Predicting Housing Prices using Regression Algorithms

March 4, 2022

1 Regression?

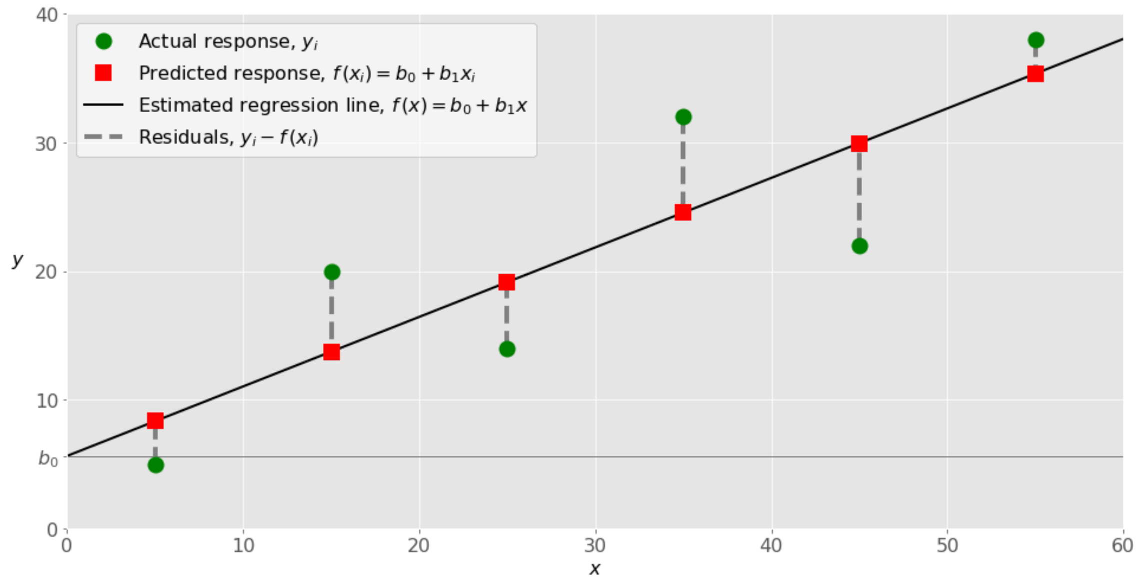
Regression searches for relationships among variables. you need regression to answer whether and how some phenomenon influences the other or how several variables are related. In short, the ability of a model to predict continuous or real values based on a training dataset is called Regression.
It is used to determine how and up to what extent the experience or gender impact salaries. For example, observe an employee in a company and try to understand how their salaries depend on the features, such as experience, level of education, role, city they work in, and so on.
Now, we know that, this is a regression problem where data related to each employee represent one observation. The presumption is that the experience, education, role, and city are the independent features, while the salary depends on them. Similar is the case for the house price prediction in an area based on numbers of bedrooms, distances to the city center, and so on. (which we will be trying to predict later.)
Generally, in regression analysis, you need to find a function (observation) that maps (RS) some features (2 or more) or variables to others sufficiently well (dependability).
dependent features are called the dependent variables, outputs, or responses.
independent features are called the independent variables, inputs, or predictors.
It is a common practice to denote the outputs with and inputs with . If there are two or more independent variables, they can be represented as the vector = (, ...,), where is the number of inputs.

2 Linear Regression?

Simple Linear Regression
Simple or single-variate linear regression is the simplest case of linear regression with a single independent variable, = . The following figure illustrates simple linear regression:

[5]:
```python
# load and show an image with Pillow
from PIL import Image
import numpy as np
image = Image.open(r'C:\Users\nomaniqbal\Downloads\linearregression.png')
image
```

[5]:



When implementing simple linear regression, you typically start with a given set of input-output (-) pairs (green circles). These pairs are your observations. For example, the leftmost observation (green circle) has the input = 5 and the actual output (response) = 5. The next one has = 15 and = 20, and so on.

The estimated regression function (black line) has the equation ( ) = + . Your goal is to calculate the optimal values of the predicted weights and (regression coefficients) that minimize SSR (The differences - ( ) for all observations = 1, ..., , are called the residuals. Regression is about determining the best predicted weights, that is the weights corresponding to the smallest residuals. To get the best weights, you usually minimize the sum of squared residuals (SSR) for all observations = 1, ..., : SSR = Σ (- ( ))².) and determine the estimated regression function. The value of , also called the intercept, shows the point where the estimated regression line crosses the axis. It is the value of the estimated response ( ) for = 0. The value of determines the slope of the estimated regression line.

