Polynomial Regression

- Steps :
  - Read the dataset
  - Data preprocessing (null values,duplicate values,shape,info)
  - Seperating features(x) and target(y)
  - select the model
  - spliting the data for training and testing
  - fit the data
  - predict the data and calculating the score

In [36]:
```
experience1 = [0,1,2,3,4,5,6,7,8]
salary1   = [5000,6000,7000,8000,15000,25000,40000,50000,80000]
```

In [37]:
```
import pandas as pd
```

In [38]:
```
df = pd.DataFrame({"experience":experience1,"salary":salary1})
```

In [39]:
```
df
```

Out[39]:

	experience	salary
0	0	5000
1	1	6000
2	2	7000
3	3	8000
4	4	15000
5	5	25000
6	6	40000
7	7	50000
8	8	80000

In [40]:
```
df.shape
```

Out[40]: (9, 2)

In [41]:
```
df.isnull().sum()
```

Out[41]:
```
experience    0
salary        0
dtype: int64
```

In [42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 2 columns):
experience    9 non-null int64
salary        9 non-null int64
dtypes: int64(2)
memory usage: 224.0 bytes
```

In [43]:
```
df.describe()
```

Out[43]:

	experience	salary
count	9.000000	9.000000
mean	4.000000	26222.222222
std	2.738613	25825.267558
min	0.000000	5000.000000
25%	2.000000	7000.000000
50%	4.000000	15000.000000
75%	6.000000	40000.000000
max	8.000000	80000.000000

In [44]:
```
# seperating features and target

x = df[["experience"]]
y = df["salary"]
```

In [45]:
```
x.ndim
```

Out[45]: 2

In [46]:
```
y.ndim
```

Out[46]: 1

In [47]:
```
# select the model

from sklearn.linear_model import LinearRegression
```

In [48]:
```
model = LinearRegression()
```

In [49]:
```
model.fit(x,y)
```

Out[49]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [50]:
```
df.shape
```
Out[50]: (9, 2)

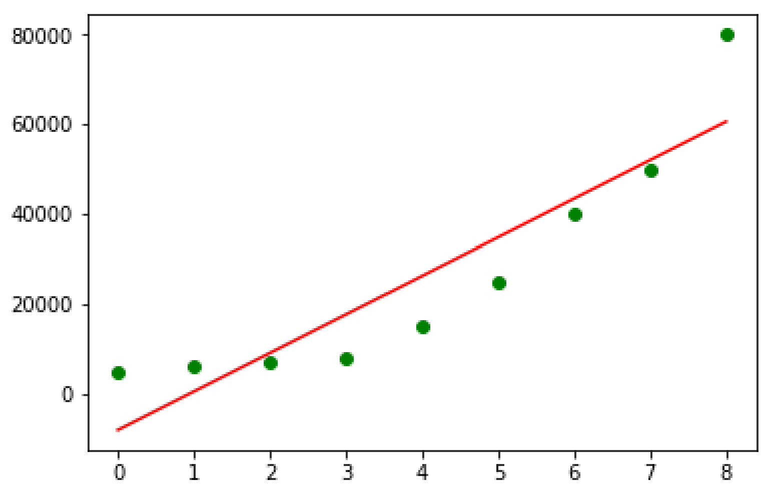
In [51]:
```
pred = model.predict(x)
```

In [52]:
```
pred
```
Out[52]:
```
array([-8111.11111111,   472.22222222,  9055.55555556, 17638.88888889,
        26222.22222222, 34805.55555556, 43388.88888889, 51972.22222222,
        60555.55555556])
```

In [53]:
```
model.score(x,y)*100
```
Out[53]: 82.84829237817574

In [54]:
```
import matplotlib.pyplot as plt
```

In [56]:
```
plt.scatter(df["experience"],df["salary"],c = "green")
plt.plot(df["experience"],pred,c="red")
```
Out[56]: [<matplotlib.lines.Line2D at 0x25c17b809e8>]



In [57]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [63]:
```
poly = PolynomialFeatures(degree=2)
poly
```
Out[63]: PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)

In [60]:
```
x_poly = poly.fit_transform(x)
```

In [61]:
```
x_poly
```

Out[61]:
```
array([[ 1.,  0.,  0.],
       [ 1.,  1.,  1.],
       [ 1.,  2.,  4.],
       [ 1.,  3.,  9.],
       [ 1.,  4., 16.],
       [ 1.,  5., 25.],
       [ 1.,  6., 36.],
       [ 1.,  7., 49.],
       [ 1.,  8., 64.]])
```

In [62]:
```
x
```

Out[62]:

	experience
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

In [64]:
```
from sklearn.linear_model import LinearRegression
```

In [65]:
```
model = LinearRegression()
```

In [66]:
```
model.fit(x_poly,y)
```

Out[66]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [70]:
```
pred1 = model.predict(x_poly)
pred1
```

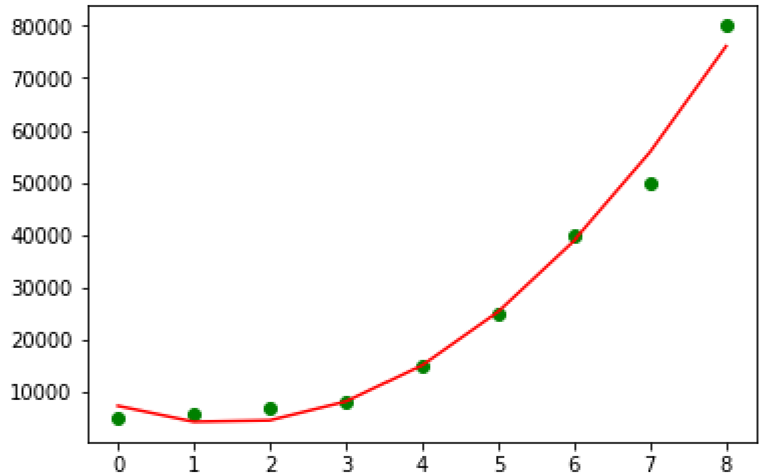
Out[70]:
```
array([ 7393.93939394,  4348.48484848,  4625.54112554,  8225.10822511,
       15147.18614719, 25391.77489177, 38958.87445887, 55848.48484848,
       76060.60606061])
```

In [68]:
```
model.score(x_poly,y)*100
```

Out[68]:
```
98.77932355025233
```

In [71]:
```
plt.scatter(df["experience"],df["salary"],c = "green")
plt.plot(df["experience"],pred1,c="red")
```

Out[71]: [<matplotlib.lines.Line2D at 0x25c17f248d0>]



In [ ]:
```
# reading the dataset
```

In [72]:
```
data = pd.read_csv("https://raw.githubusercontent.com/AP-State-Skill-Develop
```

In [73]:
```
data.head()
```

Out[73]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FU
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	

In [74]:
```
data.columns
```

Out[74]:
```
Index(['MODELYEAR', 'MAKE', 'MODEL', 'VEHICLECLASS', 'ENGINESIZE', 'CYLINDERS',
       'TRANSMISSION', 'FUELTYPE', 'FUELCONSUMPTION_CITY',
       'FUELCONSUMPTION_HWY', 'FUELCONSUMPTION_COMB',
       'FUELCONSUMPTION_COMB_MPG', 'CO2EMISSIONS'],
      dtype='object')
```

In [76]:
```
data.shape
```

Out[76]: (1067, 13)

In [77]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1067 entries, 0 to 1066
Data columns (total 13 columns):
MODELYEAR                   1067 non-null int64
MAKE                        1067 non-null object
MODEL                       1067 non-null object
VEHICLECLASS                1067 non-null object
ENGINESIZE                  1067 non-null float64
CYLINDERS                   1067 non-null int64
TRANSMISSION                1067 non-null object
FUELTYPE                    1067 non-null object
FUELCONSUMPTION_CITY        1067 non-null float64
FUELCONSUMPTION_HWY         1067 non-null float64
FUELCONSUMPTION_COMB        1067 non-null float64
FUELCONSUMPTION_COMB_MPG    1067 non-null int64
CO2EMISSIONS                1067 non-null int64
dtypes: float64(4), int64(4), object(5)
memory usage: 108.4+ KB
```

