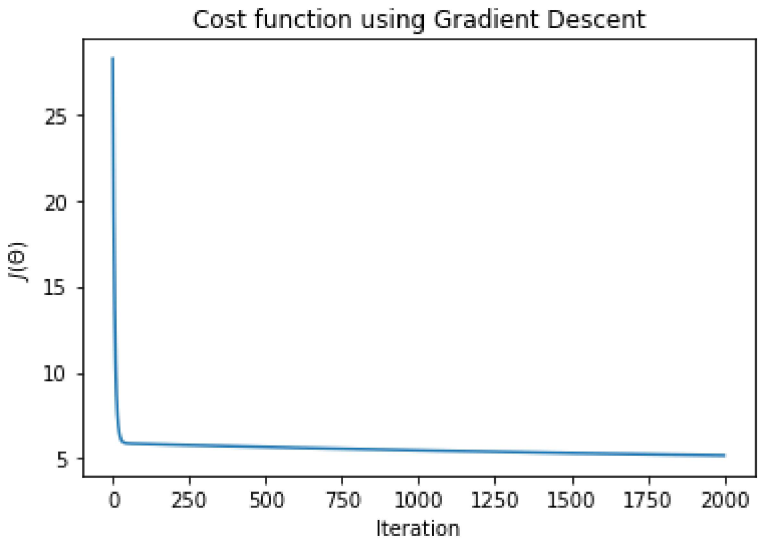
In [12]:
```
from mpl_toolkits.mplot3d import Axes3D
#Generating values for theta0, theta1 and the resulting cost value
theta0_vals=np.linspace(-10,10,100)
theta1_vals=np.linspace(-1,4,100)
J_vals=np.zeros((len(theta0_vals),len(theta1_vals)))
for i in range(len(theta0_vals)):
    for j in range(len(theta1_vals)):
        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

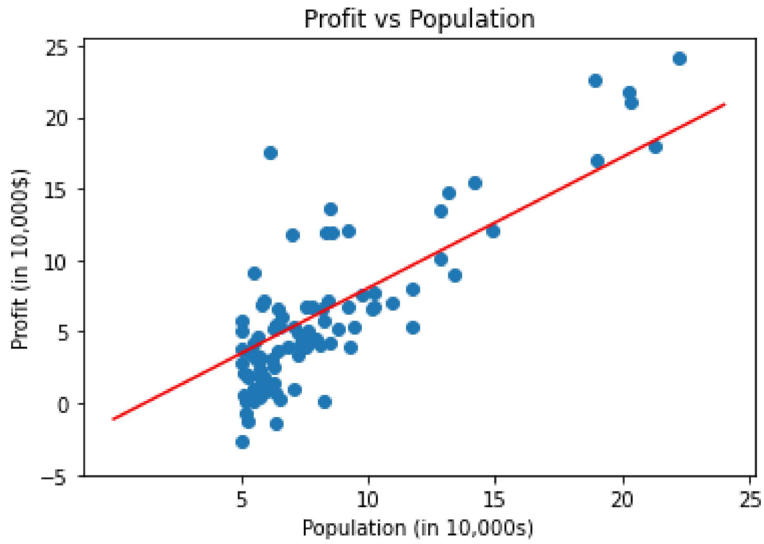
In [13]:
```
plt.plot(J_history)
plt.xlabel("Iteration")
plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
```

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



In [14]:
```
plt.scatter(data1['Population'], data1['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')



Prediction Function

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

In [16]:
```
data1.tail(1)
```

Out[16]:

	Population	Profit	x0
96	5.4369	0.61705	1

In [17]:
```
predict1 = predict(data1[['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 $[38686.24610338]
```

Multivariate Linear Regression

In [22]:
```
data2=pd.read_csv("C:\\Users\\samee\\Downloads\\ex1data2.txt", header=None)
data2.head()
```

Out[22]:

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

In [23]:
```
data2.columns = ['Size of house', 'No. of bedrooms', 'Price']
data2.head()
```