The predicted responses (red squares) are the points on the regression line that correspond to the input values. For example, for the input = 5, the predicted response is (5) = 8.33 (represented with the leftmost red square).

The residuals (vertical dashed gray lines) can be calculated as - ( ) = - - for = 1, ..., . They are the distances between the green circles and red squares. When you implement linear regression, you are actually trying to minimize these distances and make the red squares as close to the predefined green circles as possible.

2.1 Multiple Linear Regression?

Multiple or multivariate linear regression is a case of linear regression with two or more independent variables.

If there are just two independent variables, the estimated regression function is (,) = + + . It represents a regression plane in a three-dimensional space. The goal of regression is to determine the values of the weights , , and such that this plane is as close as possible to the actual responses and yield the minimal SSR.

The case of more than two independent variables is similar, but more general. The estimated regression function is (, ...,) = + + + , and there are + 1 weights to be determined when the number of inputs is .

2.2 Polynomial Regression?

You can regard polynomial regression as a generalized case of linear regression. You assume the polynomial dependence between the output and inputs and, consequently, the polynomial estimated regression function.

In other words, in addition to linear terms like , your regression function can include non-linear terms such as ^{2}, ^{3}, or even , ^{2} , and so on.

The simplest example of polynomial regression has a single independent variable, and the estimated regression function is a polynomial of degree 2: ( ) = + + ^{2}.

3 Predicting Housing Prices using Regression Algorithms

how to solve a supervised regression problem using the famous Boston housing price dataset? Other than location and square footage, a house value is determined by various other factors. Let's analyze this problem in detail and come up with our own machine learning model to predict a housing price.

Boston Housing Prices Dataset

In this dataset, each row describes a boston town or suburb. There are 506 rows and 13 attributes (features) with a target column (price).

The problem that we are going to solve here is that given a set of features that describe a house in Boston, our machine learning model must predict the house price. To train our machine learning model with boston housing data, we will be using scikit-learn's boston dataset.

We will use pandas and scikit-learn to load and explore the dataset. The dataset can easily be loaded from scikit-learn's datasets module using load_boston function.

[6]:
```python
import pandas as pd
import numpy as np
#To visualize data using 2D plots.
import matplotlib.pyplot as plt
#To make 2D plots look pretty and readable.
import seaborn as sns
import warnings
import random
import os
#To create machine learning models easily and make predictions.
from sklearn.datasets import load_boston
pd.options.display.float_format = '{:,.2f}'.format
dataset = load_boston()
```

[7]:
```
#find keys in the dataset
print("[INFO] keys : {}".format(dataset.keys()))
```

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

[8]:
```
#There are 13 features and 1 target that are accessed using data key and target␣
 ↪key.
#We can easily access the shape of features and target using shape.
print("[INFO] features shape : {}".format(dataset.data.shape))
print("[INFO] target shape   : {}".format(dataset.target.shape))
```

```
[INFO] features shape : (506, 13)
[INFO] target shape   : (506,)
```

[9]:
```
#names of the attributes
print("[INFO] feature names")
print(dataset.feature_names)
```

```
[INFO] feature names
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

[10]:
```
#desciption of each column
print("[INFO] dataset summary")
print(dataset.DESCR)
```

```
[INFO] dataset summary
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 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/


This dataset was taken from the StatLib library which is maintained at Carnegie
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 that
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, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

Analyze the dataset

[11]:
```python
df = pd.DataFrame(dataset.data)
print("[INFO] df type : {}".format(type(df)))
print("[INFO] df shape: {}".format(df.shape))
df.head()
```

```
[INFO] df type : <class 'pandas.core.frame.DataFrame'>
[INFO] df shape: (506, 13)
```

[11]:
```
     0     1    2    3    4    5     6    7    8      9    10     11   12
0 0.01 18.00 2.31 0.00 0.54 6.58 65.20 4.09 1.00 296.00 15.30 396.90 4.98
1 0.03  0.00 7.07 0.00 0.47 6.42 78.90 4.97 2.00 242.00 17.80 396.90 9.14
2 0.03  0.00 7.07 0.00 0.47 7.18 61.10 4.97 2.00 242.00 17.80 392.83 4.03
3 0.03  0.00 2.18 0.00 0.46 7.00 45.80 6.06 3.00 222.00 18.70 394.63 2.94
4 0.07  0.00 2.18 0.00 0.46 7.15 54.20 6.06 3.00 222.00 18.70 396.90 5.33
```