In [78]:
```
data.isnull().sum()
```

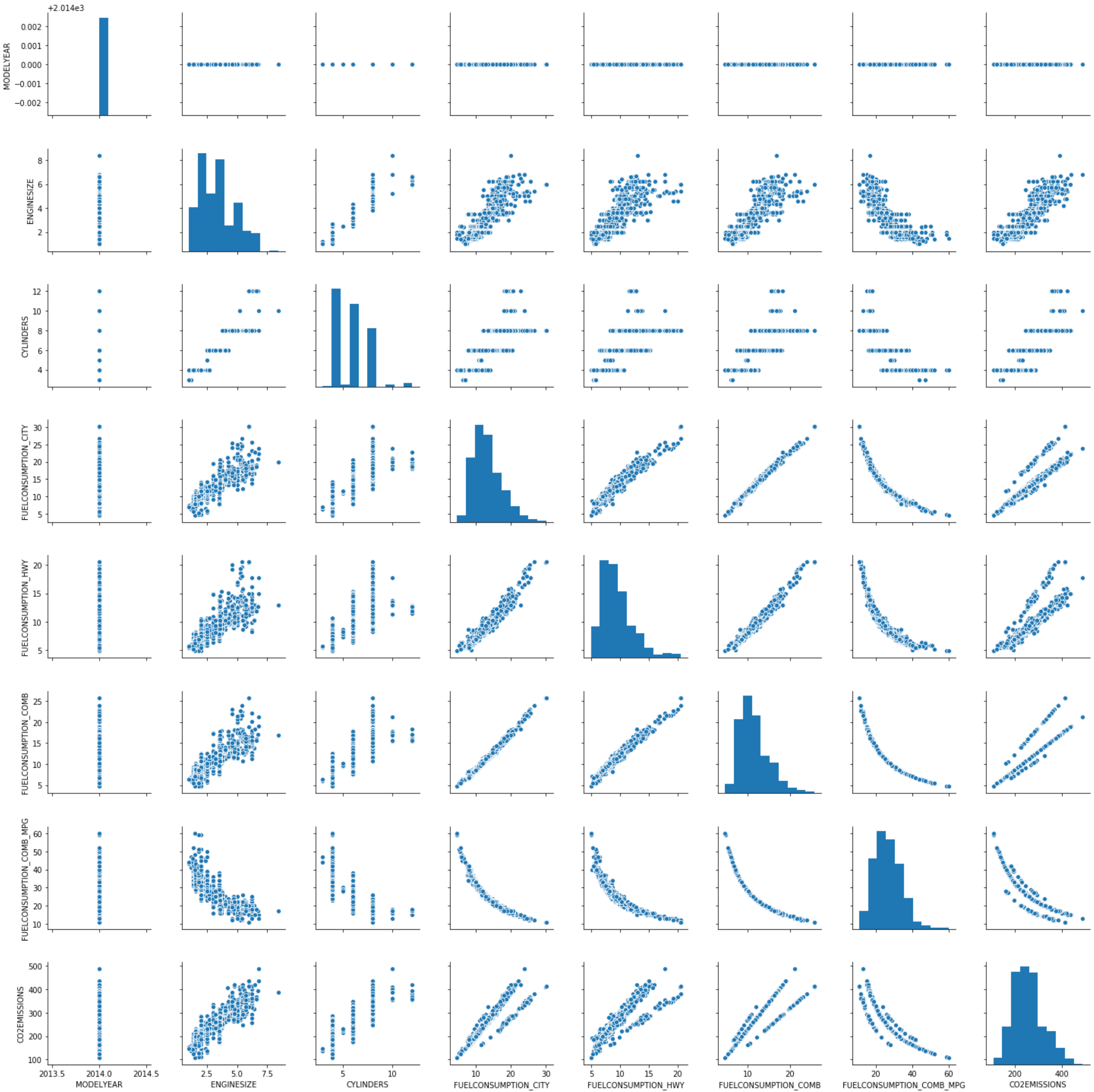
Out[78]:
```
MODELYEAR                   0
MAKE                        0
MODEL                       0
VEHICLECLASS                0
ENGINESIZE                  0
CYLINDERS                   0
TRANSMISSION                0
FUELTYPE                    0
FUELCONSUMPTION_CITY        0
FUELCONSUMPTION_HWY         0
FUELCONSUMPTION_COMB        0
FUELCONSUMPTION_COMB_MPG    0
CO2EMISSIONS                0
dtype: int64
```

In [79]:
```
data.isnull().sum().sum()
```

Out[79]: 0

In [80]:
```
import seaborn as sns
sns.pairplot(data)
```

Out[80]: <seaborn.axisgrid.PairGrid at 0x25c17f21c88>



In [82]:
```
# seperate the features and target

x = data[["FUELCONSUMPTION_COMB_MPG"]]
y = data["CO2EMISSIONS"]
```

In [84]:
```
# apply for model
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures()
xpoly = poly.fit_transform(x)
```

