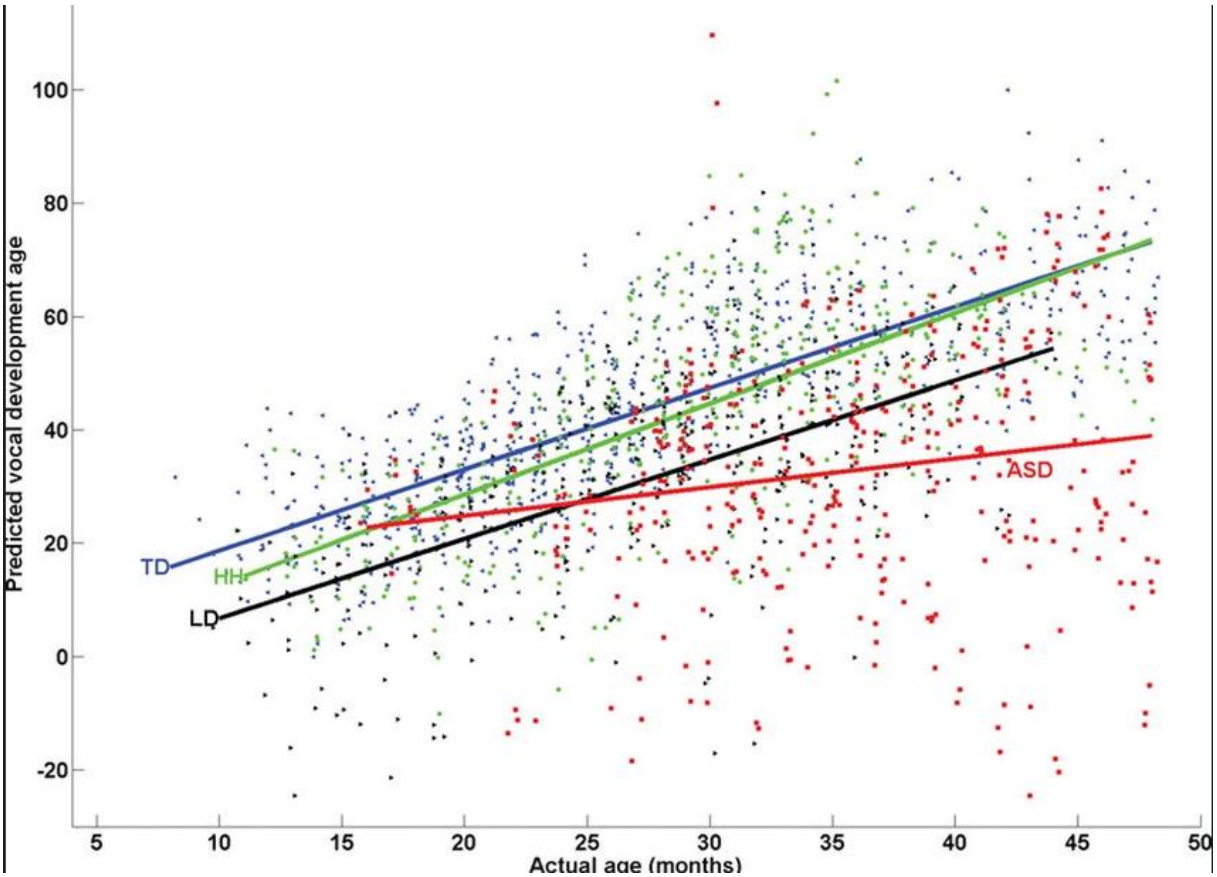
Çok Değişkenli Doğrusal Regresyon

Çok Değişkenli Doğrusal Regresyon nedir?

Çoklu doğrusal regresyon, iki veya daha fazla değişkenin değerine dayanarak bir değişkenin sonucunu tahmin etmek için kullanılan istatistiksel bir tekniktir. Bazı durumlarda sadece çoklu regresyon olarak da bilinir ve doğrusal regresyonun bir uzantısıdır. Tahmin etmek istediğimiz değişken bağımlı değişken olarak bilinirken, bağımlı değişkenin değerini tahmin etmek için kullandığımız değişkenler bağımsız veya açıklayıcı değişkenler olarak bilinir.

