To demonstrate regularization over non linear model

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
from sklearn.datasets import load_diabetes
data = load_diabetes()
```

In [3]:

```
x = data.data
y = data.target
```

In [4]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state = 45)
```

In [10]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)
```

Out[10]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [12]:

```
lr.coef_
```

Out[12]:

```
array([  23.45388514, -247.43107084,  492.10188174,  329.36498638,
       -970.79784704,  573.54460121,  182.41386124,  255.9162021 ,
        794.21654496,   89.32074078])
```

In [16]:

```
lr.intercept_
```

Out[16]:

```
152.13619339070766
```

In [18]:

```
y_pred = lr.predict(x_test)
y_pred
```

Out[18]:

```
array([226.51666145, 157.46174876,  89.85952394, 207.92164744,
       175.26801312, 146.48388955, 131.11418872,  97.37619407,
       102.94176683, 181.69421036, 237.97133408, 134.74602577,
       189.44001676,  59.93090877, 179.98603311, 117.78640765,
       120.30704482, 126.66365457, 165.19760646, 147.7794185 ,
       145.43611319, 124.41603451,  51.95784538, 227.75026698,
       218.09974354, 129.82788151, 160.13813249, 201.1733737 ,
       184.84256427,  68.91311254, 237.00908624,  58.17038384,
       154.40273359, 119.16002486, 234.03288057, 172.80124283,
       139.94748943, 169.99141456, 214.59266744, 220.47790767,
       128.98001836, 186.20573984, 162.87312596, 179.77626906,
       107.76819766, 249.53008184, 140.92296676,  32.74662537,
       177.96075481, 145.76735049, 291.96466984, 125.71393834,
       107.27437771, 156.10370698, 115.93097942, 160.35431782,
       229.47682139, 173.38591961, 159.88140556, 123.04674096,
        88.5518427 , 122.87575187, 169.12903634, 101.97650772,
       282.75280809, 148.00612928, 164.73111978, 157.31651016,
       232.69943194, 121.12075222,  95.66048311, 186.24657654,
        87.52947663, 160.95896049, 242.3763008 , 149.13877387,
       164.93049903, 209.27728194, 112.08505539, 130.02848231,
        98.40717571,  43.04758443, 104.01721785, 227.29449724,
       144.03768453, 111.74963631, 154.06279876, 174.11068281,
        77.82721919])
```

In [20]:

```
y_test
```

Out[20]:

```
array([155., 168., 115., 233., 190., 202.,  59., 101., 118., 244., 252.,
       148., 232.,  72., 107.,  71., 191.,  65., 245.,  85., 185.,  84.,
        78., 268., 248., 178., 196., 248., 144.,  83., 275.,  39., 113.,
        64., 232., 200., 200., 122., 163., 180., 135., 164., 156., 126.,
        68., 306.,  83.,  45.,  91.,  25., 270., 178., 108.,  86., 160.,
       196., 248., 139., 155., 150.,  74.,  89., 216.,  65., 242., 136.,
       185.,  91., 246.,  68., 101., 164., 113., 131., 215., 246., 265.,
       220., 107., 131.,  94., 116.,  63., 217., 302.,  72., 252., 111.,
        72.])
```

In [23]:

```
from sklearn.linear_model import Ridge
R = Ridge(alpha=100000)
R.fit(x_train, y_train)
```

Out[23]:

```
Ridge(alpha=100000, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

In [25]:

```
print("Coefficient of Ridge: ", R.coef_)
print("Intercept of Ridge: ", R.intercept_)
```

```
coff of Ridge:  [ 0.00260126  0.00057066  0.00776597  0.00609765  0.00233864
0.00184724
 -0.00513942  0.0052716   0.00734601  0.00528629]
coff of Ridge:  151.8328793076644
```