Out[23]:

	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 [24]:
```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47 entries, 0 to 46
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Size of house    47 non-null     int64
 1   No. of bedrooms  47 non-null     int64
 2   Price            47 non-null     int64
dtypes: int64(3)
memory usage: 1.2 KB
```

In [25]:
```
data2.describe()
```

Out[25]:

	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

	Size of house	No. of bedrooms	Price
50%	1888.000000	3.000000	299900.000000
75%	2269.000000	4.000000	384450.000000
max	4478.000000	5.000000	699900.000000

Scaling the columns with high values to values between 0 to 1

In [26]:
```
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

In [27]:
```
y = np.array(data2['Price'][:-1])
X = np.array(data2.drop('Price', axis = 1)[:-1])
X.shape, y.shape
```

Out[27]:
```
((46, 2), (46,))
```

Reshaping X and y according to right dimensions

In [28]:
```
y = y.reshape(y.shape[0],1)
X = np.c_[np.ones(X.shape[0]), X]
X.shape, y.shape
```

Out[28]:
```
((46, 3), (46, 1))
```

Assigning values for θ

In [29]:
```
theta = np.zeros((3,1))
theta.shape
```

Out[29]:
```
(3, 1)
```

Performing Regression

In [30]:
```
alpha = 0.01
num_iters = 50000
theta, J_history = GradientDescent(X,y,theta,alpha, num_iters)
prediction = predict(X,theta)
```

In [31]:
```
print(f"h(x) = {str(round(theta[0,0],2))}+{str(round(theta[1,0],2))}x1{str(round(the
```

```
h(x) = 0.07+0.95x1-0.02x2
```

Precting the value

In [32]:
```
data2.tail(1)
```

Out[32]:

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

In [33]:
```
predict2 = predict(X[-1], theta)*1000000
print(f'For house of size 968 sq.ft. the predicted price is : ${predict2}')
```

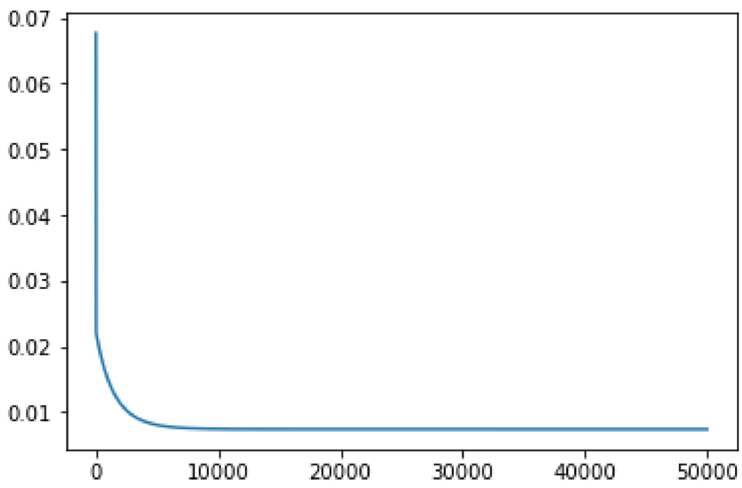
```
For house of size 968 sq.ft. the predicted price is : $[268335.61509465]
```

Cost graph

In [34]:
```
cost_arr = np.asarray(J_history)
cost_arr = cost_arr.reshape((cost_arr.shape[0],1))
cost_arr.shape
```

Out[34]:
```
(50000, 1)
```

In [35]:
```
plt.plot(cost_arr)
plt.show()
```



Comparison of parameters

In [37]:
```
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 -> {theta}')
```

```
Parameters from StatsModels -> [ 0.07198606  0.95458358 -0.01672784]
Parameters from SciKitLearn -> [[ 0.          0.95458358 -0.01672784]]
Parameters from Gradient Descent -> [[ 0.07198582]
 [ 0.95458309]
 [-0.01672771]]
```