Multi Linear Regression

Polynomial Regression

Linear Regression with Multiple Variables

- input = more than 1 features
- output = single target
- get or load the data
- data preprocessing
- define input and output
- apply the model or algothm
- apply train/test data
- evaluate the score
- y= ax^2+bx+c degree=2
- y= ax^3+bx^2+cx+1 degree=3

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

```

Prediction of the house price of boston dataset

1.get the data

In [2]:
```
from sklearn.datasets import load_boston
```

In [3]:
```
# create object
boston = load_boston()

```

In [4]:
```
boston.keys()
```

Out[4]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

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 (a
ttribute 14) is usually 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

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://a
rchive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carneg
ie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers tha
t address regression
problems.

.. topic:: References
```

```
  - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
 Data and Sources of Collinearity', Wiley, 1980. 244-261.
  - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-24
3, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [7]:

```
len(boston['feature_names'])
```

Out[7]:

```
13
```

In [8]:

```
boston['data']
```

Out[8]:

```
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]])
```

In [9]:

```
df = pd.DataFrame(boston['data'])
```

In [10]:

```
df
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

In [11]:
```
df.columns = boston['feature_names']
```

In [12]:
```
df
```

Out[12]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
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.
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.
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.
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.
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.
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.

506 rows × 13 columns

In [13]:
```
df['target']=boston['target']
```

In [14]:
```
df
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
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.
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.
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.
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.
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.
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.

506 rows × 14 columns

In [15]:
```
df.shape
```

Out[15]:
```
(506, 14)
```

In [16]:
```
# 2.Preprocessing the data
df.isna().sum()
```

Out[16]:
```
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
target     0
dtype: int64
```

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  target   506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [19]:
```
df.describe()
```

Out[19]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.79
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.10
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.10
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.20
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.18
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12

In [20]:
```
# 3. Define input and ouput
X = df[['RM']]
y = df['target']
```

- it is better to separate the data for training data and testing data
- we can say 70% data for training and 30% data for testing
- we have 506 rows available in dataframe
- how many rows for training and how many rows for testing

In [21]:
```
70*506/100 # training data
```

Out[21]: 354.2

In [22]:
```
506-354 # testing data

```

Out[22]: 152

In [23]:
```
from sklearn.model_selection import train_test_split
```

In [24]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.7)
```

In [26]:
```
X_train.shape
```

Out[26]: (354, 1)

In [27]:
```
X_test.shape
```

Out[27]: (152, 1)

In [28]:
```
y_train.shape
```
Out[28]: (354,)

In [29]:
```
y_test.shape
```
Out[29]: (152,)

In [30]:
```
# train the model
from sklearn.linear_model import LinearRegression
```

In [31]:
```
model = LinearRegression()
```

In [32]:
```
model.fit(X_train,y_train)
```
Out[32]: LinearRegression()

In [33]:
```
print('training score',model.score(X_train,y_train)*100)
```
```
training score 45.73896073809952
```

In [34]:
```
print('testing score',model.score(X_test,y_test)*100)
```
```
testing score 54.340073017363125
```

imporove the model above score very low so want to improve the model 1.giving the more examples 2.by taking different features 3.by parameter tuning

In [35]:
```
df.corr()
```

Out[35]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.
target	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.

In [50]:
```
X = df[['RM','PTRATIO','LSTAT']]
y = df['target']
```

In [51]:
```
from sklearn.model_selection import train_test_split
```

In [53]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.7)
```

In [54]:
```
X_train.shape
```

Out[54]: (354, 3)

In [55]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

Out[55]: LinearRegression()

In [56]:
```
model.score(X_train,y_train)*100
```

Out[56]: 71.1808054134889

In [58]:
```
model.score(X_test,y_test)*100
```
Out[58]: 59.86843710824545

In [67]:
```
X=df.drop('target',axis=1)
```
In [68]:
```
y=df['target']
```
In [69]:
```
from sklearn.model_selection import train_test_split
```
In [70]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.7)
```
In [71]:
```
X_train.shape
```
Out[71]: (354, 13)

In [72]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```
Out[72]: LinearRegression()

In [73]:
```
model.score(X_train,y_train)*100
```
Out[73]: 74.53585738547092

In [85]:
```
# Applying polynomial features  to linear regression

experience = [0,1,2,3,4,5,6,7,8]
salary = [5000,6000,7000,8000,15000,25000,40000,50000,80000]
```
In [86]:
```
df = pd.DataFrame({'experience':experience,'salary':salary})
```