In [30]:

```
y_predR = R.predict(x_test)
```

In [31]:

```
print(y_predR  - y_pred)
```

```
[ -74.6818887    -5.62787954   61.97155863  -56.08765957  -23.4339877
    5.34891326   20.71824092   54.45502727   48.88970748  -29.86010153
  -86.13580099   17.08595668  -37.60596138   91.90010797  -28.15244982
   34.04569847   31.52512199   25.16936508  -13.36425918    4.05324125
    6.39537327   27.41705189   99.87323567  -75.91459581  -66.26417518
   22.00407842   -8.30377629  -49.33951463  -33.00905108   82.91781443
  -85.17433695   93.66025976   -2.56999696   32.67193025  -82.19856831
  -20.96805628   11.88513056  -18.15743122  -62.75812242  -68.64273736
   22.85220122  -34.37181464  -11.04031127  -27.94292333   44.06350787
  -97.69450421   10.90963435  119.08399228  -26.12703817    6.064626
 -140.12940933   26.11786413   44.55790023   -4.27151614   35.90074064
   -8.52075328  -77.64132667  -21.55110446   -8.04891202   28.78436197
   63.27896992   28.95783321  -17.29628574   49.85425016 -130.91629025
    3.82560634  -12.89799011   -5.48267318  -80.86524013   30.71086246
   56.17013555  -34.41264821   64.30198569   -9.12546363  -90.54169537
    2.69375077  -13.09720954  -57.44369182   39.74785582   21.80325606
   53.4245393   108.7832184    47.81449709  -75.45844508    7.79464336
   40.08211666   -2.23009212  -22.27712376   74.00382595]
```

In [38]:

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

print("R2 Score of L2 Regulaization", r2_score(y_test, y_predR))
print("Mean Squared Error of L2 Regulaization", mean_squared_error(y_test, y_predR))
```

```
R2 Score of L2 Regulaization -0.0004249020401270176
Mean Squared Error of L2 Regulaization 4936.406155071465
```

In [41]:

```
m = 100
x1 = 5 * np.random.rand(m, 1) - 2
x2 = 0.7 * x1 ** 2 - 2 * x1 + 3 + np.random.randn(m, 1)
```

In [45]:

```
print(x1)
```

```
 [-0.90694096]
 [-0.68058664]
 [ 1.97376395]
 [ 1.57072782]
 [ 1.63339533]
 [ 2.28567609]
 [ 0.23558499]
 [ 0.39756983]
 [ 2.82011102]
 [-0.75668011]
 [ 1.05595367]
 [-1.39484781]
 [ 0.160893  ]
 [ 1.08804494]
 [ 0.52627986]
 [ 0.61952629]
 [ 0.11556974]
 [ 2.48432005]
 [ 1.86383231]
 [-1.02455739]]
```

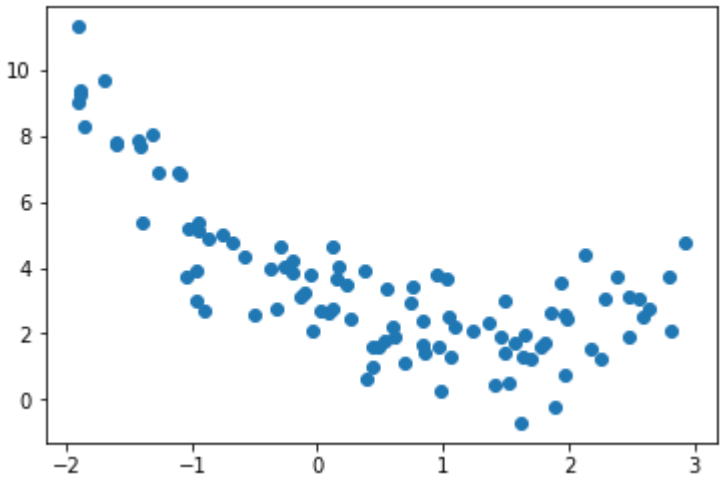
In [44]:

```
print(x2)
```

```
[[ 3.68527316]
 [ 0.51035973]
 [ 3.12673549]
 [ 7.853898  ]
 [ 2.33804204]
 [ 2.47949417]
 [ 1.24341551]
 [ 0.4371526 ]
 [ 4.21355248]
 [ 1.12066893]
 [ 6.81692424]
 [ 2.48462735]
 [ 5.37382163]
 [ 3.04221957]
 [ 3.73608085]
 [ 4.35685894]
 [ 7.71415846]
 [ 1.2132833 ]
 [ 6.87752679]
```

In [49]:

