ML 20 - Multiple Linear Regression By Virat Tiwari

December 12, 2023

1 ML 20 - Multiple Linear Regression By Virat Tiwari

In multiple linear regression we have more than one independent feature but there is only one dependent feature

[6]:
```python
from sklearn.datasets import fetch_california_housing
```

[7]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

[8]:
```python
california=fetch_california_housing()
```

[9]:
```python
california
```

[9]:
```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, …,
2.55555556,
           37.88      , -122.23      ],
        [   8.3014    ,   21.        ,    6.23813708, …,    2.10984183,
           37.86      , -122.22      ],
        [   7.2574    ,   52.        ,    8.28813559, …,    2.80225989,
           37.85      , -122.24      ],
        …,
        [   1.7       ,   17.        ,    5.20554273, …,    2.3256351 ,
           39.43      , -121.22      ],
        [   1.8672    ,   18.        ,    5.32951289, …,    2.12320917,
           39.43      , -121.32      ],
        [   2.3886    ,   16.        ,    5.25471698, …,    2.61698113,
           39.37      , -121.24      ]]),
 'target': array([4.526, 3.585, 3.521, …, 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
```

```
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
  'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing
dataset\n--------------------------\n\n**Data Set Characteristics:**\n\n
:Number of Instances: 20640\n\n    :Number of Attributes: 8 numeric, predictive
attributes and the target\n\n    :Attribute Information:\n        - MedInc
median income in block group\n        - HouseAge      median house age in block
group\n        - AveRooms      average number of rooms per household\n        -
AveBedrms     average number of bedrooms per household\n        - Population
block group population\n        - AveOccup      average number of household
members\n        - Latitude      block group latitude\n        - Longitude
block group longitude\n\n    :Missing Attribute Values: None\n\nThis dataset was
obtained from the StatLib
repository.\nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html\n\nThe
target variable is the median house value for California districts,\nexpressed
in hundreds of thousands of dollars ($100,000).\n\nThis dataset was derived from
the 1990 U.S. census, using one row per census\nblock group. A block group is
the smallest geographical unit for which the U.S.\nCensus Bureau publishes
sample data (a block group typically has a population\nof 600 to 3,000
people).\n\nAn household is a group of people residing within a home. Since the
average\nnumber of rooms and bedrooms in this dataset are provided per
household, these\ncolumns may take surpinsingly large values for block groups
with few households\nand many empty houses, such as vacation resorts.\n\nIt can
be downloaded/loaded using
the\n:func:`sklearn.datasets.fetch_california_housing` function.\n\n.. topic::
References\n\n    - Pace, R. Kelley and Ronald Barry, Sparse Spatial
Autoregressions,\n      Statistics and Probability Letters, 33 (1997)
291-297\n'}
```

[10]:
```
california.keys()
```

[10]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR'])
```

[11]:
```
print(california.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640
```

```
    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

An household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surpinsingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

[12]:
```
california.data
```

[12]:
```
array([[   8.3252    ,   41.        ,    6.98412698, …,    2.55555556,
          37.88      , -122.23      ],
       [   8.3014    ,   21.        ,    6.23813708, …,    2.10984183,
          37.86      , -122.22      ],
       [   7.2574    ,   52.        ,    8.28813559, …,    2.80225989,
          37.85      , -122.24      ],
```

```
       …,
       [   1.7       ,   17.       ,    5.20554273, …,    2.3256351 ,
          39.43      , -121.22      ],
       [   1.8672    ,   18.       ,    5.32951289, …,    2.12320917,
          39.43      , -121.32      ],
       [   2.3886    ,   16.       ,    5.25471698, …,    2.61698113,
          39.37      , -121.24      ]])
```

[13]:
```
california.data.shape
```

[13]:
```
(20640, 8)
```

[14]:
```
california.target_names
```

[14]:
```
['MedHouseVal']
```

[15]:
```
california.feature_names
```

[15]:
```
['MedInc',
 'HouseAge',
 'AveRooms',
 'AveBedrms',
 'Population',
 'AveOccup',
 'Latitude',
 'Longitude']
```

