In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('cars24-car-price-clean.csv')
```

In [ ]:
```
df.shape
```

In [ ]:
```
df.head()
```

In [ ]:
```
# Uni-variate
X = df['max_power'].values

# target
Y = df['selling_price'].values
```

In [ ]:
```
X.shape
```

In [ ]:
```
X.ndim
```

In [ ]:
```
X = X.reshape(-1,1)
```

In [ ]:
```
X.shape
```

In [ ]:
```
X.ndim
```

In [ ]:
```
Y.ndim
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
model = LinearRegression()
```

In [ ]:
```
# this is where the entire training happens
model.fit(X, Y)
```

In [ ]:
```
model.coef_
```

In [ ]:
```
model.intercept_
```

In [ ]:
```
type(model)
```

In [ ]:
```
x_query = np.array([1.5])
x_query
```

In [ ]:
```
model.predict(x_query.reshape(-1, 1))
```

In [ ]:
```
y_pred = model.predict(X)
y_pred[:5]
```

In [ ]:
```
Y[:5]
```

In [ ]:
```
model.score(X, Y)
```

Multiple Linear Regression

In [3]:
```
df = pd.read_csv('cars24-car-price-clean.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	selling_price	year	km_driven	mileage	engine	max_power	age	make	model	Individual	Trustmark Dealer	Diesel	Electric	LPG
0	-1.111046	-0.801317	1.195828	0.045745	-1.310754	-1.157780	0.801317	-0.433854	-1.125683	1.248892	-0.098382	-0.985275	-0.020095	-0.056917
1	-0.223944	0.450030	-0.737872	-0.140402	-0.537456	-0.360203	-0.450030	-0.327501	-0.333227	1.248892	-0.098382	-0.985275	-0.020095	-0.056917
2	-0.915058	-1.426990	0.035608	-0.582501	-0.537456	-0.404885	1.426990	-0.327501	-0.789807	1.248892	-0.098382	-0.985275	-0.020095	-0.056917
3	-0.892365	-0.801317	-0.409143	0.329620	-0.921213	-0.693085	0.801317	-0.433854	-0.905265	1.248892	-0.098382	-0.985275	-0.020095	-0.056917
4	-0.182683	0.137194	-0.544502	0.760085	0.042999	0.010435	-0.137194	-0.246579	-0.013096	-0.800710	-0.098382	1.014945	-0.020095	-0.056917

In [5]:
```
X = df.drop('selling_price', axis=1).values
```

In [6]:
```
X.shape
```

Out[6]: (19820, 17)

In [7]:
```
ones = np.ones((19820, 1))
X_new = np.hstack((ones,X))
X_new.shape
```

Out[7]: (19820, 18)

In [8]:
```
X_new[:2]
```

Out[8]:
```
array([[ 1.        , -0.80131654,  1.19582817,  0.04574517, -1.31075443,
        -1.15777962,  0.80131654, -0.43385435, -1.12568266,  1.24889206,
        -0.09838223, -0.9852749 , -0.02009467, -0.0569168 ,  1.0246219 ,
         0.4958182 ,  0.44450319, -0.42472845],
       [ 1.        ,  0.45003028, -0.73787208, -0.14040198, -0.53745638,
        -0.36020313, -0.45003028, -0.32750073, -0.3332271 ,  1.24889206,
        -0.09838223, -0.9852749 , -0.02009467, -0.0569168 ,  1.0246219 ,
         0.4958182 ,  0.44450319, -0.42472845]])
```

In [9]:
```
Y = df['selling_price'].values
Y = Y.reshape(-1, 1)
Y.shape
```

Out[9]: (19820, 1)

In [ ]:

In [10]:
```
def hypothesis(X, weights):
    '''
    X :  (n, d+1)
    weights : (d+1, 1)
    '''
    return np.dot(X, weights)
```

In [11]:
```
def error(X, Y, weights):
    '''
    X :  (n, d+1)
    Y : (n, 1)
    weights : (d+1, 1)
    '''
    Y_hat = hypothesis(X, weights)
    err = np.mean((Y - Y_hat)**2)
    return err
```

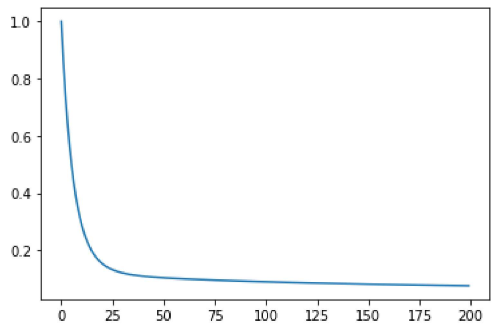
In [12]:
```
def gradients(X, Y, weights):
    Y_hat = hypothesis(X, weights)
    grads = np.dot( X.T ,  (Y_hat - Y )    )
    return 2*grads/len(Y)
```

In [16]:
```
def gradient_descent(X, Y, max_itr = 200, learning_rate = 0.01):

    # step 1 : init() randomly
    weights = np.zeros((X.shape[1], 1))
    error_list = []

    # step 2 repeate until convergence
    for i in range(max_itr):

        e = error(X, Y, weights)
        error_list.append(e)

        grads = gradients(X, Y, weights)

        weights = weights - learning_rate*grads



    return weights, error_list
```

In [17]:
```
opt_weights, error_list = gradient_descent(X_new, Y)
```

In [18]:
```
plt.plot(error_list)
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x7fd4b0d1ff70>]
```



In [39]:
```
opt_weights.round(2)
```

Out[39]:
```
array([[ 0.  ],
       [ 0.11],
       [-0.03],
       [-0.05],
       [ 0.07],
       [ 0.13],
       [-0.11],
       [ 0.18],
       [ 0.47],
       [-0.02],
       [-0.01],
       [ 0.04],
       [ 0.02],
       [ 0.01],
       [-0.03],
       [-0.07],
       [-0.  ],
       [ 0.  ]])
```

In [42]:
```python
# feature importances.
np.abs(opt_weights.round(2))
```

Out[42]:
```
array([[0.  ],
       [0.11],
       [0.03],
       [0.05],
       [0.07],
       [0.13],
       [0.11],
       [0.18],
       [0.47],
       [0.02],
       [0.01],
       [0.04],
       [0.02],
       [0.01],
       [0.03],
       [0.07],
       [0.  ],
       [0.  ]])
```

In [22]:
```python
y_pred = hypothesis(X_new, opt_weights)
```

In [25]:
```python
y_pred[:5]
```

Out[25]:
```
array([[-1.20007006],
       [-0.3020007 ],
       [-0.95360925],
       [-0.98039799],
       [ 0.01098332]])
```

In [26]:
```python
Y[:5]
```

Out[26]:
```
array([[-1.11104589],
       [-0.22394353],
       [-0.91505816],
       [-0.89236484],
       [-0.18268296]])
```

In [29]:
```python
def r2_score(Y, Y_hat):
    num =  np.sum((Y - Y_hat)**2)
    denom = np.sum((Y - Y.mean() )**2)
    return (1 - num/denom)
```

In [30]:
```python
r2_score(Y, y_pred)
```

Out[30]:
```
0.9240707427695862
```

In [ ]:

In [37]:
```python
n = X.shape[0]
d = X.shape[1]


adj_r2 = 1- ((1- r2_score(Y, y_pred))*(n-1)/ (n-d-1))
print(adj_r2)
```

```
0.9240055575674391
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```