Todays topics:

- Linear Regression
  - Simple Linear Regression
  - Multiple Linear Regression
- Evaluation metrics

Algorithm

Algorithm is a step by step procedure which defines a set of instruction to be execuected in certain order to get the desired output

Model:

It reprasent what was learned by ML algorithms

Data model:

one of the main objectives in both ML and data science is finding an equation that best fits a given dataset is known data modeling

Linear Regression:

What is Linear Regression?

It is a linear approach to modeling the relation between dependent values and one or more independent values

Simple Linear Regression:

It provides the one independet values and one dependent values

y=mx+c # y is depedent values and x is independet and m=> cofficient c=>intercept

Multiple Linear Regression:

it provides one dependent values and two or more independent values

```
y=m1x1+m2x2+.....+c
```

Simple Linear Regression Example

In [10]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

#implement dataset
df=pd.DataFrame({"noofrooms":[2,3,4,5,6],"price":[200,300,400,500,600]})
df
```

Out[10]:

	noofrooms	price
0	2	200
1	3	300
2	4	400
3	5	500
4	6	600

Read X and y values

In [6]:
```
X=df[['noofrooms']]
y=df[['price']]
```

Take Linear Regression model

In [7]:
```
model=LinearRegression()

# pass the data to model
model.fit(X,y)
```

Out[7]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [9]:
```
# find the score for check the accuracy
model.score(X,y)*100
```

Out[9]:
```
100.0
```

In [12]:
```
# predict the new data
model.predict([[9]])
```

Out[12]:
```
array([[900.]])
```

In [23]:
```
a=np.array([7,8,9]).reshape(-1,1)
model.predict(a)
```

Out[23]:
```
array([[700.],
       [800.],
       [900.]])
```

In [24]:
```
y_pred=model.predict(X)
y_pred
```

Out[24]:
```
array([[200.],
       [300.],
       [400.],
       [500.],
       [600.]])
```

In [30]:
```
# using r2 score
from sklearn.metrics import r2_score

#r2_score(actual values, predicted values)
r2_score(y,y_pred)
```

Out[30]:
```
1.0
```

27-08-2020:

Today Topics:

- Multiple Linear Regression
- Model selections
  - train test split
  - cross validation
- use cases of Linear Regression
- Polynomial Regression

Multiple Linear Regression example

In [1]:
```
import pandas as pd
import numpy as np

d={"rooms":[2,3,4,5,6],"areasize":[20,30,40,50,60],"price":[200,300,400,500,600]
df=pd.DataFrame(d)
df
```

Out[1]:

	rooms	areasize	price
0	2	20	200
1	3	30	300
2	4	40	400
3	5	50	500
4	6	60	600

In [2]:
```
# Take X and y values
X=df[['rooms','areasize']]
y=df[['price']]
```

In [3]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X,y)
```

Out[3]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [4]:
```
# find the score
model.score(X,y)
```

Out[4]:
```
1.0
```

Datasets for Linear Regression

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

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

Out[2]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+0
2,
         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, 1
5. ,
        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', 'DI
S', '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 orde
r):\n        - CRIM     per capita crime rate by town\n        - ZN       propo
rtion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS
proportion of non-retail business acres per town\n        - CHAS     Charles Ri
ver 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 u
nits built prior to 1940\n        - DIS      weighted distances to five Boston
employment centres\n        - RAD      index of accessibility to radial highway
s\n        - TAX      full-value property-tax rate per $10,000\n        - PTRAT
IO  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 p
opulation\n        - MEDV     Median value of owner-occupied homes in $1000's\n
\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and Rubinfe
ld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://archive.ics.uci.
edu/ml/machine-learning-databases/housing/\n\n\nThis dataset was taken from the
StatLib library which is maintained at Carnegie Mellon University.\n\nThe Bosto
n house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the
demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-102, 197
8.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley, 1980.
N.B. Various transformations are used in the table on\npages 244-261 of the lat
ter.\n\nThe Boston house-price data has been used in many machine learning pape
rs that address regression\nproblems.   \n     \n.. topic:: References\n\n   -
Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data an
d Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Comb
ining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Inte
rnational Conference of Machine Learning, 236-243, University of Massachusetts,
Amherst. Morgan Kaufmann.\n",
 'filename': 'C:\\Users\\Kanakamma\\Anaconda3\\lib\\site-packages\\sklearn\\dat
asets\\data\\boston_house_prices.csv'}
```

In [3]:
```
boston_data.keys()
```

Out[3]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [4]:
```
data1=boston_data.data
data1
```

Out[4]:
```
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 [5]:
```
cn=boston_data.feature_names
cn
```