[16]:
```
california.target
```

[16]:
```
array([4.526, 3.585, 3.521, …, 0.923, 0.847, 0.894])
```

[17]:
```
# Now we made the Dataset

dataset=pd.DataFrame(california.data,columns=california.feature_names)
dataset.head()
```

[17]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude
0    -122.23
1    -122.22
2    -122.24
```

```
3    -122.25
4    -122.25
```

[18]:
```
# Here we add OUTPUT FEATURE

dataset["Price"]=california.target
```

[19]:
```
dataset.head()
```

[19]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  Price
0    -122.23  4.526
1    -122.22  3.585
2    -122.24  3.521
3    -122.25  3.413
4    -122.25  3.422
```

[20]:
```
# EDA
```

[21]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   Price       20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

[22]:
```
dataset.describe()
```

[22]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744
std	1.899822	12.585558	2.474173	0.473911	1132.462122
min	0.499900	1.000000	0.846154	0.333333	3.000000
25%	2.563400	18.000000	4.440716	1.006079	787.000000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000
max	15.000100	52.000000	141.909091	34.066667	35682.000000

	AveOccup	Latitude	Longitude	Price
count	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.070655	35.631861	-119.569704	2.068558
std	10.386050	2.135952	2.003532	1.153956
min	0.692308	32.540000	-124.350000	0.149990
25%	2.429741	33.930000	-121.800000	1.196000
50%	2.818116	34.260000	-118.490000	1.797000
75%	3.282261	37.710000	-118.010000	2.647250
max	1243.333333	41.950000	-114.310000	5.000010

[23]:

```
dataset.isnull().sum()
```

[23]:

```
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
Price         0
dtype: int64
```

[24]:

```
# Here we get PEARSON CORRELATION

dataset.corr()
```

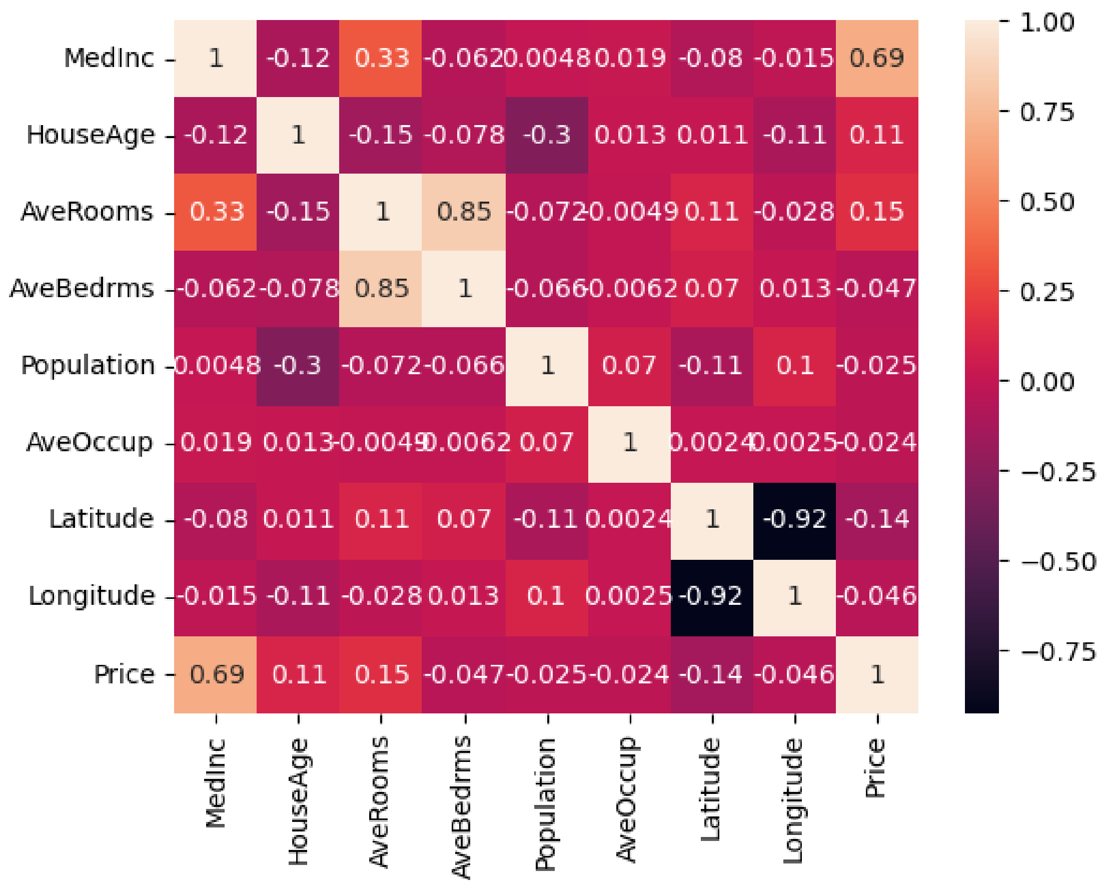
[24]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476
Price	0.688075	0.105623	0.151948	-0.046701	-0.024650	-0.023737

```
            Latitude  Longitude     Price
MedInc     -0.079809  -0.015176  0.688075
HouseAge    0.011173  -0.108197  0.105623
AveRooms    0.106389  -0.027540  0.151948
AveBedrms   0.069721   0.013344 -0.046701
Population -0.108785   0.099773 -0.024650
AveOccup    0.002366   0.002476 -0.023737
Latitude    1.000000  -0.924664 -0.144160
Longitude  -0.924664   1.000000 -0.045967
Price      -0.144160  -0.045967  1.000000
```