In [85]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(xpoly,y)
```

Out[85]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [88]:
```
pre = model.predict(xpoly)
```

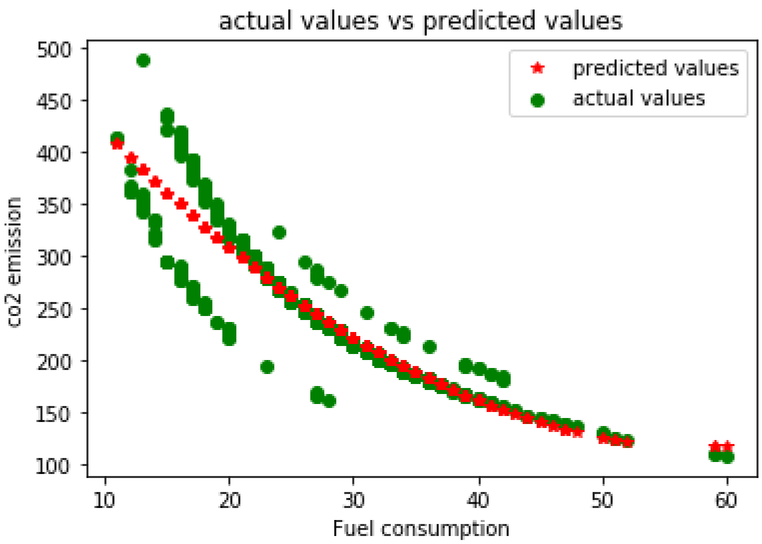
In [89]:
```
pre
```

Out[89]:
```
array([200.47884409, 228.8001283 , 130.87884148, ..., 269.98144092,
       261.23142663, 288.25209714])
```

In [91]:
```
model.score(xpoly,y)*100
```

Out[91]:
```
85.21512602222471
```

In [96]:
```
plt.scatter(x,y,c = "g",label = "actual values")
plt.plot(x,pre,"r*",label = "predicted values")
plt.xlabel("Fuel consumption")
plt.ylabel("co2 emission")
plt.title("actual values vs predicted values")
plt.legend()
plt.show()
```



- overfitting
  - Lasso regularization/L1 regularization
  - Ridge regularization/L2 regularization
  - Elastic Net

Ridge regularization

In [97]:
```
from sklearn.datasets import load_boston
```

In [98]:
```
boston = load_boston()
```

In [99]:
```
boston
```

Out[99]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e
+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9,
15. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
```

```
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n-------------
-------------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances:
506 \n\n    :Number of Attributes: 13 numeric/categorical predictive. Median
Value (attribute 14) is usually the target.\n\n    :Attribute Information (in
order):\n        - CRIM     per capita crime rate by town\n        - ZN
proportion of residential land zoned for lots over 25,000 sq.ft.\n        - I
NDUS    proportion of non-retail business acres per town\n        - CHAS
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n
- NOX      nitric oxides concentration (parts per 10 million)\n        - RM
average number of rooms per dwelling\n        - AGE      proportion of owner-
occupied units built prior to 1940\n        - DIS      weighted distances to
five Boston employment centres\n        - RAD      index of accessibility to
radial highways\n        - TAX      full-value property-tax rate per $10,000
\n        - PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk
- 0.63)^2 where Bk is the proportion of blacks by town\n        - LSTAT    %
lower status of the population\n        - MEDV     Median value of owner-occu
pied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator:
Harrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing datase
t.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nT
his dataset was taken from the StatLib library which is maintained at Carnegi
e Mellon University.\n\nThe Boston house-price data of Harrison, D. and Rubin
feld, D.L. 'Hedonic\nprices and the demand for clean air', J. Environ. Econom
ics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Reg
ression diagnostics\n...', Wiley, 1980.   N.B. Various transformations are us
ed in the table on\npages 244-261 of the latter.\n\nThe Boston house-price da
ta has been used in many machine learning papers that address regression\npro
blems.   \n     \n.. topic:: References\n\n   - Belsley, Kuh & Welsch, 'Regre
ssion diagnostics: Identifying Influential Data and Sources of Collinearity',
Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining Instance-Based and M
odel-Based Learning. In Proceedings on the Tenth International Conference of
Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufm
ann.\n",
 'filename': 'C:\\Users\\Alekhya\\Anaconda3\\lib\\site-packages\\sklearn\\dat
asets\\data\\boston_house_prices.csv'}
```

In [100]:

```
boston.keys()
```

Out[100]:

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [101]:

```
data1 = pd.DataFrame(boston.data,columns=boston.feature_names)
```

In [102]:

```
data1.head()
```

Out[102]:

	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 [103]:

```
boston.target
```

Out[103]:

```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
```

In [104]:

```
data1["LandPrice"] = boston.target
```

In [105]:
```
data1.head()
```

Out[105]:

	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 [107]:
```
data1.isnull().sum()
```

Out[107]:
```
CRIM         0
ZN           0
INDUS        0
CHAS         0
NOX          0
RM           0
AGE          0
DIS          0
RAD          0
TAX          0
PTRATIO      0
B            0
LSTAT        0
LandPrice    0
dtype: int64
```

In [108]:
```
data1.shape
```

Out[108]:
```
(506, 14)
```

In [109]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM         506 non-null float64
ZN           506 non-null float64
INDUS        506 non-null float64
CHAS         506 non-null float64
NOX          506 non-null float64
RM           506 non-null float64
AGE          506 non-null float64
DIS          506 non-null float64
RAD          506 non-null float64
TAX          506 non-null float64
PTRATIO      506 non-null float64
B            506 non-null float64
LSTAT        506 non-null float64
LandPrice    506 non-null float64
dtypes: float64(14)
memory usage: 55.4 KB
```

In [111]:
```
# seperate features and target
x = data1.iloc[:,0:-1]
y = data1["LandPrice"]
```

In [ ]:
```

```

In [113]:
```
x.head()
```

Out[113]:

	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 [130]:
```
from sklearn.linear_model import LinearRegression
```

In [131]:
```
model = LinearRegression()
```

In [132]:
```
model.fit(x,y)
```

Out[132]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [133]:
```
model.predict(x)
```

Out[133]:
```
array([30.00384338, 25.02556238, 30.56759672, 28.60703649, 27.94352423,
       25.25628446, 23.00180827, 19.53598843, 11.52363685, 18.92026211,
       18.99949651, 21.58679568, 20.90652153, 19.55290281, 19.28348205,
       19.29748321, 20.52750979, 16.91140135, 16.17801106, 18.40613603,
       12.52385753, 17.67103669, 15.83288129, 13.80628535, 15.67833832,
       13.38668561, 15.46397655, 14.70847428, 19.54737285, 20.8764282 ,
       11.45511759, 18.05923295,  8.81105736, 14.28275814, 13.70675891,
       23.81463526, 22.34193708, 23.10891142, 22.91502612, 31.35762569,
       34.21510225, 28.02056414, 25.20386628, 24.60979273, 22.94149176,
       22.09669817, 20.42320032, 18.03655088,  9.10655377, 17.20607751,
       21.28152535, 23.97222285, 27.6558508 , 24.04901809, 15.3618477 ,
       31.15264947, 24.85686978, 33.10919806, 21.77537987, 21.08493555,
       17.8725804 , 18.51110208, 23.98742856, 22.55408869, 23.37308644,
       30.36148358, 25.53056512, 21.11338564, 17.42153786, 20.78483633,
       25.20148859, 21.7426577 , 24.55744957, 24.04295712, 25.50499716,
       23.9669302 , 22.94545403, 23.35699818, 21.26198266, 22.42817373,
       28.40576968, 26.99486086, 26.03576297, 25.05873482, 24.78456674,
       27.79049195, 22.16853423, 25.89276415, 30.67461827, 30.83110623,
       27.1190194 , 27.41266734, 28.94122762, 29.08105546, 27.03977365,
       28.62459949, 24.72744978, 35.78159518, 35.11454587, 32.25102801,
       24.58022019, 25.59413475, 19.79013684, 20.31167129, 21.43482591,
       18.53994008, 17.18755992, 20.75049026, 22.64829115, 19.7720367 ,
       20.64965864, 26.52586744, 20.77323638, 20.71548315, 25.17208881,
       20.43025591, 23.37724626, 23.69043261, 20.33578364, 20.79180873,
       21.91632071, 22.47107777, 20.55738556, 16.36661977, 20.56099819,
       22.48178446, 14.61706633, 15.17876684, 18.93868592, 14.05573285,
       20.03527399, 19.41013402, 20.06191566, 15.75807673, 13.25645238,
       17.26277735, 15.87841883, 19.36163954, 13.81483897, 16.44881475,
       13.57141932,  3.98885508, 14.59495478, 12.1488148 ,  8.72822362,
       12.03585343, 15.82082058,  8.5149902 ,  9.71844139, 14.80451374,
       20.83858153, 18.30101169, 20.12282558, 17.28601894, 22.36600228,
       20.10375923, 13.62125891, 33.25982697, 29.03017268, 25.56752769,
       32.70827666, 36.77467015, 40.55765844, 41.84728168, 24.78867379,
       25.37889238, 37.20347455, 23.08748747, 26.40273955, 26.65382114,
       22.5551466 , 24.29082812, 22.97657219, 29.07194308, 26.5219434 ,
       30.72209056, 25.61669307, 29.13740979, 31.43571968, 32.92231568,
       34.72440464, 27.76552111, 33.88787321, 30.99238036, 22.71820008,
       24.7664781 , 35.88497226, 33.42476722, 32.41199147, 34.51509949,
       30.76109485, 30.28934141, 32.91918714, 32.11260771, 31.55871004,
       40.84555721, 36.12770079, 32.6692081 , 34.70469116, 30.09345162,
       30.64393906, 29.28719501, 37.07148392, 42.03193124, 43.18949844,
       22.69034796, 23.68284712, 17.85447214, 23.49428992, 17.00587718,
       22.39251096, 17.06042754, 22.73892921, 25.21942554, 11.11916737,
       24.51049148, 26.60334775, 28.35518713, 24.91525464, 29.68652768,
       33.18419746, 23.77456656, 32.14051958, 29.7458199 , 38.37102453,
       39.81461867, 37.58605755, 32.3995325 , 35.45665242, 31.23411512,
       24.48449227, 33.28837292, 38.0481048 , 37.16328631, 31.71383523,
       25.26705571, 30.10010745, 32.71987156, 28.42717057, 28.42940678,
       27.29375938, 23.74262478, 24.12007891, 27.40208414, 16.3285756 ,
       13.39891261, 20.01638775, 19.86184428, 21.2883131 , 24.0798915 ,
       24.20633547, 25.04215821, 24.91964007, 29.94563374, 23.97228316,
       21.69580887, 37.51109239, 43.30239043, 36.48361421, 34.98988594,
       34.81211508, 37.16631331, 40.98928501, 34.44634089, 35.83397547,
       28.245743  , 31.22673593, 40.8395575 , 39.31792393, 25.70817905,
```