Şekil: Bireysel gözlemler için çoklu doğrusal regresyon modeli tahminleri



Özet

Çoklu doğrusal regresyon, iki veya daha fazla bağımsız değişkenin bağımlı bir değişkenin sonucunu tahmin etmek için kullanılan bir istatistiksel tekniktir. Bu teknik, analistlere modelin değişkenliğini belirlemelerine ve her bağımsız değişkenin toplam değişkenlik içindeki göreceli katkısını belirlemelerine olanak tanır. Çoklu regresyon iki şekilde olabilir, yani doğrusal regresyon ve doğrusal olmayan regresyon.

Çok Değişkenli Doğrusal Regresyon Formülü

$Y = M0X0 + M1X1 + M2X2 + M3X3 + ... + MkXk + \epsilon$

Bu denklem, çoklu doğrusal regresyon modelini temsil eder. İlgili terimlerin açıklamaları şu şekildedir:

- Y, bağımlı veya tahmin edilen değişkendir.
- M0, y-kesişimini temsil eder, yani hem M1 hem de M2 0 olduğunda y'nin değeridir.
- M1 ve M2, sırasıyla X1 ve X2'deki bir birimlik değişime göre y'deki değişimi temsil eden regresyon katsayılarıdır.
- Mk, her bağımsız değişken için eğim katsayısıdır.
- ϵ, modelin rastgele hata (artık) terimidir. Bu terim, modelin gerçek verilere tam olarak uymadığı hataları temsil eder.

Çoklu Doğrusal Regresyonun Anlaşılması

Basit doğrusal regresyon, istatistikçilere bir değişkenin değerini başka bir değişken hakkında mevcut bilgileri kullanarak tahmin etme imkanı sunar. Doğrusal regresyon, iki değişken arasındaki ilişkiyi bir doğru çizgide belirlemeye çalışır.

Çoklu regresyon, bağımlı değişkenin iki veya daha fazla bağımsız değişkenle doğrusal bir ilişki gösterdiği bir regresyon türüdür. Ayrıca, bağımlı ve bağımsız değişkenlerin doğrusal bir çizgi izlemediği non-lineer olabilir.

Hem doğrusal hem de non-lineer regresyon, grafiksel olarak iki veya daha fazla değişkeni belirli bir yanıtı takip etmek için izler. Ancak, non-lineer regresyon genellikle uygulaması zor olabilir, çünkü deneme yanılma yoluyla elde edilen varsayımlardan oluşturulmuştur.

Çoklu Doğrusal Regresyon Uygulama

1 - burada Boston veri kümesi ile ilgileniyoruz.

Adımlar:

Step 1 --> Boston verisetini yükle

Step 2 --> Sıfırdan Çoklu Doğrusal Regresyon Uygulamak

Step 1 --> Vektörleştirme ile kodu optimize etmek

Step 1 --> Başarı Ölçütü(SCORE)

