In [1]:
```
# MA3832 A1
# ZHANGJIAYU 13851049
```

In [2]:
```
# import library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [3]:
```
# Read dataset
data = pd.read_csv('marketing.csv',sep = ',')
data.head()
```

Out[3]:

	youtube	facebook	newspaper	sales
0	276.12	45.36	83.04	26.52
1	53.40	47.16	54.12	12.48
2	20.64	55.08	83.16	11.16
3	181.80	49.56	70.20	22.20
4	216.96	12.96	70.08	15.48

In [4]:
```
# check missing value
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   youtube    200 non-null    float64
 1   facebook   200 non-null    float64
 2   newspaper  200 non-null    float64
 3   sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

In [12]:
```
# Split test and train set
train_set, test_set = train_test_split(data,test_size = 0.3)
print(train_set.shape,test_set.shape)
```

```
(140, 4) (60, 4)
```

In [13]:
```
# Split X,Y
Xtrain_set = train_set.iloc[:,[0,1,2]]
Xtrain_set = (Xtrain_set - Xtrain_set.min()) / (Xtrain_set.max() - Xtrain_set.min())
Xtrain_set['constant']=1

Ytrain_set = train_set.iloc[:,[3]]
print(Xtrain_set.head())
print(Ytrain_set.head())
```

```
      youtube  facebook  newspaper  constant
91   0.083818  0.030242   0.287599         1
2    0.044817  0.925403   0.606860         1
18   0.222716  0.413306   0.158311         1
71   0.361615  0.288306   0.276165         1
164  0.386931  0.296371   0.044855         1
     sales
91    8.76
2    11.16
18   13.56
71   14.88
164  14.28
```

In [14]:
```
Xtest_set = test_set.iloc[:,[0,1,2]]
Xtest_set = (Xtest_set - Xtest_set.min()) / (Xtest_set.max() - Xtest_set.min())
Xtest_set['constant']=1
```

```
Ytest_set = test_set.iloc[:,[3]]
print(Xtest_set.head())
print(Ytest_set.head())
```

```
      youtube  facebook  newspaper  constant
120  0.480027  0.538776   0.573643         1
117  0.258450  0.008163   0.167959         1
42   1.000000  0.557143   0.000000         1
37   0.252646  1.000000   0.567183         1
69   0.737794  0.887755   0.328165         1
     sales
120  18.60
117  11.28
42   24.84
37   17.64
69   26.76
```

In [15]:

```
# transform dataframe into array
X1 = np.array(Xtrain_set)

print(X1[0:5])
Y1 = np.array(Ytrain_set)
print(Y1[0:5])
```

```
[[0.083818   0.03024194 0.28759894 1.        ]
 [0.04481697 0.92540323 0.60686016 1.        ]
 [0.22271639 0.41330645 0.15831135 1.        ]
 [0.36161478 0.28830645 0.27616535 1.        ]
 [0.38693124 0.29637097 0.04485488 1.        ]]
[[ 8.76]
 [11.16]
 [13.56]
 [14.88]
 [14.28]]
```

In [16]:

```
# Calculate optimal beta
beta= (np.linalg.inv(X1.transpose().dot(X1))).dot(X1.transpose()).dot(Y1)
beta
```

Out[16]:

```
array([[15.4453813 ],
       [11.61981962],
       [-1.44370448],
       [ 4.12916012]])
```

In [17]:

```
# Calculate S2
s2 = (1/(len(Xtrain_set)+4))*np.sum(np.power((Y1-X1.dot(beta)),2))
s2
```

Out[17]:

```
3.5948966285346877
```

In [18]:

```
# Calculate standard deviation

#std= math.sqrt(
std=np.sqrt(s2*(np.linalg.inv(X1.transpose().dot(X1))))
std
```

```
C:\Users\ZHANGJ~1\AppData\Local\Temp/ipykernel_16588/3769323905.py:4: RuntimeWarning: invalid value encountered i
n sqrt
  std=np.sqrt(s2*(np.linalg.inv(X1.transpose().dot(X1))))
```