In [87]:
```
df
```

Out[87]:

	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 [88]:
```
df.shape
```

Out[88]:
```
(9, 2)
```

In [89]:
```
df.isna().sum()
```

Out[89]:
```
experience    0
salary        0
dtype: int64
```

In [90]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   experience  9 non-null      int64
 1   salary      9 non-null      int64
dtypes: int64(2)
memory usage: 272.0 bytes
```

In [91]:
```
df.describe()
```

Out[91]:

	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 [92]:
```
X=df[['experience']]
y = df['salary']
```

In [93]:
```
X_train =X.head(7)
X_test =X.tail(2)
y_train = y.head(7)
y_test = y.tail(2)
```

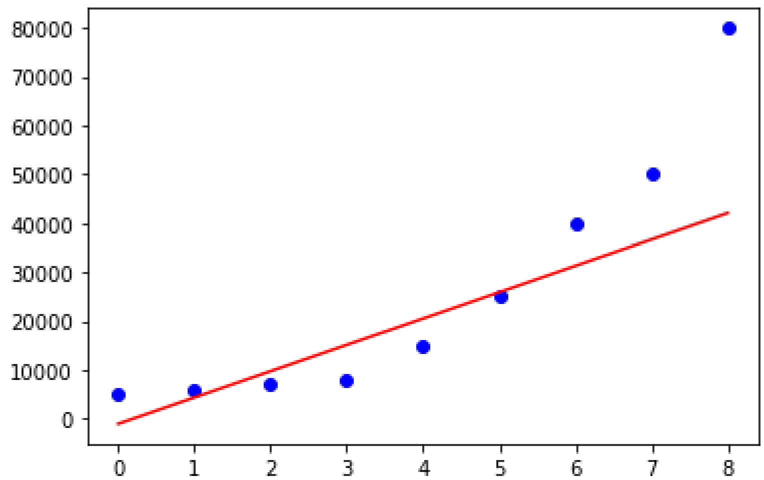
In [94]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

Out[94]: LinearRegression()

In [95]:
```
model.score(X_train,y_train)*100
```

Out[95]: 79.92498597868762

In [97]:
```
plt.scatter(df['experience'],df['salary'],c='blue',label ='true values')
plt.plot(df['experience'],model.predict(X),c='red',label = 'predicted line')
plt.show()
```



In [98]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [99]:
```
poly = PolynomialFeatures()

```

In [100]:
```
X_poly_train = poly.fit_transform(X_train)
```

In [101]:
```
X_poly_test = poly.transform(X_test)
```

In [102]:
```
from sklearn.linear_model import LinearRegression

```

In [103]:
```
model = LinearRegression()
```

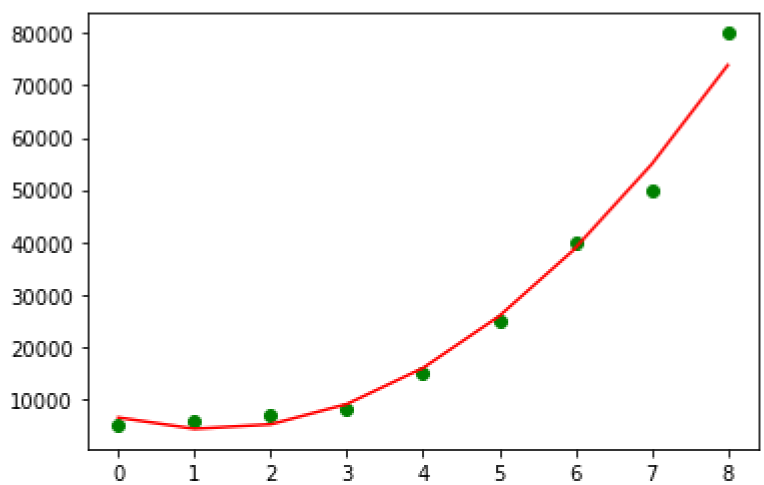
In [104]:
```
model.fit(X_poly_train,y_train)
```

Out[104]:
```
LinearRegression()
```

In [105]:
```
model.score(X_poly_train,y_train)*100
```

Out[105]:
```
98.77079828005235
```

In [106]:
```
plt.scatter(df['experience'],df['salary'],c='green',label='true values')
plt.plot(df['experience'],model.predict(poly.transform(X)),c='red',label='pr
plt.show()
```



In [ ]:
```

```