```
       22.30295533, 27.20340972, 28.51169472, 35.47676598, 36.10639164,
       33.79668274, 35.61085858, 34.83993382, 30.35192656, 35.30980701,
       38.79756966, 34.33123186, 40.33963075, 44.67308339, 31.59689086,
       27.3565923 , 20.10174154, 27.04206674, 27.2136458 , 26.91395839,
       33.43563311, 34.40349633, 31.8333982 , 25.81783237, 24.42982348,
       28.45764337, 27.36266999, 19.53928758, 29.11309844, 31.91054611,
       30.77159449, 28.94275871, 28.88191022, 32.79887232, 33.20905456,
       30.76831792, 35.56226857, 32.70905124, 28.64244237, 23.58965827,
       18.54266897, 26.87889843, 23.28133979, 25.54580246, 25.48120057,
       20.53909901, 17.61572573, 18.37581686, 24.29070277, 21.32529039,
       24.88682244, 24.86937282, 22.86952447, 19.45123791, 25.11783401,
       24.66786913, 23.68076177, 19.34089616, 21.17418105, 24.25249073,
       21.59260894, 19.98446605, 23.33888   , 22.14060692, 21.55509929,
       20.61872907, 20.16097176, 19.28490387, 22.1667232 , 21.24965774,
       21.42939305, 30.32788796, 22.04734975, 27.70647912, 28.54794117,
       16.54501121, 14.78359641, 25.27380082, 27.54205117, 22.14837562,
       20.45944095, 20.54605423, 16.88063827, 25.40253506, 14.32486632,
       16.59488462, 19.63704691, 22.71806607, 22.20218887, 19.20548057,
       22.66616105, 18.93192618, 18.22846804, 20.23150811, 37.4944739 ,
       14.28190734, 15.54286248, 10.83162324, 23.80072902, 32.6440736 ,
       34.60684042, 24.94331333, 25.9998091 ,  6.126325  ,  0.77779806,
       25.30713064, 17.74061065, 20.23274414, 15.83331301, 16.83512587,
       14.36994825, 18.47682833, 13.4276828 , 13.06177512,  3.27918116,
        8.06022171,  6.12842196,  5.6186481 ,  6.4519857 , 14.20764735,
       17.21225183, 17.29887265,  9.89116643, 20.22124193, 17.94181175,
       20.30445783, 19.29559075, 16.33632779,  6.55162319, 10.89016778,
       11.88145871, 17.81174507, 18.26126587, 12.97948781,  7.37816361,
        8.21115861,  8.06626193, 19.98294786, 13.70756369, 19.85268454,
       15.22308298, 16.96071981,  1.71851807, 11.80578387, -4.28131071,
        9.58376737, 13.36660811,  6.89562363,  6.14779852, 14.60661794,
       19.6000267 , 18.12427476, 18.52177132, 13.1752861 , 14.62617624,
        9.92374976, 16.34590647, 14.07519426, 14.25756243, 13.04234787,
       18.15955693, 18.69554354, 21.527283  , 17.03141861, 15.96090435,
       13.36141611, 14.52079384,  8.81976005,  4.86751102, 13.06591313,
       12.70609699, 17.29558059, 18.740485  , 18.05901029, 11.51474683,
       11.97400359, 17.68344618, 18.12695239, 17.5183465 , 17.22742507,
       16.52271631, 19.41291095, 18.58215236, 22.48944791, 15.28000133,
       15.82089335, 12.68725581, 12.8763379 , 17.18668531, 18.51247609,
       19.04860533, 20.17208927, 19.7740732 , 22.42940768, 20.31911854,
       17.88616253, 14.37478523, 16.94776851, 16.98405762, 18.58838397,
       20.16719441, 22.97718032, 22.45580726, 25.57824627, 16.39147632,
       16.1114628 , 20.534816  , 11.54272738, 19.20496304, 21.86276391,
       23.46878866, 27.09887315, 28.56994302, 21.08398783, 19.45516196,
       22.22225914, 19.65591961, 21.32536104, 11.85583717,  8.22386687,
        3.66399672, 13.75908538, 15.93118545, 20.62662054, 20.61249414,
       16.88541964, 14.01320787, 19.10854144, 21.29805174, 18.45498841,
       20.46870847, 23.53334055, 22.37571892, 27.6274261 , 26.12796681,
       22.34421229])
```

In [134]:

```
model.score(x,y)*100
```

Out[134]: 74.06426641094095

In [116]:

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

In [119]:
```
rd = Ridge(alpha = 0.1)
```

In [118]:
```
1 help(Ridge)
```

```
Help on class Ridge in module sklearn.linear_model.ridge:

class Ridge(_BaseRidge, sklearn.base.RegressorMixin)
 |  Ridge(alpha=1.0, fit_intercept=True, normalize=False, copy_X=True, max_it
er=None, tol=0.001, solver='auto', random_state=None)
 |  
 |  Linear least squares with l2 regularization.
 |  
 |  Minimizes the objective function::
 |  
 |  ||y - Xw||^2_2 + alpha * ||w||^2_2
 |  
 |  This model solves a regression model where the loss function is
 |  the linear least squares function and regularization is given by
 |  the l2-norm. Also known as Ridge Regression or Tikhonov regularization.
 |  This estimator has built-in support for multi-variate regression
 |  (i.e., when y is a 2d-array of shape [n_samples, n_targets]).
 |  
 |  Read more in the :ref:`User Guide <ridge_regression>`.
 |  
 |  Parameters
 |  ----------
 |  alpha : {float, array-like}, shape (n_targets)
 |      Regularization strength; must be a positive float. Regularization
 |      improves the conditioning of the problem and reduces the variance of
 |      the estimates. Larger values specify stronger regularization.
 |      Alpha corresponds to ``C^-1`` in other linear models such as
 |      LogisticRegression or LinearSVC. If an array is passed, penalties are
 |      assumed to be specific to the targets. Hence they must correspond in
 |      number.
 |  
 |  fit_intercept : boolean
 |      Whether to calculate the intercept for this model. If set
 |      to false, no intercept will be used in calculations
 |      (e.g. data is expected to be already centered).
 |  
 |  normalize : boolean, optional, default False
 |      This parameter is ignored when ``fit_intercept`` is set to False.
 |      If True, the regressors X will be normalized before regression by
 |      subtracting the mean and dividing by the l2-norm.
 |      If you wish to standardize, please use
 |      :class:`sklearn.preprocessing.StandardScaler` before calling ``fit``
 |      on an estimator with ``normalize=False``.
 |  
 |  copy_X : boolean, optional, default True
 |      If True, X will be copied; else, it may be overwritten.
 |  
 |  max_iter : int, optional
 |      Maximum number of iterations for conjugate gradient solver.
 |      For 'sparse_cg' and 'lsqr' solvers, the default value is determined
 |      by scipy.sparse.linalg. For 'sag' solver, the default value is 1000.
 |  
 |  tol : float
 |      Precision of the solution.
```