[12]:
```python
# looks confusing right
df.columns = dataset.feature_names
print(df.head())
```

```
   CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD    TAX  PTRATIO      B  \
0  0.01 18.00   2.31  0.00 0.54 6.58 65.20 4.09 1.00 296.00    15.30 396.90
1  0.03  0.00   7.07  0.00 0.47 6.42 78.90 4.97 2.00 242.00    17.80 396.90
2  0.03  0.00   7.07  0.00 0.47 7.18 61.10 4.97 2.00 242.00    17.80 392.83
3  0.03  0.00   2.18  0.00 0.46 7.00 45.80 6.06 3.00 222.00    18.70 394.63
4  0.07  0.00   2.18  0.00 0.46 7.15 54.20 6.06 3.00 222.00    18.70 396.90

   LSTAT
0   4.98
1   9.14
2   4.03
3   2.94
4   5.33
```

[13]:
```python
dataset.feature_names
```

[13]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

[14]:
```python
df.columns
```

[14]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

[15]:
```python
df.head()
```

[15]:
```
   CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD    TAX  PTRATIO      B  \
0  0.01 18.00   2.31  0.00 0.54 6.58 65.20 4.09 1.00 296.00    15.30 396.90
1  0.03  0.00   7.07  0.00 0.47 6.42 78.90 4.97 2.00 242.00    17.80 396.90
2  0.03  0.00   7.07  0.00 0.47 7.18 61.10 4.97 2.00 242.00    17.80 392.83
3  0.03  0.00   2.18  0.00 0.46 7.00 45.80 6.06 3.00 222.00    18.70 394.63
4  0.07  0.00   2.18  0.00 0.46 7.15 54.20 6.06 3.00 222.00    18.70 396.90
```

```
   LSTAT
0   4.98
1   9.14
2   4.03
3   2.94
4   5.33
```

[16]:
```
#target column
df["PRICE"] = dataset.target
print(df.head())
```

```
   CRIM    ZN  INDUS  CHAS  NOX   RM   AGE  DIS  RAD    TAX  PTRATIO      B  \
0  0.01 18.00   2.31  0.00 0.54 6.58 65.20 4.09 1.00 296.00    15.30 396.90
1  0.03  0.00   7.07  0.00 0.47 6.42 78.90 4.97 2.00 242.00    17.80 396.90
2  0.03  0.00   7.07  0.00 0.47 7.18 61.10 4.97 2.00 242.00    17.80 392.83
3  0.03  0.00   2.18  0.00 0.46 7.00 45.80 6.06 3.00 222.00    18.70 394.63
4  0.07  0.00   2.18  0.00 0.46 7.15 54.20 6.06 3.00 222.00    18.70 396.90

   LSTAT  PRICE
0   4.98  24.00
1   9.14  21.60
2   4.03  34.70
3   2.94  33.40
4   5.33  36.20
```

[17]:
```
#check the data type
print(df.dtypes)
```

```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
PRICE      float64
dtype: object
```

[18]:
```
#descriptive statistics
#statistical summary of the dataset using the describe() function. Using this␣
 ↪function,
#we can understand the count, min, max, mean and standard deviation for each␣
↪attribute (column) in the dataset.
df.describe()
```

```
[18]:        CRIM     ZN  INDUS   CHAS    NOX     RM    AGE    DIS    RAD    TAX  \
count  506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00 506.00
mean     3.61  11.36  11.14   0.07   0.55   6.28  68.57   3.80   9.55 408.24
std      8.60  23.32   6.86   0.25   0.12   0.70  28.15   2.11   8.71 168.54
min      0.01   0.00   0.46   0.00   0.39   3.56   2.90   1.13   1.00 187.00
25%      0.08   0.00   5.19   0.00   0.45   5.89  45.02   2.10   4.00 279.00
50%      0.26   0.00   9.69   0.00   0.54   6.21  77.50   3.21   5.00 330.00
75%      3.68  12.50  18.10   0.00   0.62   6.62  94.07   5.19  24.00 666.00
max     88.98 100.00  27.74   1.00   0.87   8.78 100.00  12.13  24.00 711.00

       PTRATIO      B  LSTAT  PRICE
count   506.00 506.00 506.00 506.00
mean     18.46 356.67  12.65  22.53
std       2.16  91.29   7.14   9.20
min      12.60   0.32   1.73   5.00
25%      17.40 375.38   6.95  17.02
50%      19.05 391.44  11.36  21.20
75%      20.20 396.23  16.96  25.00
max      22.00 396.90  37.97  50.00
```

correlation between attributes
Pandas offers three different ways to find correlation between attributes (columns).
The output of each of these correlation functions fall within the range [-1, 1].