Out[18]:

```
array([[0.5591853 ,        nan, 0.12710501,        nan],
       [       nan, 0.57635428,        nan,        nan],
       [0.12710501,        nan, 0.91114355,        nan],
       [       nan,        nan,        nan, 0.42505163]])
```

In [ ]:

In [19]:

```
# Q3
```

In [24]:
```
B=np.array([1,1,1,1])
```

In [25]:
```
def f(x,b):
    return np.dot(x,b)

def grad(x,y,b):
    n=len(Xtrain_set)
    grd=np.dot(x.T,(f(x,b)-y))
    return grd/n

def loss(x,y,b):
    l = np.dot((f(x,b)-y).T,f(x,b)-y)/2
    return l.mean()
```

In [26]:
```
X_=X1
Y_=Y1.reshape(-1,1)
b_=B.reshape(-1,1)
learn_rate=0.05
tolerance=1e-6
lossNow=loss(X_,Y_,b_)
for i in range(20000):
    b_=b_-learn_rate*grad(X_,Y_,b_)
    lossNew=loss(X_,Y_,b_)
    if i%50==0:
        print("the %d times: loss: %f"%(i,lossNew))
        print(b_)
    if abs(lossNow-lossNew)<tolerance:
        break
    lossNow=lossNew
```

```
the 0 times: loss: 14263.190131
[[1.41700501]
 [1.36170558]
 [1.18922965]
 [1.71843551]]
the 50 times: loss: 816.175411
[[7.3426082 ]
 [6.1078922 ]
 [3.12011445]
 [9.50835813]]
the 100 times: loss: 676.015970
[[8.58449412]
 [6.76207702]
 [2.80298306]
 [8.88256203]]
the 150 times: loss: 574.321114
[[9.5753059 ]
 [7.2839727 ]
 [2.47781541]
 [8.20760294]]
the 200 times: loss: 497.988108
[[10.42084772]
 [ 7.75072831]
 [ 2.18153302]
 [ 7.62272316]]
the 250 times: loss: 440.591650
[[11.14389465]
 [ 8.16833256]
 [ 1.91135893]
 [ 7.11829517]]
the 300 times: loss: 397.355964
[[11.76252609]
 [ 8.54131869]
 [ 1.66432319]
 [ 6.68356084]]
the 350 times: loss: 364.725066
[[12.29206165]
 [ 8.87397077]
 [ 1.43788945]
 [ 6.30913355]]
the 400 times: loss: 340.047454
[[12.74551538]
 [ 9.170303  ]
 [ 1.22988847]
 [ 5.98686516]]
the 450 times: loss: 321.343407
[[13.13395465]
 [ 9.4340369 ]
 [ 1.03845392]
 [ 5.70968687]]
the 500 times: loss: 307.132902
[[13.46680073]
 [ 9.66859311]
 [ 0.86197034]
 [ 5.47146924]]
the 550 times: loss: 296.308101
```

```
[[13.75208113]
 [ 9.87709371]
 [ 0.69903098]
 [ 5.26689981]]
the 600 times: loss: 288.038731
[[13.99664188]
 [10.06237189]
 [ 0.5484036 ]
 [ 5.09137624]]
the 650 times: loss: 281.701752
[[14.20632621]
 [10.22698629]
 [ 0.40900268]
 [ 4.94091309]]
the 700 times: loss: 276.829027
[[14.3861254 ]
 [10.37323828]
 [ 0.27986667]
 [ 4.81206057]]
the 750 times: loss: 273.068317
[[14.54030602]
 [10.50319079]
 [ 0.16013944]
 [ 4.70183377]]
the 800 times: loss: 270.154205
[[14.67251768]
 [10.61868767]
 [ 0.049055  ]
 [ 4.60765108]]
the 850 times: loss: 267.886370
[[14.78588418]
 [10.72137289]
 [-0.05407506]
 [ 4.52728055]]
the 900 times: loss: 266.113363
[[14.8830808 ]
 [10.8127092 ]
 [-0.14987203]
 [ 4.45879333]]
the 950 times: loss: 264.720463
[[14.96639996]
 [10.89399579]
 [-0.23889875]
 [ 4.40052317]]
the 1000 times: loss: 263.620586
[[15.03780691]
 [10.96638487]
 [-0.32166684]
 [ 4.35103129]]
the 1050 times: loss: 262.747474
[[15.09898721]
 [11.03089701]
 [-0.39864278]
 [ 4.30907582]]