STEP-1

In [1]:
```
from sklearn.datasets import load_boston
import pandas as pd
```

In [2]:
```
boston = load_boston()

X = boston.data
y = boston.target
```

In [3]:
```
print(X.shape)
print(y.shape)
```

```
(506, 13)
(506,)
```

In [4]:
```
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [5]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is us
ually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

In [6]:
```
import pandas as pd
df = pd.DataFrame(X)
df.columns = boston.feature_names
df.head()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [7]:
```
df.describe()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.54940
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.70725
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.00000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.00000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.00000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.0000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.0000

In [8]:
```
# Normalise this dataset
# Each feature must have 0 mean, unit variance
import numpy as np
u = np.mean(X,axis=0)
std = np.std(X,axis=0)
#print(u.shape,std.shape)
```

In [9]:
```
# Normalise the Data
X = (X-u)/std
```

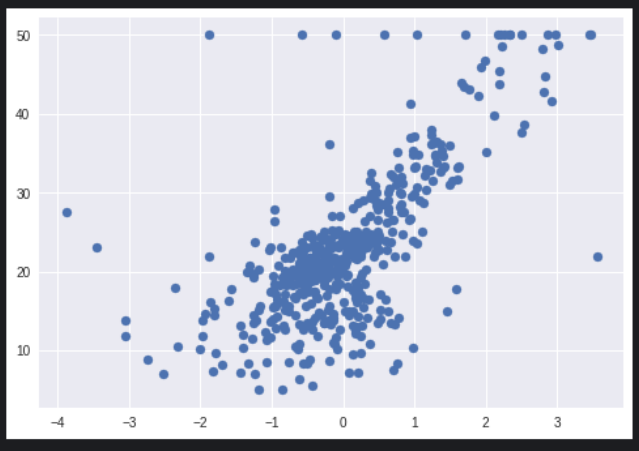
In [10]:
```
# Normalised Data
pd.DataFrame(X[:5,:]).head()
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982843	-0.666608
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867883	-0.987329
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867883	-0.987329
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752922	-1.106115
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752922	-1.106115

In [11]:
```
# Plot Y vs any feature
import matplotlib.pyplot as plt

plt.style.use('seaborn')
plt.scatter(X[:,5],y)
plt.show()
```



STEP-2

In [12]:
```
X.shape, y.shape
```

Out[12]:
```
((506, 13), (506,))
```

In [13]:
```
ones = np.ones((X.shape[0],1))
X = np.hstack((ones,X))
print(X.shape)
```

```
(506, 14)
```

In [14]:
```
print(X[3])
```

```
[ 1.         -0.41675042 -0.48772236 -1.30687771 -0.27259857 -0.83528384
  1.01630251 -0.80988851  1.07773662 -0.75292215 -1.10611514  0.1130321
  0.41616284 -1.36151682]
```

In [15]:
```
# X - Matrix ( m x n)
# x - Vector (Single Example with n features)

def hypothesis(x,theta):
    y_ = 0.0
    n = x.shape[0]
    for i in range(n):
        y_  += (theta[i]*x[i])
    return y_

def error(X,y,theta):
    e = 0.0
    m = X.shape[0]

    for i in range(m):
        y_ = hypothesis(X[i],theta)
        e += (y[i] - y_)**2

    return e/m

def gradient(X,y,theta):
    m,n = X.shape

    grad = np.zeros((n,))

    # for all values of j
    for j in range(n):
        #sum over all examples
        for i in range(m):
            y_ = hypothesis(X[i],theta)
            grad[j] += (y_ - y[i])*X[i][j]
    # Out of the loops
    return grad/m

def gradient_descent(X,y,learning_rate=0.1,max_epochs=300):
    m,n = X.shape
    theta = np.zeros((n,))
    error_list = []

    for i in range(max_epochs):
        e = error(X,y,theta)
        error_list.append(e)

        # Gradient Descent
        grad = gradient(X,y,theta)
        for j in range(n):
            theta[j] = theta[j] - learning_rate*grad[j]

    return theta,error_list
```

In [16]:
```
import time
start = time.time()
theta,error_list = gradient_descent(X,y)
end = time.time()
print("Time taken is ", end-start)
```

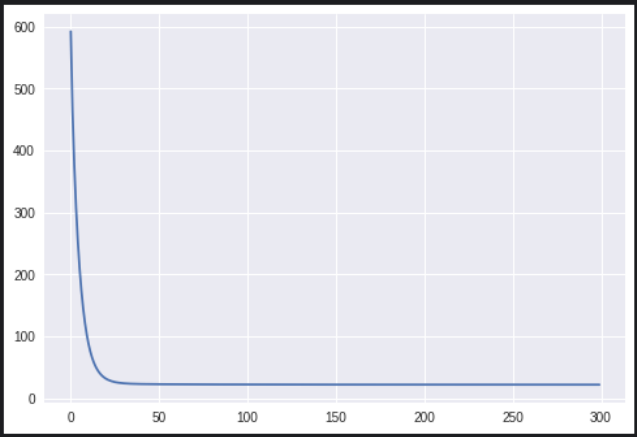
```
Time taken is  41.22899150848389
```