```
plt.scatter(x1, x2);
```



In [64]:

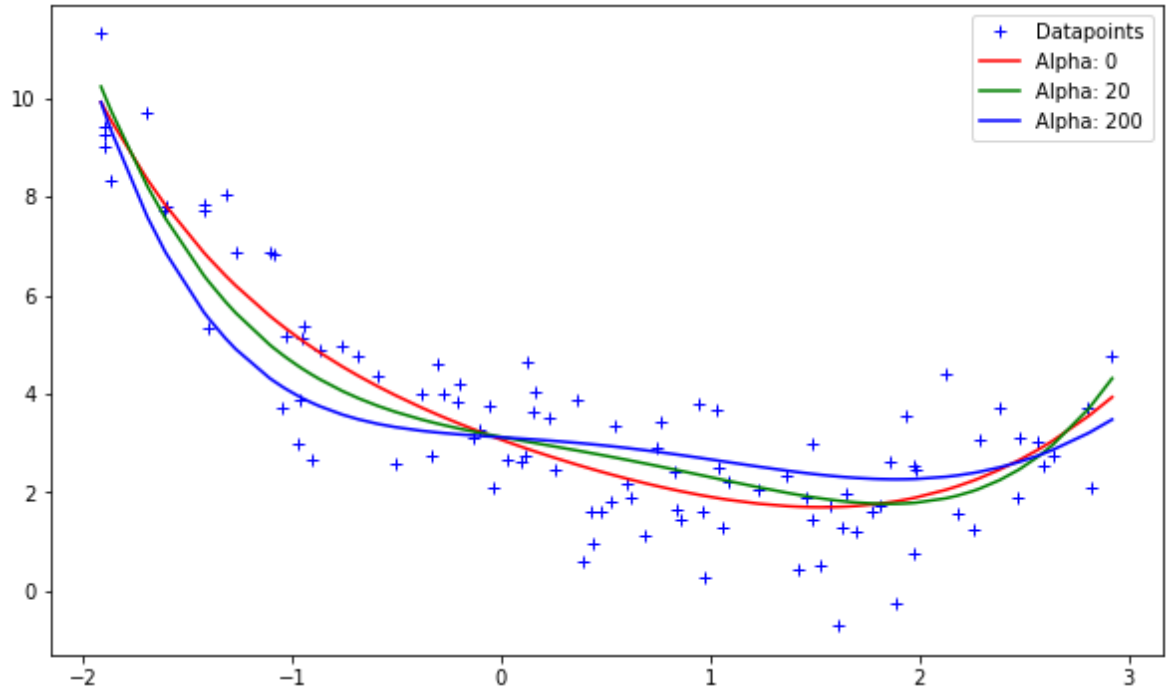
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

def get_preds_ridge(x1, x2, alpha):
    model = Pipeline([
        ('poly_features', PolynomialFeatures(degree = 5)),
        ('ridge', Ridge(alpha=alpha))
    ])
    model.fit(x1, x2)
    return model.predict(x1)
```

In [73]:

```
# lambda ki value jaise jaise increase karenge waise waise slope(m) kam hoga
alphas = [0, 20, 200]
cs = ['r', 'g', 'b']

plt.figure(figsize=(10,6))
plt.plot(x1, x2, 'b+', label='Datapoints')

for alpha, c in zip(alphas, cs):
    preds = get_preds_ridge(x1, x2, alpha)
    #plot
    plt.plot(sorted(x1[:,0]), preds[np.argsort(x1[:,0])],c, label = 'Alpha: {}'.format(alph

plt.legend()
```

Out[73]:

```
<matplotlib.legend.Legend at 0x201d1aedba8>
```



Lasso Regularization

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:

```
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge

from sklearn.datasets import make_regression  # make_regression= for regression #make_moon
from sklearn.model_selection import train_test_split
```

In [5]:

```
x,y=make_regression(n_samples=100,n_features=1,n_targets=1,noise=20,random_state=13) #sampl
```