[25]:
```
import seaborn as sns
sns.heatmap(dataset.corr(),annot=True)
```

[25]: <AxesSubplot: >



[26]:
```
dataset.head()
```

[26]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  Price
0    -122.23  4.526
1    -122.22  3.585
2    -122.24  3.521
3    -122.25  3.413
4    -122.25  3.422
```

[27]:
```python
#INDEPENDENT AND DEPENDENT FEATURE
```

[28]:
```python
x=dataset.iloc[:,:-1] # Independent Feature
y=dataset.iloc[:,-1] # Dependent Feature
```

[29]:
```python
# Independent Features

x.head()
```

[29]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude
0    -122.23
1    -122.22
2    -122.24
3    -122.25
4    -122.25
```

[30]:
```python
# Dependent Features

y
```

[30]:
```
0         4.526
1         3.585
2         3.521
3         3.413
4         3.422
          …
```

```
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
Name: Price, Length: 20640, dtype: float64
```

[31]:
```
x.shape,y.shape
```

[31]:
```
((20640, 8), (20640,))
```

[32]:
```
# TRAIN - TEST SPLIT
```

[33]:
```
from sklearn.model_selection import train_test_split
```

[34]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
↪33,random_state=10)
```

[35]:
```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

[35]:
```
((13828, 8), (13828,), (6812, 8), (6812,))
```

Note - In Multiple Linear Regression we should Scale down all the Indepndent Features and for doing this task we have to use Standard Scaler Concept

IMPORTANT - WE ALWAYS SCALE DOWN INPUT DATA (X TRAIN AND X TEST)

[36]:
```
from sklearn.preprocessing import StandardScaler
```

[37]:
```
scaler=StandardScaler()
```

[38]:
```
x_train_scaled=scaler.fit_transform(x_train)
```

[39]:
```
x_test_scaled=scaler.transform(x_test)
```

[40]:
```
x_train_scaled
```

[40]:
```
array([[-0.72986836,  1.22081889, -0.70305988, …,  0.05861244,
         0.96929441, -1.43979718],
       [-0.61046678, -0.28439808,  0.07828001, …,  0.13015917,
        -0.75823526,  1.08204942],
       [ 0.00784578, -0.60128586, -0.2447376 , …, -0.09793279,
         0.94594941, -1.2454256 ],
       …,
       [ 0.88684913, -1.78961504, -0.21300658, …,  0.09549475,
         0.78720344, -1.10587678],
       [-0.87672223,  0.50782138, -1.10043274, …,  0.18513096,
        -0.77224225,  0.66838683],
```

```
       [-0.62742573, -0.99739558, -0.60483749, …, -0.08418874,
         0.77786545, -1.15073176]])
```