the 1100 times: loss: 262.050584
[[15.15138711]
 [11.08843531]
 [-0.47025314]
 [ 4.27358531]]
the 1150 times: loss: 261.491253
[[15.19624796]
 [11.13979819]
 [-0.53688888]
 [ 4.2436358 ]]
the 1200 times: loss: 261.039810
[[15.23463555]
 [11.18569111]
 [-0.59890912]
 [ 4.21843085]]
the 1250 times: loss: 260.673408
[[15.26746526]
 [11.22673706]
 [-0.65664441]
 [ 4.19728428]]
the 1300 times: loss: 260.374391
[[15.2955234 ]
 [11.26348597]
 [-0.71039944]
 [ 4.17960524]]
the 1350 times: loss: 260.129054
[[15.31948558]
 [11.29642326]
 [-0.76045552]
 [ 4.16488521]]
the 1400 times: loss: 259.926718
[[15.3399324 ]
 [11.32597725]
 [-0.80707272]
 [ 4.15268682]]
```

```
the 1450 times: loss: 259.759022
[[15.35736283]
 [11.35252597]
 [-0.85049171]
 [ 4.14263407]]
the 1500 times: loss: 259.619385
[[15.37220578]
 [11.37640306]
 [-0.89093542]
 [ 4.13440401]]
the 1550 times: loss: 259.502602
[[15.38482989]
 [11.39790306]
 [-0.92861055]
 [ 4.12771941]]
the 1600 times: loss: 259.404536
[[15.39555206]
 [11.41728608]
 [-0.96370889]
 [ 4.12234245]]
the 1650 times: loss: 259.321877
[[15.40464468]
 [11.43478189]
 [-0.99640846]
 [ 4.11806936]]
the 1700 times: loss: 259.251968
[[15.41234186]
 [11.45059359]
 [-1.02687465]
 [ 4.11472563]]
the 1750 times: loss: 259.192656
[[15.4188448 ]
 [11.46490079]
 [-1.05526115]
 [ 4.11216202]]
the 1800 times: loss: 259.142195
[[15.42432638]
 [11.47786243]
 [-1.08171083]
 [ 4.11025099]]
the 1850 times: loss: 259.099155
[[15.42893515]
 [11.48961928]
 [-1.10635659]
 [ 4.1088837 ]]
the 1900 times: loss: 259.062362
[[15.43279868]
 [11.50029607]
 [-1.12932203]
 [ 4.10796741]]
the 1950 times: loss: 259.030846
[[15.43602651]
 [11.51000347]
 [-1.15072218]
 [ 4.10742316]]
the 2000 times: loss: 259.003803
[[15.43871269]
 [11.51883971]
 [-1.17066412]
 [ 4.1071839 ]]
the 2050 times: loss: 258.980561
[[15.44093792]
 [11.52689209]
 [-1.18924749]
 [ 4.10719275]]
the 2100 times: loss: 258.960559
[[15.44277139]
 [11.53423828]
 [-1.20656511]
 [ 4.10740155]]
the 2150 times: loss: 258.943324
[[15.44427245]
 [11.54094744]
 [-1.22270336]
 [ 4.10776965]]
the 2200 times: loss: 258.928458
[[15.44549195]
 [11.54708123]
 [-1.23774272]
 [ 4.10826281]]
the 2250 times: loss: 258.915623
[[15.4464734 ]
 [11.55269468]
 [-1.25175814]
 [ 4.10885228]]
the 2300 times: loss: 258.904533
[[15.44725406]
 [11.55783698]
 [-1.26481943]
```

```
 [ 4.10951398]]
the 2350 times: loss: 258.894944
[[15.44786579]
 [11.56255212]
 [-1.27699163]
 [ 4.11022783]]
the 2400 times: loss: 258.886648
[[15.44833579]
 [11.56687953]
 [-1.28833531]
 [ 4.11097718]]
the 2450 times: loss: 258.879467
[[15.44868728]
 [11.57085454]
 [-1.29890693]
 [ 4.11174828]]
the 2500 times: loss: 258.873247
[[15.44894005]
 [11.57450891]
 [-1.30875906]
 [ 4.11252981]]
the 2550 times: loss: 258.867859
[[15.44911097]
 [11.5778712 ]
 [-1.31794072]
 [ 4.11331259]]
the 2600 times: loss: 258.863189
[[15.44921435]
 [11.58096711]
 [-1.32649755]
 [ 4.11408919]]