In [6]:

```
x
```

Out[6]:

```
array([[-0.71239066],
       [-0.43714566],
       [-0.45375238],
       [ 0.95283061],
       [ 0.23785784],
       [ 0.86121137],
       [-0.24332625],
       [ 0.18494595],
       [-0.72099967],
       [-0.42989708],
       [ 2.01522083],
       [ 0.39724133],
       [ 0.20780005],
       [-0.23242587],
       [-0.76862702],
       [ 0.45315861],
       [ 0.63988397],
       [ 0.3595323 ],
       [-1.61510796],
       [ 1.74924179],
       [-0.78898902],
       [-0.51196509],
       [-0.92833523],
       [ 2.15038297],
       [-0.2073497 ],
       [-1.63909341],
       [-0.33861825],
       [-0.32212366],
       [-0.48137142],
       [-0.52316421],
       [ 0.72196506],
       [ 0.76591105],
       [ 0.45348104],
       [-1.26160595],
       [-2.18711527],
       [-1.18541881],
       [ 0.21745166],
       [ 1.33031692],
       [-1.08718159],
       [ 0.56226171],
       [-1.51284512],
       [-0.00238903],
       [-0.27813452],
       [ 0.45181234],
       [ 1.19070527],
       [ 0.92234415],
       [ 0.81499544],
       [-0.6209797 ],
       [ 0.9137407 ],
       [ 1.13833305],
       [ 1.47868574],
       [-0.65105648],
       [-0.37591996],
       [-0.77466003],
       [ 0.50113729],
```

```
       [ 1.3501879 ],
       [ 0.72916547],
       [-0.08165156],
       [-0.85414295],
       [ 0.46565797],
       [-0.04450308],
       [-0.05753239],
       [ 1.89274222],
       [-1.04537713],
       [ 0.56465429],
       [-1.92415945],
       [-0.76403397],
       [ 0.12730328],
       [-0.02677165],
       [-0.14521133],
       [ 0.56284679],
       [ 0.31735092],
       [ 0.71097479],
       [ 0.75376638],
       [-0.37011608],
       [ 1.34510171],
       [ 0.53233789],
       [-0.98416078],
       [ 1.350306  ],
       [-0.34660679],
       [ 0.51432886],
       [ 0.10126979],
       [-0.65751727],
       [ 0.83090566],
       [-0.31726597],
       [-0.98027432],
       [ 1.39923842],
       [ 0.54791831],
       [-0.53032741],
       [ 0.49087183],
       [ 0.34875059],
       [ 2.05369324],
       [ 0.60628866],
       [-0.38445769],
       [-1.94539068],
       [-0.31485808],
       [ 1.84961257],
       [-1.12050687],
       [-0.33267578],
       [-0.75745323]])
```

In [7]:

```
y
```

Out[7]:

```
array([-3.43198806e+01, -9.42120961e+00, -1.90881877e+01,  2.04372122e+01,
        2.77559659e+01, -2.90750046e+00, -1.41987828e+01,  5.40025891e+00,
       -2.64264302e+01, -3.49067872e+01,  3.73362043e+01,  1.28532816e+01,
        2.50289888e+01, -1.89608736e+01, -2.34655852e+01,  3.77839324e+01,
        6.69670792e+00, -5.57201352e+00, -4.92158778e+01,  1.59474399e+01,
       -4.29667324e+01,  6.09015466e+00, -2.53194769e+01,  6.28216706e+01,
        1.24870400e+01, -3.27136530e+01, -1.88255476e+01, -2.93912926e+01,
       -2.86886731e+01,  4.38924069e+00,  4.63542396e+01,  2.43919519e+01,
        3.79848517e+01, -3.45767718e+01, -6.18736296e+01, -4.64421597e+01,
       -6.88808416e+00,  3.96988084e+01, -3.52373298e+01,  8.36850884e+00,
       -3.96814412e+01,  8.27318308e+00, -4.40722161e+00, -3.01350607e+00,
        5.78213629e+01,  2.46525603e+01,  1.81131707e+01, -5.22849035e+01,
        3.59187182e+01,  1.58411788e+01,  2.40080546e+01, -2.51245994e+01,
       -4.39284313e+01,  1.73925049e+01,  1.50322799e+01,  3.78339073e+01,
        6.33015059e+00,  5.07266140e+00, -5.57047189e+00,  1.98411137e+01,
        7.04149700e+00, -1.89353310e+01,  3.48047372e+01, -5.71391242e+01,
        3.10524837e+01, -7.57291461e+01, -4.43062883e+01,  2.12404103e+01,
       -1.86288622e-01, -2.85657165e+01, -7.39169323e+00,  2.52239775e+01,
        3.25432136e+01,  2.93379129e-01, -1.86381423e+01,  3.93501682e+01,
        2.54005098e+00, -4.86320699e+01,  5.42818051e+01, -1.02170910e+01,
       -5.66921620e+00,  2.34628620e+01, -6.92299670e-02,  5.63079459e+00,
       -9.68771368e+00, -4.46638656e+01,  3.27083097e+01,  1.81174790e+01,
       -1.61432262e+01,  4.50414413e+01,  2.27466324e+01,  3.71085471e+01,
        1.24760264e+01, -3.58452002e+01, -2.86386661e+01, -2.79790662e+01,
        6.41794705e+01, -6.07154394e+01,  2.22333646e+01, -2.40222151e+00])
```