```
|
|  solver : {'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'}
|      Solver to use in the computational routines:
|
|      - 'auto' chooses the solver automatically based on the type of data.
|
|      - 'svd' uses a Singular Value Decomposition of X to compute the Ridge
|        coefficients. More stable for singular matrices than
|        'cholesky'.
|
|      - 'cholesky' uses the standard scipy.linalg.solve function to
|        obtain a closed-form solution.
|
|      - 'sparse_cg' uses the conjugate gradient solver as found in
|        scipy.sparse.linalg.cg. As an iterative algorithm, this solver is
|        more appropriate than 'cholesky' for large-scale data
|        (possibility to set `tol` and `max_iter`).
|
|      - 'lsqr' uses the dedicated regularized least-squares routine
|        scipy.sparse.linalg.lsqr. It is the fastest and uses an iterative
|        procedure.
|
|      - 'sag' uses a Stochastic Average Gradient descent, and 'saga' uses
|        its improved, unbiased version named SAGA. Both methods also use an
|        iterative procedure, and are often faster than other solvers when
|        both n_samples and n_features are large. Note that 'sag' and
|        'saga' fast convergence is only guaranteed on features with
|        approximately the same scale. You can preprocess the data with a
|        scaler from sklearn.preprocessing.
|
|      All last five solvers support both dense and sparse data. However,
|      only 'sag' and 'saga' supports sparse input when `fit_intercept` is
|      True.
|
|      .. versionadded:: 0.17
|         Stochastic Average Gradient descent solver.
|      .. versionadded:: 0.19
|         SAGA solver.
|
|  random_state : int, RandomState instance or None, optional, default None
|      The seed of the pseudo random number generator to use when shuffling
|      the data.  If int, random_state is the seed used by the random number
|      generator; If RandomState instance, random_state is the random number
|      generator; If None, the random number generator is the RandomState
|      instance used by `np.random`. Used when ``solver`` == 'sag'.
|
|      .. versionadded:: 0.17
|         *random_state* to support Stochastic Average Gradient.
|
|  Attributes
|  ----------
|  coef_ : array, shape (n_features,) or (n_targets, n_features)
|      Weight vector(s).
|
|  intercept_ : float | array, shape = (n_targets,)
|      Independent term in decision function. Set to 0.0 if
|      ``fit_intercept = False``.
```

```
 |  
 |  n_iter_ : array or None, shape (n_targets,)
 |      Actual number of iterations for each target. Available only for
 |      sag and lsqr solvers. Other solvers will return None.
 |  
 |      .. versionadded:: 0.17
 |  
 |  See also
 |  --------
 |  RidgeClassifier : Ridge classifier
 |  RidgeCV : Ridge regression with built-in cross validation
 |  :class:`sklearn.kernel_ridge.KernelRidge` : Kernel ridge regression
 |      combines ridge regression with the kernel trick
 |  
 |  Examples
 |  --------
 |  >>> from sklearn.linear_model import Ridge
 |  >>> import numpy as np
 |  >>> n_samples, n_features = 10, 5
 |  >>> np.random.seed(0)
 |  >>> y = np.random.randn(n_samples)
 |  >>> X = np.random.randn(n_samples, n_features)
 |  >>> clf = Ridge(alpha=1.0)
 |  >>> clf.fit(X, y) # doctest: +NORMALIZE_WHITESPACE
 |  Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
 |        normalize=False, random_state=None, solver='auto', tol=0.001)
 |  
 |  Method resolution order:
 |      Ridge
 |      _BaseRidge
 |      abc.NewBase
 |      sklearn.linear_model.base.LinearModel
 |      abc.NewBase
 |      sklearn.base.BaseEstimator
 |      sklearn.base.RegressorMixin
 |      builtins.object
 |  
 |  Methods defined here:
 |  
 |  __init__(self, alpha=1.0, fit_intercept=True, normalize=False, copy_X=Tru
e, max_iter=None, tol=0.001, solver='auto', random_state=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  fit(self, X, y, sample_weight=None)
 |      Fit Ridge regression model
 |      
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape = [n_samples, n_features]
 |          Training data
 |      
 |      y : array-like, shape = [n_samples] or [n_samples, n_targets]
 |          Target values
 |      
 |      sample_weight : float or numpy array of shape [n_samples]
 |          Individual weights for each sample
 |      
```

```
 |      Returns
 |      -------
 |      self : returns an instance of self.
 |  
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |  
 |  __abstractmethods__ = frozenset()
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.linear_model.base.LinearModel:
 |  
 |  predict(self, X)
 |      Predict using the linear model
 |      
 |      Parameters
 |      ----------
 |      X : array_like or sparse matrix, shape (n_samples, n_features)
 |          Samples.
 |      
 |      Returns
 |      -------
 |      C : array, shape (n_samples,)
 |          Returns predicted values.
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |  
 |  __getstate__(self)
 |  
 |  __repr__(self)
 |      Return repr(self).
 |  
 |  __setstate__(self, state)
 |  
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |      
 |      Parameters
 |      ----------
 |      deep : boolean, optional
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |      
 |      Returns
 |      -------
 |      params : mapping of string to any
 |          Parameter names mapped to their values.
 |  
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |      
 |      The method works on simple estimators as well as on nested objects
 |      (such as pipelines). The latter have parameters of the form
 |      ``<component>__<parameter>`` so that it's possible to update each
 |      component of a nested object.
 |  
```

```
 |      Returns
 |      -------
 |      self
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.BaseEstimator:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
 |      list of weak references to the object (if defined)
 |  
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.RegressorMixin:
 |  
 |  score(self, X, y, sample_weight=None)
 |      Returns the coefficient of determination R^2 of the prediction.
 |      
 |      The coefficient R^2 is defined as (1 - u/v), where u is the residual
 |      sum of squares ((y_true - y_pred) ** 2).sum() and v is the total
 |      sum of squares ((y_true - y_true.mean()) ** 2).sum().
 |      The best possible score is 1.0 and it can be negative (because the
 |      model can be arbitrarily worse). A constant model that always
 |      predicts the expected value of y, disregarding the input features,
 |      would get a R^2 score of 0.0.
 |      
 |      Parameters
 |      ----------
 |      X : array-like, shape = (n_samples, n_features)
 |          Test samples. For some estimators this may be a
 |          precomputed kernel matrix instead, shape = (n_samples,
 |          n_samples_fitted], where n_samples_fitted is the number of
 |          samples used in the fitting for the estimator.
 |      
 |      y : array-like, shape = (n_samples) or (n_samples, n_outputs)
 |          True values for X.
 |      
 |      sample_weight : array-like, shape = [n_samples], optional
 |          Sample weights.
 |      
 |      Returns
 |      -------
 |      score : float
 |          R^2 of self.predict(X) wrt. y.
```

In [120]:

```
rd.fit(x,y)
```

Out[120]:

```
Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

In [122]:

```
rd.score(x,y)*100
```

Out[122]:

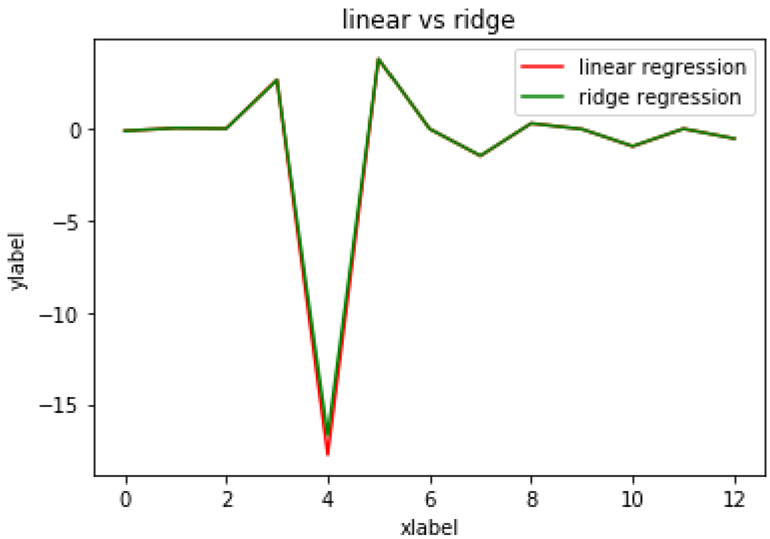
```
74.06002922228036
```

In [123]:
```
rd.predict(x)
```

Out[123]:
```
array([30.04164633, 24.99087654, 30.56235738, 28.65418856, 27.98110937,
       25.28351105, 22.99401212, 19.49937732, 11.46728387, 18.90419332,
       18.97087312, 21.57299999, 20.91823412, 19.6254361 , 19.32814596,
       19.37049176, 20.62952803, 16.94974945, 16.24598576, 18.4509557 ,
       12.52760065, 17.69942777, 15.85433067, 13.81722347, 15.70491352,
       13.41416113, 15.49855111, 14.73599841, 19.58319544, 20.91781426,
       11.4679403 , 18.08106463,  8.82039355, 14.29257421, 13.70923246,
       23.80342454, 22.33361154, 23.13292888, 22.94770419, 31.38031765,
       34.24998075, 28.05591938, 25.22935652, 24.63439678, 22.92985037,
       22.0775393 , 20.40064264, 17.96941634,  9.01333925, 17.1676709 ,
       21.25921938, 23.93929999, 27.67170273, 24.05648216, 15.36245728,
       31.20814843, 24.90051683, 33.11169862, 21.83330279, 21.10713092,
       17.87480913, 18.48007375, 24.00193815, 22.60324489, 23.41236028,
       30.33457983, 25.47994385, 21.09586087, 17.38088639, 20.75546251,
       25.17224245, 21.69495602, 24.52596045, 24.01135345, 25.45939482,
       23.88468994, 22.8234214 , 23.26479737, 21.17513268, 22.35108485,
       28.40215231, 26.95246748, 26.02810923, 25.03470499, 24.77130983,
       27.7694946 , 22.14535692, 25.85369489, 30.6534188 , 30.83240543,
       27.10278039, 27.38963669, 28.8711823 , 29.03524343, 26.94169148,
       28.58186944, 24.66575093, 35.73859434, 35.10760968, 32.21312064,
       24.57757695, 25.6036001 , 19.75881919, 20.29103742, 21.40846338,
       18.49455804, 17.14579297, 20.72123485, 22.61807221, 19.74311291,
       20.66265886, 26.49779259, 20.71508993, 20.65863375, 25.13023144,
       20.37624932, 23.35186394, 23.65650033, 20.2961484 , 20.76819322,
       21.89263652, 22.43318597, 20.50452763, 16.2972279 , 20.50387993,
       22.43616312, 14.54127411, 15.18006325, 18.94633194, 14.06008025,
       20.05120065, 19.42951478, 20.08281277, 15.77434899, 13.25781404,
       17.27305338, 15.88665202, 19.37127884, 13.80848744, 16.44910029,
       13.56760705,  3.95531205, 14.75635079, 12.32980839,  8.9013787 ,
       12.20728142, 16.00274351,  8.69138451,  9.89941792, 14.99603449,
       21.04313947, 18.49568991, 20.31063727, 17.47802286, 22.55296548,
       20.29185539, 13.80569981, 33.19821476, 28.95411317, 25.7825959 ,
       32.62505213, 36.72966705, 40.49338418, 41.79194389, 24.72583427,
       25.30859571, 37.15578332, 23.02672764, 26.32968049, 26.58658478,
       22.48012897, 24.2193064 , 22.91261127, 29.02679447, 26.4869673 ,
       30.73901429, 25.61675442, 29.11570853, 31.41020755, 32.92615523,
       34.7063593 , 27.74970753, 33.85876568, 30.9565073 , 22.67175688,
       24.75023857, 35.90691948, 33.35002283, 32.36125912, 34.45997777,
       30.75050881, 30.2692034 , 32.90930013, 32.10070568, 31.53645346,
       40.83071105, 36.08402804, 32.61375626, 34.65353801, 30.11549793,
       30.66808501, 29.24268244, 37.06621118, 41.99823151, 43.15955244,
       22.6944198 , 23.66787107, 17.80667604, 23.44694591, 16.89824364,
       22.30434301, 16.95970627, 22.68410977, 25.21911113, 11.10035668,
       24.49921625, 26.5636612 , 28.3201037 , 24.83114666, 29.62363875,
       33.11312749, 23.68322359, 32.08557443, 29.70560252, 38.34108381,
       39.78335077, 37.55285227, 32.36058516, 35.48827933, 31.25245309,
       24.44829432, 33.26325593, 38.04406444, 37.15599635, 31.67490448,
       25.24174308, 30.05850983, 32.7116793 , 28.41336377, 28.39289547,
       27.24597126, 23.67669784, 24.07941998, 27.41247903, 16.29463364,
       13.36485894, 20.03345208, 19.83524659, 21.28939471, 24.11839706,
       24.24129553, 25.08239598, 24.99082108, 30.02692351, 23.97464772,
       21.70551795, 37.47863688, 43.37611841, 36.53121049, 35.03678281,
       34.877882  , 37.23042989, 41.06263837, 34.49858956, 35.88897372,
       28.31308128, 31.28192032, 40.8578899 , 39.35583241, 25.65365921,
```

```
22.29216356, 27.22876613, 28.48912048, 35.46212573, 36.07139661,
33.77339919, 35.57250893, 34.82216683, 30.32869764, 35.2664569 ,
38.74175598, 34.30257505, 40.28845142, 44.63074482, 31.58887775,
27.3169374 , 20.12187008, 27.00901068, 27.16717864, 26.88767595,
33.43352556, 34.40372843, 31.83202931, 25.75154207, 24.33737245,
28.39386702, 27.27706279, 19.43640175, 29.0877494 , 31.90119146,
30.72433988, 28.88448814, 28.84434167, 32.77104223, 33.23173527,
30.75664101, 35.54291866, 32.68649823, 28.65063295, 23.58777604,
18.55594777, 26.89897161, 23.26027702, 25.54536844, 25.48288817,
20.54556242, 17.61096822, 18.38473427, 24.30773754, 21.35286626,
24.90802593, 24.88864143, 22.89218871, 19.44241911, 25.15495144,
24.74160742, 23.73880323, 19.3747408 , 21.16376037, 24.25699768,
21.5891205 , 19.97755737, 23.34334955, 22.25288308, 21.66562364,
20.72217455, 20.23748046, 19.35392702, 22.24831018, 21.32140809,
21.48895534, 30.33854063, 22.0939475 , 27.72882227, 28.61327119,
16.56539749, 14.79560485, 25.31875284, 27.59419723, 22.23183016,
20.52992818, 20.60578329, 16.94971136, 25.4909209 , 14.44113248,
16.71839071, 19.72621394, 22.82425972, 22.32003245, 19.3363201 ,
22.79823093, 19.04677838, 18.33436505, 20.3214836 , 37.56993866,
14.31934102, 15.57992877, 10.74891488, 23.74573625, 32.58999843,
34.55475914, 24.89037101, 25.97593446,  6.07479165,  0.71935322,
25.31360232, 17.73033339, 20.21999027, 15.82451558, 16.82038218,
14.41363885, 18.46771862, 13.43497813, 13.06638079,  3.27402054,
 8.06010737,  6.12903853,  5.62763941,  6.44687252, 14.2198978 ,
17.23593275, 17.33937186,  9.89895722, 20.24682564, 17.96378083,
20.32276972, 19.31129285, 16.3431333 ,  6.55644012, 10.90198481,
11.88978578, 17.82479548, 18.27220612, 12.99646191,  7.40132827,
 8.24129296,  8.01609303, 19.95346432, 13.59512987, 19.75288783,
15.13472826, 16.8501942 ,  1.57775802, 11.70282581, -4.30276841,
 9.55726571, 13.35133916,  6.8799398 ,  6.15617828, 14.65296253,
19.64553694, 18.16822692, 18.46185754, 13.09262274, 14.52955939,
 9.90169105, 16.26581049, 14.09527595, 14.25895557, 13.02494045,
18.05817913, 18.58915619, 21.44625903, 17.07390865, 15.99894206,
13.42027649, 14.58096501,  8.858732  ,  4.90821996, 13.12786642,
12.77930106, 17.36845286, 18.81004558, 18.1294529 , 11.56917063,
12.02418177, 17.75306392, 18.20519534, 17.56393028, 17.26649209,
16.5569376 , 19.46044681, 18.63509931, 22.54752163, 15.31568272,
15.86185601, 12.72119866, 12.92442312, 17.25038808, 18.58098734,
19.10387279, 20.23490729, 19.84695342, 22.50221342, 20.35047697,
17.92479559, 14.35590283, 16.85054504, 16.92228145, 18.5415355 ,
20.09470526, 22.85176089, 22.3937593 , 25.5616331 , 16.29380117,
16.00354595, 20.47828653, 11.4648035 , 19.14353933, 21.81492697,
23.37727196, 27.00367295, 28.47760302, 21.02120198, 19.43594898,
22.20642982, 19.60050645, 21.29104653, 11.77823048,  8.12949912,
 3.55997401, 13.68047679, 15.87722596, 20.67268898, 20.66950259,
16.95509651, 14.03413636, 19.14505331, 21.33524153, 18.47765366,
20.4918321 , 23.59754348, 22.4265918 , 27.67396591, 26.17696436,
22.39530856])
```