[41]:
```python
x_test_scaled=scaler.transform(x_test)
```

[42]:
```python
x_test_scaled
```

[42]:
```
array([[ 0.75154854, -1.31428337, -0.39376169, …,  0.12606697,
        -0.68820027,  0.19491761],
       [ 0.05935857, -0.12595418, -0.33070668, …, -0.12021013,
         0.89459042, -1.36503888],
       [ 0.34405687, -1.31428337, -0.41007104, …, -0.15581759,
        -0.91698123,  0.89764561],
       …,
       [ 0.36483158,  0.27015554,  0.04216837, …, -0.08014641,
        -0.46875731, -0.43803598],
       [-0.90412152, -0.91817364,  0.66736933, …, -0.10263685,
         2.51006411, -1.96808915],
       [-0.43377577,  1.22081889, -0.44835491, …,  0.2807072 ,
        -0.74422826,  0.69330627]])
```

MODEL TRAINING

[43]:
```python
from sklearn.linear_model import LinearRegression
```

[44]:
```python
regression=LinearRegression()
regression
```

[44]:
```
LinearRegression()
```

[45]:
```python
regression.fit(x_train_scaled,y_train)
```

[45]:
```
LinearRegression()
```

[46]:
```python
# SLOPES OF 8 FEATURE

regression.coef_
```

[46]:
```
array([ 0.82872299,  0.1231163 , -0.27068752,  0.32859106,  0.00213572,
       -0.02810091, -0.93017985, -0.89505497])
```

[47]:
```python
# INTERCEPT

regression.intercept_
```

[47]:
```
2.0634768086491184
```

[48]:
```
# PREDICTION ABOUT THE TEST DATA

y_pred_test=regression.predict(x_test_scaled)
```

[49]:
```
y_pred_test
```

[49]:
```
array([3.00397485, 2.58011486, 2.3489077 , …, 3.09003708, 0.79152007,
       2.04477012])
```

[50]:
```
# PERFORMANCE METRICS

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,y_pred_test))
print(mean_absolute_error(y_test,y_pred_test))
print(np.sqrt(mean_squared_error(y_test,y_pred_test)))
```

```
0.5522332399363619
0.537105694300796
0.7431239734636219
```

PYTHON " PICKLING CONCEPT " -

IN THIS MODEULE WE HAVE TO SERIALISED AND DE-SERIALISED A PYTHON ONJECT STRUCTURE . ANY O=BJECT IN PYTHON CAN BE PICKELED SO THAT IT CAN BE SAVED ON DISK . WHAT PICKLE DOES IT THAT IT SERIALISES THE OBJECT FIRST BEFORE WRITING IT TO FILE . PICKLING IS A WAY TO COVERT PYTHON OBJECT INTO A CHARACTER STREAM . THE IDEA IS THAT , THIS CHARACTER STREAM CONTAIN ALL THE INFORMATION NECESSORY TO RECONSTRUCT THE OBJECT IN ANOTHET PYTHON OBJECT.

[59]:
```
import pickle
pickle.dump(scaler,open("scaler.pkl","wb"))
pickle.dump(regression,open("regressor","wb"))
```

[61]:
```
model_regressor=pickle.load(open("regressor","rb"))
model_regressor.predict(x_test_scaled)
```

[61]:
```
array([3.00397485, 2.58011486, 2.3489077 , …, 3.09003708, 0.79152007,
       2.04477012])
```

[62]:
```
standard_scaler=pickle.load(open("scaler.pkl","rb"))
```

[63]:
```
model_regressor.predict(standard_scaler.transform(x_test))
```

[63]:
```
array([3.00397485, 2.58011486, 2.3489077 , …, 3.09003708, 0.79152007,
       2.04477012])
```

```
[ ]:
```