the 2650 times: loss: 258.859140
[[15.44926238]
 [11.58381981]
 [-1.3344721 ]
 [ 4.11485366]]
the 2700 times: loss: 258.855629
[[15.44926536]
 [11.58645023]
 [-1.341904  ]
 [ 4.11560135]]
the 2750 times: loss: 258.852584
[[15.44923202]
 [11.58887729]
 [-1.3488302 ]
 [ 4.11632863]]
the 2800 times: loss: 258.849942
[[15.44916971]
 [11.5911181 ]
 [-1.35528512]
 [ 4.11703279]]
the 2850 times: loss: 258.847650
[[15.44908463]
 [11.59318818]
 [-1.36130083]
 [ 4.11771184]]
the 2900 times: loss: 258.845661
[[15.44898199]
 [11.5951016 ]
 [-1.36690722]
 [ 4.1183644 ]]
the 2950 times: loss: 258.843935
[[15.44886615]
 [11.59687115]
 [-1.37213215]
 [ 4.11898963]]
the 3000 times: loss: 258.842436
[[15.44874073]
 [11.59850846]
 [-1.37700159]
 [ 4.11958707]]
the 3050 times: loss: 258.841136
[[15.44860874]
 [11.60002412]
 [-1.38153971]
 [ 4.12015662]]
the 3100 times: loss: 258.840007
[[15.44847268]
 [11.6014278 ]
 [-1.38576908]
 [ 4.12069846]]
the 3150 times: loss: 258.839026
[[15.44833458]
 [11.6027283 ]
 [-1.38971068]
 [ 4.12121299]]
the 3200 times: loss: 258.838175
[[15.44819612]
 [11.60393369]
```

```
 [-1.39338412]
 [ 4.12170077]]
the 3250 times: loss: 258.837436
[[15.44805864]
 [11.60505132]
 [-1.39680763]
 [ 4.12216251]]
the 3300 times: loss: 258.836794
[[15.44792321]
 [11.60608794]
 [-1.39999821]
 [ 4.12259901]]
the 3350 times: loss: 258.836237
[[15.44779068]
 [11.60704973]
 [-1.40297172]
 [ 4.12301116]]
the 3400 times: loss: 258.835753
[[15.4476617 ]
 [11.60794237]
 [-1.40574293]
 [ 4.12339989]]
the 3450 times: loss: 258.835333
[[15.44753678]
 [11.60877106]
 [-1.4083256 ]
 [ 4.12376617]]
the 3500 times: loss: 258.834968
[[15.44741627]
 [11.60954058]
 [-1.41073256]
 [ 4.12411099]]
the 3550 times: loss: 258.834651
[[15.44730043]
 [11.61025534]
 [-1.41297576]
 [ 4.12443535]]
the 3600 times: loss: 258.834376
[[15.44718942]
 [11.61091939]
 [-1.41506635]
 [ 4.12474022]]
the 3650 times: loss: 258.834136
[[15.44708332]
 [11.61153646]
 [-1.4170147 ]
 [ 4.12502659]]
the 3700 times: loss: 258.833929
[[15.44698215]
 [11.61210999]
 [-1.4188305 ]
 [ 4.12529541]]
the 3750 times: loss: 258.833749
[[15.44688589]
 [11.61264315]
 [-1.42052277]
 [ 4.12554762]]
the 3800 times: loss: 258.833592
[[15.44679446]
 [11.61313887]
 [-1.4220999 ]
 [ 4.12578412]]
the 3850 times: loss: 258.833456
[[15.44670778]
 [11.61359985]
 [-1.42356973]
 [ 4.12600577]]
the 3900 times: loss: 258.833338
[[15.44662572]
 [11.6140286 ]
 [-1.42493957]
 [ 4.12621344]]
the 3950 times: loss: 258.833235
[[15.44654814]
 [11.61442742]
 [-1.42621621]
 [ 4.12640791]]
the 4000 times: loss: 258.833146
[[15.44647488]
 [11.61479846]
 [-1.427406  ]
 [ 4.12658996]]
the 4050 times: loss: 258.833069
[[15.44640577]
 [11.61514368]
 [-1.42851484]
 [ 4.12676033]]
the 4100 times: loss: 258.833001
[[15.44634065]
```

```
 [11.61546494]
 [-1.42954824]
 [ 4.12691971]]
the 4150 times: loss: 258.832943
[[15.44627935]
 [11.61576391]
 [-1.43051133]
 [ 4.12706878]]
```