In [138]:
```
plt.plot(model.coef_,c="r",label = "linear regression")
plt.plot(rd.coef_,c="g",label = "ridge regression")
plt.title("linear vs ridge")
plt.xlabel("xlabel")
plt.ylabel("ylabel")
plt.legend()
plt.show()
```



In [ ]:
```

```

In [124]:
```
from sklearn.linear_model import Lasso
```

In [126]:
```
la = Lasso(alpha=100)
```

In [127]:
```
la.fit(x,y)
```

Out[127]:
```
Lasso(alpha=100, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

In [128]:
```
la.predict(x)
```

Out[128]:
```
array([25.06630096, 26.19878441, 26.18060604, 26.60808393, 26.61822272,
       26.60580605, 24.74591586, 24.75172222, 24.70585198, 24.7062093 ,
       24.73215926, 24.75172222, 24.72313707, 24.83560989, 24.76021654,
       24.82989286, 24.79072226, 24.79027562, 24.35363736, 24.80903462,
       24.74480735, 24.81609158, 24.83560989, 24.82506911, 24.82413116,
       24.41808795, 24.74619195, 24.43130858, 24.79559067, 24.76115449,
       24.67155789, 24.74552198, 24.10177533, 24.66530489, 24.17194296,
       25.42282353, 25.33644276, 25.42282353, 25.40732501, 25.98339288,
       25.98334822, 26.33621236, 26.32710084, 26.37663356, 26.35398876,
       26.38753165, 26.38753165, 26.3689513 , 26.38753165, 26.38753165,
       26.17182748, 26.16472585, 26.17781249, 26.17781249, 21.43815954,
       26.53000262, 24.70977839, 25.88731187, 25.29018275, 25.31796395,
       25.30996904, 25.23390572, 25.31796395, 25.31206826, 26.72770708,
       24.20645242, 24.20645242, 24.03559525, 24.03867709, 24.03867709,
       24.81873083, 24.78840376, 24.8507998 , 24.78943104, 22.91832202,
       22.86610945, 22.82336571, 22.88276924, 22.88026803, 22.92341375,
       25.3808797 , 25.37520733, 25.3808797 , 25.35291984, 26.09392483,
       26.07337925, 26.08986038, 26.08610858, 25.61157077, 25.60781897,
       25.59048922, 25.59660823, 25.60312921, 25.6090249 , 25.61157077,
       25.31190579, 25.46309447, 25.48573927, 25.4706874 , 25.48573927,
       23.21121421, 23.21487669, 21.76426934, 23.20991895, 23.2019687 ,
       23.20804305, 23.21527867, 23.17963655, 23.2133581 , 23.19544771,
       23.20554185, 22.2082694 , 22.20541088, 22.21412042, 22.17767435,
       21.98191069, 22.19804127, 22.20344565, 21.95386151, 22.1900017 ,
       27.29665302, 27.2453784 , 27.2472543 , 27.21250547, 27.25301599,
       27.27820666, 27.1632854 , 22.08786664, 22.10926084, 22.10926084,
       22.10095328, 22.10926084, 22.05950481, 22.07259145, 21.51013384,
       22.0993007 , 22.02596191, 22.09666551, 22.08755399, 22.10926084,
       22.06986692, 22.10926084, 22.82230598, 22.82230598, 22.82230598,
       21.82187018, 21.80561237, 22.7991252 , 22.64405073, 22.62109328,
       22.714665  , 22.57531236, 22.58281597, 22.21956116, 22.48339322,
       21.44267022, 21.4454394 , 22.67281454, 22.6302048 , 22.676745  ,
       22.56334233, 22.72194528, 22.7897457 , 22.78456464, 22.81431107,
       22.12223764, 22.69903249, 22.06618393, 22.37651153, 22.52368042,
       22.35507267, 22.60447815, 25.06630096, 25.06004796, 25.04990916,
       25.03977036, 25.04990916, 25.0604946 , 25.04115496, 27.22640828,
       27.22042326, 27.22640828, 27.2139916 , 27.22640828, 27.20005634,
       27.18268192, 27.20733662, 22.91363227, 22.86436754, 22.92716556,
       22.84132076, 22.89505192, 22.89853574, 25.70066385, 25.62620845,
       25.91422413, 24.37422774, 24.18400246, 24.35323552, 22.84327789,
       22.78700087, 23.96178141, 23.96897237, 26.5578772 , 26.54791706,
       25.46476736, 25.4557005 , 25.43140312, 25.39518037, 25.46476736,
       25.44846489, 25.45735308, 25.43814743, 25.41523464, 25.25051268,
       25.45016213, 25.46742691, 25.46733758, 25.48573927, 25.47162535,
       24.81238445, 24.82819561, 24.80653342, 24.83560989, 24.78268268,
       24.76906007, 24.79308947, 24.72475308, 24.7490058 , 24.7616458 ,
       24.7527576 , 24.74288679, 24.78652381, 24.75543746, 24.67169188,
       24.74561131, 24.79786855, 24.80510417, 24.90429542, 24.92381373,
       24.95717796, 24.97222983, 24.874549  , 24.8833032 , 24.24423025,
       24.31855166, 24.32324141, 24.26053272, 24.25414573, 24.3391419 ,
       24.35048664, 24.26468651, 24.30497371, 24.35325583, 24.64992417,
       24.6601523 , 26.10967507, 25.70524396, 25.67661414, 25.7152041 ,
       25.7190899 , 25.69930361, 25.6925593 , 25.72185908, 25.69715972,
       25.71730332, 25.68009796, 25.68219718, 25.70792382, 26.57241745,
```