Out[5]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [6]:
```
df=pd.DataFrame(data1,columns=cn)
df['Target']=boston_data.target
df.head()
```

Out[6]:

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

In [7]:
```
df.shape
```

Out[7]:
```
(506, 14)
```

In [8]:
```
# find the missing values
df.isnull().sum()
```

Out[8]:
```
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 [9]:
```
# apply the simple linear regression for this boston dataset

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

In [10]:
```
X=df[['RM']]
y=df[['Target']]
```

In [11]:
```
model.fit(X,y)
```

Out[11]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [12]:
```
model.score(X,y)
```

Out[12]:
```
0.4835254559913343
```

correlation:

Correlation provides a relation between two features

- if value is 0 => their is no correlation between two features
- if value is 1=> their is posssitive relation
- if value is -1=> their is negative relation

In [13]:
```
# find the correlation
df.corr()
```

Out[13]:

	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 [14]:
```
X=df[['RM','ZN','LSTAT']]
y=df[['Target']]

model.fit(X,y)
model.score(X,y)
```

Out[14]: 0.6398856030562653

Model Selection

- Train Test Split: it is one of the model selection for spliting our dataset into two parts i.e training and testing it will provide hight efficiency and good accuracy for our model
- Cross Validaton(K-Fold)

In [58]:
```
X=df.drop('Target',axis=1)
y=df[['Target']]
```

In [59]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42
X_train.head()
```

Out[59]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.
116	0.13158	0.0	10.01	0.0	0.547	6.176	72.5	2.7301	6.0	432.0	17.8	393.30	12.
45	0.17142	0.0	6.91	0.0	0.448	5.682	33.8	5.1004	3.0	233.0	17.9	396.90	10.
16	1.05393	0.0	8.14	0.0	0.538	5.935	29.3	4.4986	4.0	307.0	21.0	386.85	6.
468	15.57570	0.0	18.10	0.0	0.580	5.926	71.0	2.9084	24.0	666.0	20.2	368.74	18.

In [50]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X_train,y_train)
```

Out[50]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [51]:
```
model.score(X_train,y_train)
```

Out[51]:
```
0.7434997532004697
```

In [52]:
```
model.score(X_test,y_test)
```

Out[52]:
```
0.7112260057484916
```

Cross validation:

Syntax: cross_val_score(modelname,inputvalues,targetvalues,cv=number of samples)

In [57]:
```
from sklearn.model_selection import cross_val_score
score=cross_val_score(model,X,y,cv=5)
score.mean()
```

Out[57]:
```
0.3532759243958782
```

improve accuracy:

In [33]:
```
X=df.drop('Target',axis=1)
y=df[['Target']]

#model.fit(X,y)
#model.score(X,y)
```

In [47]:
```
from sklearn.model_selection import cross_val_score,KFold
# Create 5 folds
# seed = 7
kfold = KFold(n_splits=10,shuffle=True)

# Create a model
model = LinearRegression()

# Train and evaluate multiple models using kfolds
results = cross_val_score(model, X, y, cv=kfold, scoring='r2')
print(results)
print("Mean:", results.mean())
print("Std:", results.std())
```

```
[0.78079328 0.66807429 0.71816623 0.80710498 0.70591488 0.77724365
 0.67229182 0.71217609 0.57070419 0.72025184]
Mean: 0.7132721245223294
Std: 0.0645227973803984
```

In [48]:
```
from sklearn.metrics import SCORERS
sorted(SCORERS.keys())
```

...

In [51]:
```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib as mpl
import numpy as np
import seaborn as sns
```

In [65]:
```
# Visualizing 3-D numeric data with Scatter Plots
# length, breadth and depth
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

xs = df['RM']
ys = df['LSTAT']
zs = df['Target']
ax.scatter(xs,ys,zs, s=50, alpha=0.6, edgecolors='w')


ax.set_xlabel('RM')
ax.set_ylabel('LSTAT')
ax.set_zlabel('Target')
```

...

In [ ]:
```
```