Notebook:

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sn
import random
import math
```

In [2]:
```python
df=pd.read_csv('./Advertising.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  200 non-null    int64  
 1   TV          200 non-null    float64
 2   radio       200 non-null    float64
 3   newspaper   200 non-null    float64
 4   sales       200 non-null    float64
dtypes: float64(4), int64(1)
memory usage: 7.9 KB
```

X would represent TV, Radio and Newspaper while Y would represent our sales. As all these sales might be on different scales, we then normalise our X & Y variables.

In [3]:
```python
X=df[['TV','radio','newspaper']]
Y=df['sales']
Y=np.array((Y-Y.mean())/Y.std())
X=X.apply(lambda rec:(rec-rec.mean())/rec.std(),axis=0)
```

To implement a gradient descent algorithm we need to follow 4 steps:
1)Randomly initialize the bias and the weight theta
2)Calculate predicted value of y that is Y given the bias and the weight
3)Calculate the cost function from predicted and actual values of Y
4)Calculate gradient and the weights

In [4]:
```python
def initialize(dim):
    b=random.random()
    theta=np.random.rand(dim)
    return b,theta
b,theta=initialize(3)
print("Bias: ",b,"Weights: ",theta)
```

```
Bias:  0.5283414452076892 Weights:  [0.05177122 0.19233261 0.1127032
4]
```

In [5]:
```
def predict_Y(b,theta,X):
    return b + np.dot(X,theta)
Y_hat=predict_Y(b,theta,X)
Y_hat[0:10]
```

Out[5]:
```
array([0.9667235 , 0.74951975, 0.94379078, 0.90929539, 0.5313347 ,
       1.0070258 , 0.56137594, 0.3666039 , 0.0177499 , 0.24405717])
```

In [6]:
```
def get_cost(Y,Y_hat):
    Y_resd=Y-Y_hat
    return np.sum(np.dot(Y_resd.T,Y_resd))/len(Y-Y_resd)
get_cost(Y,Y_hat)
```

Out[6]:
```
0.990939665283301
```

In [7]:
```
def update_theta(x,y,y_hat,b_0,theta_0,learning_rate):
    db=(np.sum(y_hat-y)*2)/len(y)
    dw=(np.dot((y_hat-y),x)*2)/len(y)
    b_1=b_0-learning_rate*db
    theta_1=theta_0-learning_rate*dw
    return b_1,theta_1
print("After initialization :- Bias: ",b,"theta: ",theta)
Y_hat=predict_Y(b,theta,X)
b,theta=update_theta(X,Y,Y_hat,b,theta,0.001)
print("After first update :- Bias: ",b,"theta: ",theta)
get_cost(Y,Y_hat)
```

```
After initialization :- Bias:  0.5283414452076892 theta:  [0.0517712
2 0.19233261 0.11270324]
After first update :- Bias:  0.5272847623172738 theta:  [0.05319113
0.19301149 0.11279191]
```

Out[7]:
```
0.990939665283301
```

In [8]:
```
def gradient_descent(X,Y,alpha,num_iterations):
    b,theta=initialize(X.shape[1])
    iter_num=0
    gd_iterations_df=pd.DataFrame(columns=['iteration','cost'])
    result_idx=0
    for each_iter in range(num_iterations):
        Y_hat=predict_Y(b,theta,X)
        this_cost=get_cost(Y,Y_hat)
        prev_b=b
        prev_theta=theta
        b,theta=update_theta(X,Y,Y_hat,prev_b,prev_theta,alpha)
        if(iter_num%10==0):
            gd_iterations_df.loc[result_idx]=[iter_num,this_cost]
        result_idx +=1
        iter_num +=1
    print("Final Estimate of b and theta : ",b,theta)
    return gd_iterations_df,b,theta
gd_iterations_df,b,theta=gradient_descent(X,Y,alpha=0.001,num_iterati
ons=2000)
```

```
Final Estimate of b and theta :  0.004252215719740022 [ 0.74798846
0.54105806 -0.00515033]
```

In [9]:
```
gd_iterations_df[0:10]
```

Out[9]:

	iteration	cost
0	0.0	0.414257
10	10.0	0.399586
20	20.0	0.385622
30	30.0	0.372328
40	40.0	0.359674
50	50.0	0.347627
60	60.0	0.336158
70	70.0	0.325237
80	80.0	0.314840
90	90.0	0.304939

Therefore if we print the cost function for each iteration we can see the decrease in the cost function.

In [10]:
```
alpha_values = [0.001, 0.005, 0.01, 0.05, 0.1]
num_iterations = 2000
for alpha in alpha_values:
    alpha_df,b,theta = gradient_descent(X, Y, alpha, num_iterations)
    plt.plot(alpha_df['iteration'], alpha_df['cost'], label=f'alpha = {alpha}')
plt.xlabel('Number of iterations')
plt.ylabel('Cost')
plt.title('Gradient Descent - Cost vs Iterations')
plt.legend()
plt.show()
```

```
Final Estimate of b and theta :  0.003652856139686653 [0.75078897 0.53044134 0.00519106]
Final Estimate of b and theta :  1.837706544668351e-09 [ 0.75306591 0.53648114 -0.00433027]
Final Estimate of b and theta :  -5.101298377136599e-17 [ 0.75306591 0.53648155 -0.00433069]
Final Estimate of b and theta :  -4.121159063147589e-17 [ 0.75306591 0.53648155 -0.00433069]
Final Estimate of b and theta :  -4.022116668705733e-17 [ 0.75306591 0.53648155 -0.00433069]
```