```
26.56040275, 26.59725081, 26.58858593, 26.56987159, 25.94712142,
25.94712142, 25.91295323, 25.93171223, 25.94712142, 26.74405422,
26.70122115, 26.72319599, 26.6554849 , 27.11538503, 25.28725521,
24.97267648, 25.97276271, 25.12921671, 25.12921671, 25.01675198,
26.13586866, 26.13586866, 26.13586866, 25.21310437, 25.21310437,
25.21310437, 25.19501533, 25.21310437, 23.63803428, 23.65295216,
23.73906494, 24.36909135, 24.31486888, 24.34532993, 26.60384082,
26.60241156, 26.61822272, 26.61822272, 24.89852563, 24.89557779,
24.69105993, 24.89852563, 24.89584578, 24.88284846, 24.89312125,
24.89638175, 24.87083376, 24.89852563, 24.89097737, 24.89553313,
25.2550482 , 25.2550482 , 25.2550482 , 25.2292769 , 25.2550482 ,
25.24066629, 25.2550482 , 25.2550482 , 22.19147967, 22.15918737,
22.12953027, 24.88566231, 24.74376381, 26.54416526, 26.54278066,
26.57627889, 26.57627889, 26.56694405, 26.5728844 , 26.57627889,
26.57627889, 25.30831646, 22.39284255, 23.5143792 , 23.47449397,
23.84158167, 23.74765263, 23.89288065, 25.37523168, 24.21690791,
24.24839625, 22.53786748, 22.63411906, 27.29667738, 24.20639149,
24.17619842, 17.22107067, 17.28185879, 17.30012649, 17.27917893,
17.20691208, 17.10011973, 17.23473795, 17.11079449, 17.11753881,
17.11820877, 16.9454919 , 16.12094414, 17.21119986, 17.21035123,
17.28502995, 17.1693494 , 17.08774771, 17.30669214, 17.30669214,
17.30669214, 17.15536947, 17.30669214, 17.30669214, 17.29257822,
17.30669214, 17.30669214, 17.30669214, 17.30669214, 16.81060569,
17.30669214, 17.30669214, 17.30669214, 17.19958714, 17.30669214,
17.29566006, 17.22397385, 17.30669214, 17.30669214, 17.30669214,
17.2847173 , 17.30669214, 17.28971971, 17.30669214, 17.04433401,
17.30669214, 17.30669214, 17.21383505, 17.30669214, 17.00547606,
17.25340763, 17.18752777, 17.01722277, 16.93928356, 16.33506482,
15.54557854, 15.69051421, 15.66255436, 16.47624869, 15.92821765,
15.65567606, 15.63030673, 16.10281043, 15.6074386 , 15.75036438,
16.95764059, 16.9631343 , 16.83615367, 15.54522123, 15.55026829,
15.56826801, 15.64406334, 15.61802404, 15.96600366, 15.80516749,
15.90668946, 15.89722062, 15.9714527 , 15.98145751, 15.98342274,
16.02460323, 15.656748  , 15.57559295, 15.84192621, 17.30669214,
17.2823501 , 17.25782939, 17.30128776, 17.26126854, 16.60823172,
15.72629032, 16.95433543, 17.26926345, 17.30669214, 16.89269869,
15.53539508, 17.12084396, 17.2539436 , 17.21276311, 15.56380158,
15.76139647, 15.58077401, 15.54959833, 16.74977291, 17.30669214,
16.67393291, 17.28226077, 17.30669214, 17.29293554, 17.30669214,
17.02754023, 15.63227196, 17.01364963, 17.18091746, 17.30669214,
17.30669214, 17.29967985, 17.29092564, 17.20744805, 17.10873994,
16.88622236, 17.30361031, 17.094894  , 17.22986954, 17.24603801,
17.30669214, 17.28958571, 17.29945653, 17.28891575, 17.18980565,
17.2697101 , 17.28784381, 17.26792352, 16.3548717 , 16.12690508,
16.01247512, 16.33262887, 16.36295594, 23.07396897, 23.07396897,
23.05784515, 23.07396897, 23.07396897, 23.07396897, 23.0689219 ,
23.07396897, 25.52672485, 25.54865502, 25.54865502, 25.53324584,
25.54865502])
```

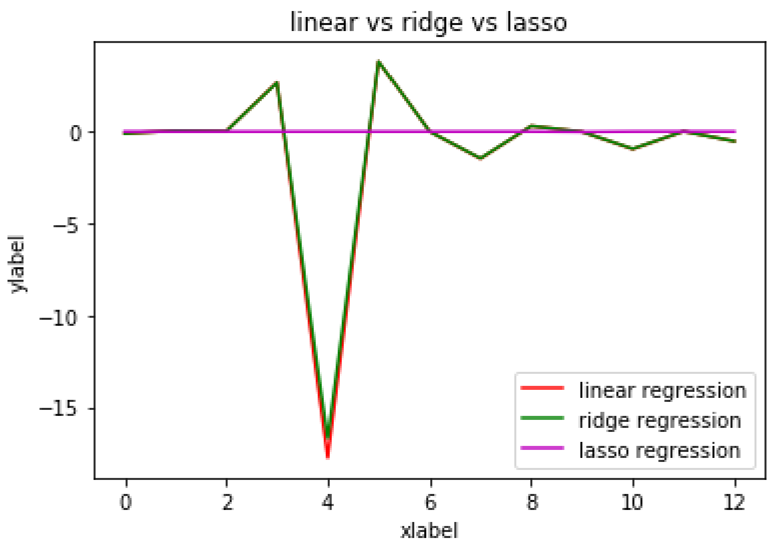
In [129]:

```
la.score(x,y)*100
```

Out[129]: 22.497922550751603

In [139]:

```
plt.plot(model.coef_,c="r",label = "linear regression")
plt.plot(rd.coef_,c="g",label = "ridge regression")
plt.plot(la.coef_,c="m",label = "lasso regression")
plt.title("linear vs ridge vs lasso")
plt.xlabel("xlabel")
plt.ylabel("ylabel")
plt.legend()
plt.show()
```



In [ ]:

```

```