QUITE SLOW 18 SECONDS JUST 506 EXAMPLES IN THE TRAINING DATA

In [17]:
```
print(theta)
```

```
[ 2.25328063e+01 -9.03091692e-01  1.03815625e+00  1.53477685e-02
  6.99554920e-01 -2.02101672e+00  2.70014278e+00 -1.93085233e-03
 -3.10234837e+00  2.34354753e+00 -1.72031485e+00 -2.04614394e+00
  8.47845679e-01 -3.73089521e+00]
```

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



prediction

In [19]:
```
y_ = []
m = X.shape[0]

for i in range(m):
    pred = hypothesis(X[i],theta)
    y_.append(pred)
y_ = np.array(y_)
```

In [20]:
```
def r2_score(y,y_):
    num = np.sum((y-y_)**2)
    denom = np.sum((y- y.mean())**2)
    score = (1- num/denom)
    return score*100
```

In [21]:
```
# SCORE
r2_score(y,y_)
```

Out[21]:
```
74.04541323942743
```

STEP-3

In [22]:

```
def hypothesis(X,theta):
    return np.dot(X,theta)

def error(X,y,theta):
    e = 0.0
    y_ = hypothesis(X,theta)
    e = np.sum((y-y_)**2)

    return e/m

def gradient(X,y,theta):

    y_ = hypothesis(X,theta)
    grad = np.dot(X.T,(y_ - y))
    m = X.shape[0]
    return grad/m

def gradient_descent(X,y,learning_rate = 0.1,max_iters=300):

    n = X.shape[1]
    theta = np.zeros((n,))
    error_list = []

    for i in range(max_iters):
        e = error(X,y,theta)
        error_list.append(e)

        #Gradient descent
        grad = gradient(X,y,theta)
        theta = theta - learning_rate*grad

    return theta,error_list
```

In [23]:

```
start = time.time()
theta,error_list = gradient_descent(X,y)
end = time.time()
print("Time taken by Vectorized Code",end-start)
```

```
Time taken by Vectorized Code 0.017749309539794922
```

really very fast! :D

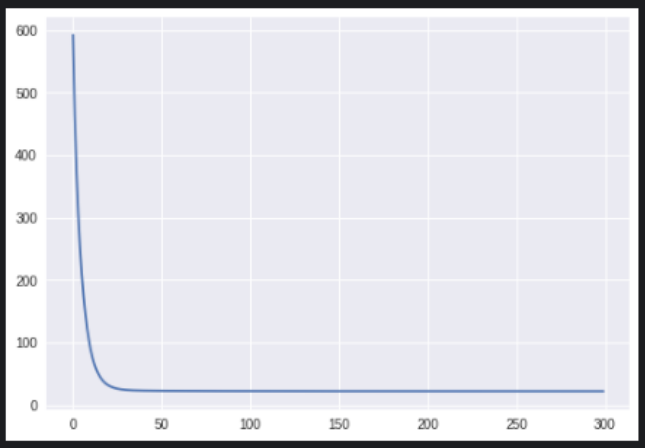
In [24]:

```
theta
```

Out[24]:

```
array([ 2.25328063e+01, -9.03091692e-01,  1.03815625e+00,  1.53477685e-02,
        6.99554920e-01, -2.02101672e+00,  2.70014278e+00, -1.93085233e-03,
       -3.10234837e+00,  2.34354753e+00, -1.72031485e+00, -2.04614394e+00,
        8.47845679e-01, -3.73089521e+00])
```

In [25]:
```
plt.plot(error_list)
plt.show()
```



STEP-4

In [26]:
```
y_ = hypothesis(X,theta)
r2_score(y,y_)
```

Out[26]:
```
74.04541323942743
```

KAYNAK

https://www.kaggle.com/code/abheeshthmishra/multiple-linear-regression