In [8]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=13)
```

In [13]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

In [14]:

```
lr.fit(x_train, y_train)
```

Out[14]:

```
LinearRegression()
```

In [15]:

```
lr.coef_
```

Out[15]:

```
array([28.67684216])
```

In [16]:

```
lr.intercept_
```

Out[16]:

```
-2.088699766134101
```

In [22]:

```
y_pred = lr.predict(x_test)
y_pred
```

Out[22]:

```
array([ 36.06058868,  52.18917009,  21.73905063, -24.13049552,
        10.86785136, -14.41679053,  32.05696739,  19.8752105 ,
        -6.25290223,  -4.43020855, -38.26757433,  13.62386712,
         3.87034956,  56.80473702,  -2.15720948,  59.57749317,
        13.17706983, -17.09139738,  40.31533773,  -2.85642615])
```

In [17]:

```
#ridge regression
R = Ridge(alpha=5)
R.fit(x_train, y_train)
```

Out[17]:

```
Ridge(alpha=5)
```

In [27]:

```
R_pred = R.predict(x_test)
R_pred
```

Out[27]:

```
array([ 33.19857246,  48.16471156,  19.90923689, -22.65433338,
         9.82156232, -13.6407285 ,  29.48350592,  18.17973013,
        -6.0652397 ,  -4.37391382, -35.77250432,  12.37894255,
         3.32839454,  52.44761871,  -2.2647376 ,  55.0205328 ,
        11.96434748, -16.12256724,  37.14666712,  -2.91355932])
```

In [28]:

```
print("Coefficient of Ridge: ", R.coef_)
print("Intercept of Ridge: ", R.intercept_)
```

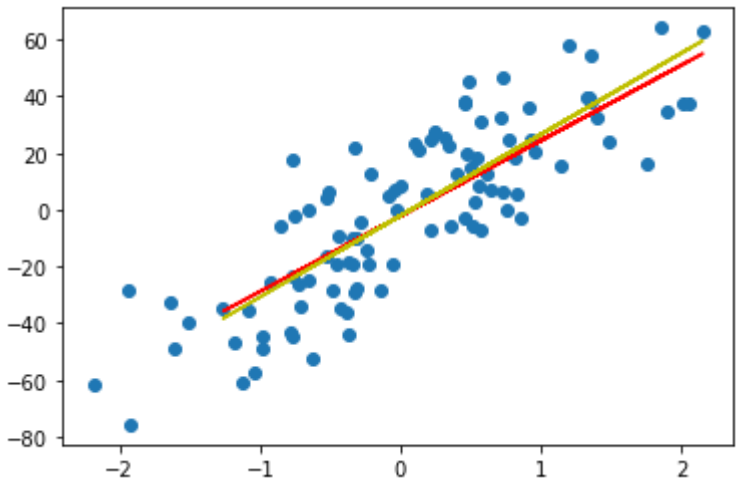
```
Coefficient of Ridge:  [26.61000356]
Intercept of Ridge:  -2.201165618510922
```

In [29]:

```
plt.scatter(x,y)
plt.plot(x_test,R.predict(x_test),label='Rid',c='r')
plt.plot(x_test,lr.predict(x_test),label='LR',c='y')
```

Out[29]:

```
[<matplotlib.lines.Line2D at 0x1bc590bbee0>]
```



In [32]:

```
L= Lasso(alpha=10)
L.fit(x_train, y_train)
```

Out[32]:

```
Lasso(alpha=10)
```

In [33]:

```
L_pred = L.predict(x_test)
L_pred
```

Out[33]:

```
array([ 18.85192524,  27.99100103,  10.73678986, -15.2546627 ,
         4.5767494 ,  -9.75050311,  16.58331913,   9.68066622,
        -5.12452938,  -4.0917209 , -23.26527637,   6.13841414,
         0.61169522,  30.60635912,  -2.80375203,  32.17750962,
         5.88524158, -11.26603842,  21.26283002,  -3.19995514])
```

In [34]:

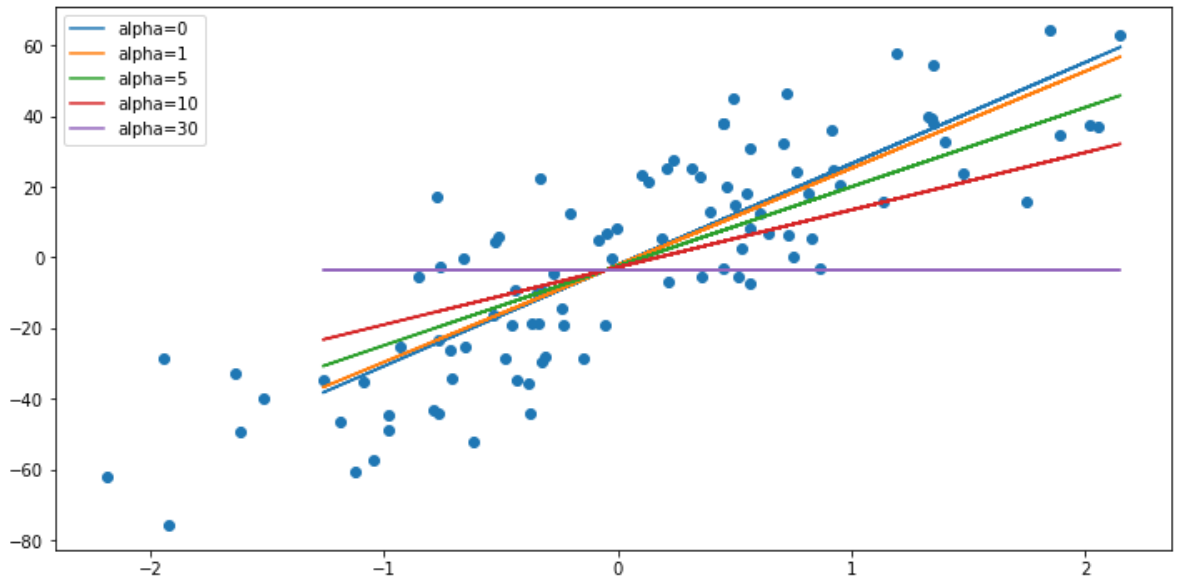
```
print("Coefficient of Lasso: ", L.coef_)
print("Intercept of Lasso: ", L.intercept_)
```

```
Coefficient of Lasso:  [16.24940391]
Intercept of Lasso:  -2.7649317831463702
```

In [35]:

```
alphas=[0,1,5,10,30]
plt.figure(figsize=(12,6))
plt.scatter(x,y)
for i in alphas:
    L=Lasso(alpha=i)
    L.fit(x_train,y_train)
    plt.plot(x_test,L.predict(x_test),label='alpha={}'.format(i))
plt.legend()
plt.show(
)
```

```
C:\Users\MSCIT\AppData\Local\Temp/ipykernel_12024/2188626010.py:6: UserWarni
ng: With alpha=0, this algorithm does not converge well. You are advised to
use the LinearRegression estimator
  L.fit(x_train,y_train)
C:\Users\MSCIT\anaconda3.1\lib\site-packages\sklearn\linear_model\_coordinat
e_descent.py:530: UserWarning: Coordinate descent with no regularization may
lead to unexpected results and is discouraged.
  model = cd_fast.enet_coordinate_descent(
C:\Users\MSCIT\anaconda3.1\lib\site-packages\sklearn\linear_model\_coordinat
e_descent.py:530: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations. Duality gap: 12057.35658838604, t
olerance: 7.705313925235513
  model = cd_fast.enet_coordinate_descent(
```



In [37]:

```
from sklearn.datasets import load_diabetes
```

In [38]:

```
data = load_diabetes()
```

In [44]:

```
x1=data.data
y1=data.target
```

In [45]:

```
from sklearn.model_selection import train_test_split
x1_train,x1_test,y1_train,y1_test=train_test_split(x1,y1,test_size=0.2,random_state=2)
```

In [49]:

```
# we are creating 2 empty list
from sklearn.metrics import r2_score
coefs=[]
r2_scores=[]

for i in [0,0.1,1,10]:
    reg=Lasso(alpha=i)
    reg.fit(x1_train,y1_train)

    coefs.append(reg.coef_.tolist())   # coeffiecient will convert  into list
    y_pred=reg.predict(x1_test)
    r2_scores.append(r2_score(y1_test,y_pred))
```

```
C:\Users\MSCIT\AppData\Local\Temp/ipykernel_12024/917754590.py:8: UserWarnin
g: With alpha=0, this algorithm does not converge well. You are advised to u
se the LinearRegression estimator
  reg.fit(x1_train,y1_train)
C:\Users\MSCIT\anaconda3.1\lib\site-packages\sklearn\linear_model\_coordinat
e_descent.py:530: UserWarning: Coordinate descent with no regularization may
lead to unexpected results and is discouraged.
  model = cd_fast.enet_coordinate_descent(
C:\Users\MSCIT\anaconda3.1\lib\site-packages\sklearn\linear_model\_coordinat
e_descent.py:530: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations. Duality gap: 496708.2205472759, t
olerance: 212.4353654390935
  model = cd_fast.enet_coordinate_descent(
```

In [50]:

```
r2_scores
```

Out[50]:

```
[0.4399387661037827,
 0.4334654091230754,
 0.32568169677801695,
 -0.012517603619692785]
```

In [54]:

```
coefs
```

Out[54]:

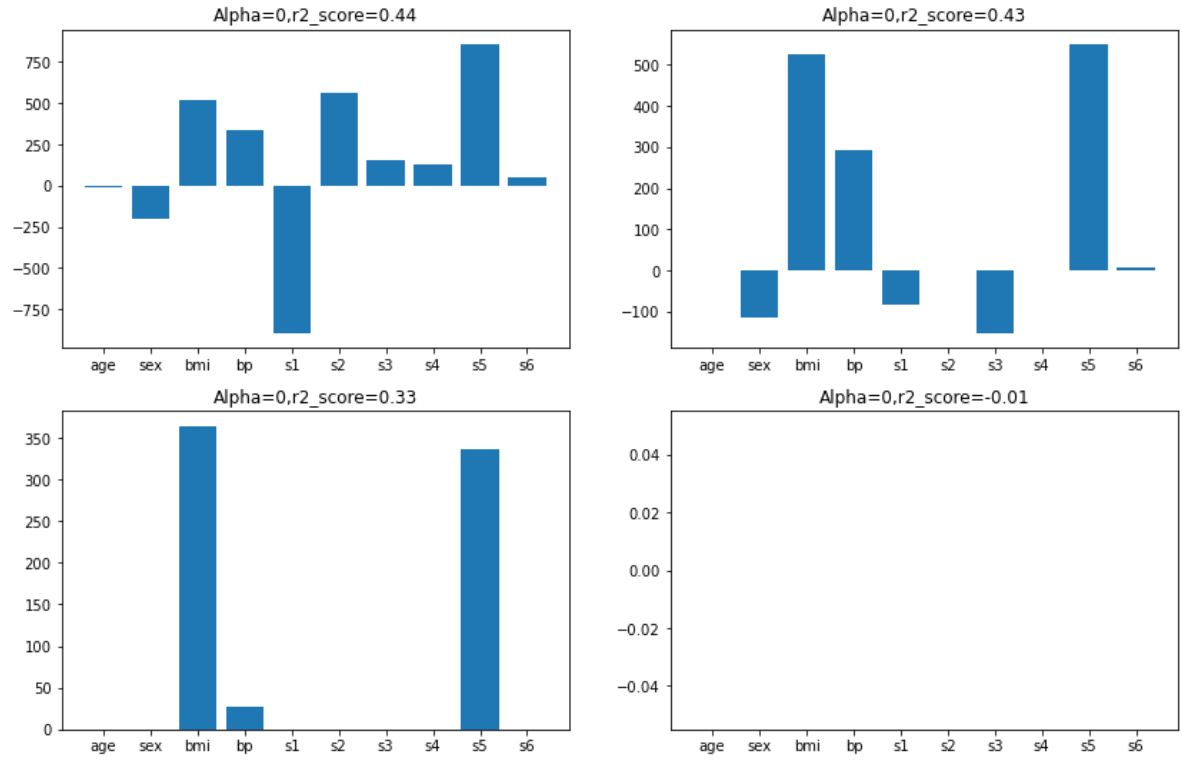
```
[[-9.160884830102175,
  -205.46225976623592,
  516.6846240655403,
  340.62734101990014,
  -895.5435958123823,
  561.2145229912762,
  153.88478023466485,
  126.73431430271964,
  861.1213947849758,
  52.41982835037267],
 [0.0,
  -113.97604628790536,
  526.7371120851084,
  292.6354225287151,
  -82.69192843359384,
  -0.0,
  -152.6913315736965,
  0.0,
  551.0771996122245,
  7.16985202642276],
 [0.0,
  0.0,
  363.88263603837737,
  27.27842001502803,
  0.0,
  0.0,
  -0.0,
  0.0,
  336.13597084247954,
  0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, 0.0, 0.0, 0.0]]
```

In [57]:

```
#feature slection
# bar graph and check kitne feature jaarhe h
plt.figure(figsize=(14,9))
plt.subplot(221)
plt.bar(data.feature_names,coefs[0])
plt.title('Alpha=0,r2_score={}'.format(round(r2_scores[0],2)))

plt.subplot(222)
plt.bar(data.feature_names,coefs[1])
plt.title('Alpha=0,r2_score={}'.format(round(r2_scores[1],2)))

plt.subplot(223)
plt.bar(data.feature_names,coefs[2])
plt.title('Alpha=0,r2_score={}'.format(round(r2_scores[2],2)))

plt.subplot(224)
plt.bar(data.feature_names,coefs[3])
plt.title('Alpha=0,r2_score={}'.format(round(r2_scores[3],2)))
```

Out[57]:

```
Text(0.5, 1.0, 'Alpha=0,r2_score=-0.01')
```



In [59]:

```
#shrinking method
alphas=[0,0.001,0.001,0.01,0.1,1,10,100,1000,10000]

coefs=[]

for i in alphas:
    reg=Lasso(alpha=i)
    reg.fit(x1_train,y1_train)

    coefs.append(reg.coef_.tolist())
```

```
C:\Users\MSCIT\AppData\Local\Temp/ipykernel_12024/2515527026.py:7: UserWarni
ng: With alpha=0, this algorithm does not converge well. You are advised to
use the LinearRegression estimator
  reg.fit(x1_train,y1_train)
C:\Users\MSCIT\anaconda3.1\lib\site-packages\sklearn\linear_model\_coordinat
e_descent.py:530: UserWarning: Coordinate descent with no regularization may
lead to unexpected results and is discouraged.
  model = cd_fast.enet_coordinate_descent(
C:\Users\MSCIT\anaconda3.1\lib\site-packages\sklearn\linear_model\_coordinat
e_descent.py:530: ConvergenceWarning: Objective did not converge. You might
want to increase the number of iterations. Duality gap: 496708.2205472759, t
olerance: 212.4353654390935
  model = cd_fast.enet_coordinate_descent(
```

In [60]:

```
input_array=np.array(coefs)

coef_df=pd.DataFrame(input_array,columns=data.feature_names)

coef_df['alpha']=alphas

coef_df.set_index('alpha')
```

Out[60]:

alpha	age	sex	bmi	bp	s1	s2	s3
0.000	-9.160885	-205.462260	516.684624	340.627341	-895.543596	561.214523	153.884780
0.001	-8.264924	-204.213177	517.641106	339.751339	-826.653342	508.609613	120.899583
0.001	-8.264924	-204.213177	517.641106	339.751339	-826.653342	508.609613	120.899583
0.010	-1.361404	-192.944226	526.348511	332.649058	-430.205495	191.277876	-44.048113
0.100	0.000000	-113.976046	526.737112	292.635423	-82.691928	-0.000000	-152.691332
1.000	0.000000	0.000000	363.882636	27.278420	0.000000	0.000000	-0.000000
10.000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000
100.000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000
1000.000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000
10000.000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.000000

ElasticNet Regularization

In [63]:

```
from sklearn.linear_model import ElasticNet
#ElasticNet
reg=ElasticNet(alpha=0.005,l1_ratio=0.9)
reg.fit(x_train,y_train)
y_pred=reg.predict(x_test)
r2_score(y_test,y_pred)
```

Out[63]:

```
0.7136152589126589
```

In [ ]:

```

```