In [23]:

```
# comparison
beta
```

Out[23]:

```
array([[15.4453813 ],
       [11.61981962],
       [-1.44370448],
       [ 4.12916012]])
```

In [29]:

```
Xtest_set.head()
```

Out[29]:

	youtube	facebook	newspaper	constant
120	0.480027	0.538776	0.573643	1
117	0.258450	0.008163	0.167959	1
42	1.000000	0.557143	0.000000	1
37	0.252646	1.000000	0.567183	1
69	0.737794	0.887755	0.328165	1

In [30]:

```
# Predict

X2=np.array(Xtest_set)

print(X2[0:5])
```

```
[[0.48002731 0.53877551 0.57364341 1.        ]
 [0.25844998 0.00816327 0.16795866 1.        ]
 [1.         0.55714286 0.         1.        ]
 [0.25264595 1.         0.56718346 1.        ]
 [0.73779447 0.8877551  0.32816537 1.        ]]
```

In [31]:

```
Ytest_set.head()
```

Out[31]:

	sales
120	18.60
117	11.28
42	24.84
37	17.64
69	26.76

In [32]:

```
y_predict=X2.dot(beta)
print(y_predict[:5])
```

```
[[16.97566769]
 [ 7.97339166]
 [26.04844093]
 [18.83234754]
 [25.36645735]]
```

In [33]:

```
# Build predicted value into dataframe
y_predict=pd.DataFrame(y_predict)
y_predict.columns=['predict']
y_predict[:5]
```

Out[33]:

Out[33]:

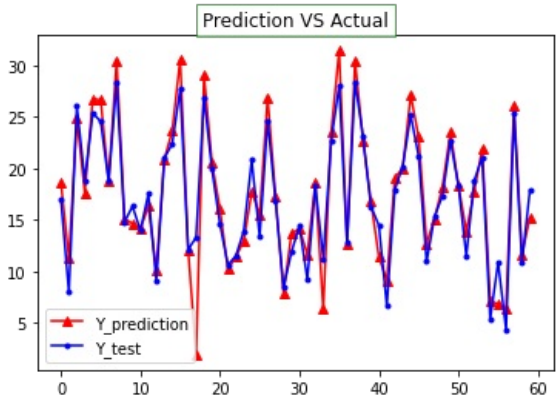
	predict
0	16.975668
1	7.973392
2	26.048441
3	18.832348
4	25.366457

In [34]:

```
# Plot prediction and actual y
x=range(60)
y1=Ytest_set
y2=y_predict
plt.title('Prediction VS Actual',bbox=dict(facecolor='white',edgecolor='g',alpha=0.65))
plt.plot(x, y1, color = 'red',marker="^", label = "Y_prediction")
plt.plot(x, y2, color = 'blue',marker=".", label = "Y_test")
plt.legend()
```

Out[34]:

```
<matplotlib.legend.Legend at 0x1861f33c280>
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js