1 - Positively correlated
-1 - Negatively correlated.
0 - Not correlated.

```
[19]: # correlation between attributes
print("PEARSON CORRELATION")
print(df.corr(method="pearson"))
sns.heatmap(df.corr(method="pearson"))
plt.savefig("plots/heatmap_pearson.png")
plt.clf()
plt.close()

print("SPEARMAN CORRELATION")
print(df.corr(method="spearman"))
sns.heatmap(df.corr(method="spearman"))
plt.savefig("plots/heatmap_spearman.png")
plt.clf()
plt.close()

print("KENDALL CORRELATION")
```

```python
print(df.corr(method="kendall"))
sns.heatmap(df.corr(method="kendall"))
plt.savefig("plots/heatmap_kendall.png")
plt.clf()
plt.close()
```

```
PEARSON CORRELATION
          CRIM    ZN  INDUS  CHAS   NOX    RM   AGE   DIS   RAD   TAX  PTRATIO  \
CRIM      1.00 -0.20   0.41 -0.06  0.42 -0.22  0.35 -0.38  0.63  0.58     0.29
ZN       -0.20  1.00  -0.53 -0.04 -0.52  0.31 -0.57  0.66 -0.31 -0.31    -0.39
INDUS     0.41 -0.53   1.00  0.06  0.76 -0.39  0.64 -0.71  0.60  0.72     0.38
CHAS     -0.06 -0.04   0.06  1.00  0.09  0.09  0.09 -0.10 -0.01 -0.04    -0.12
NOX       0.42 -0.52   0.76  0.09  1.00 -0.30  0.73 -0.77  0.61  0.67     0.19
RM       -0.22  0.31  -0.39  0.09 -0.30  1.00 -0.24  0.21 -0.21 -0.29    -0.36
AGE       0.35 -0.57   0.64  0.09  0.73 -0.24  1.00 -0.75  0.46  0.51     0.26
DIS      -0.38  0.66  -0.71 -0.10 -0.77  0.21 -0.75  1.00 -0.49 -0.53    -0.23
RAD       0.63 -0.31   0.60 -0.01  0.61 -0.21  0.46 -0.49  1.00  0.91     0.46
TAX       0.58 -0.31   0.72 -0.04  0.67 -0.29  0.51 -0.53  0.91  1.00     0.46
PTRATIO   0.29 -0.39   0.38 -0.12  0.19 -0.36  0.26 -0.23  0.46  0.46     1.00
B        -0.39  0.18  -0.36  0.05 -0.38  0.13 -0.27  0.29 -0.44 -0.44    -0.18
LSTAT     0.46 -0.41   0.60 -0.05  0.59 -0.61  0.60 -0.50  0.49  0.54     0.37
PRICE    -0.39  0.36  -0.48  0.18 -0.43  0.70 -0.38  0.25 -0.38 -0.47    -0.51

             B  LSTAT  PRICE
CRIM     -0.39   0.46  -0.39
ZN        0.18  -0.41   0.36
INDUS    -0.36   0.60  -0.48
CHAS      0.05  -0.05   0.18
NOX      -0.38   0.59  -0.43
RM        0.13  -0.61   0.70
AGE      -0.27   0.60  -0.38
DIS       0.29  -0.50   0.25
RAD      -0.44   0.49  -0.38
TAX      -0.44   0.54  -0.47
PTRATIO -0.18   0.37  -0.51
B         1.00  -0.37   0.33
LSTAT    -0.37   1.00  -0.74
PRICE     0.33  -0.74   1.00
SPEARMAN CORRELATION
          CRIM    ZN  INDUS  CHAS   NOX    RM   AGE   DIS   RAD   TAX  PTRATIO  \
CRIM      1.00 -0.57   0.74  0.04  0.82 -0.31  0.70 -0.74  0.73  0.73     0.47
ZN       -0.57  1.00  -0.64 -0.04 -0.63  0.36 -0.54  0.61 -0.28 -0.37    -0.45
INDUS     0.74 -0.64   1.00  0.09  0.79 -0.42  0.68 -0.76  0.46  0.66     0.43
CHAS      0.04 -0.04   0.09  1.00  0.07  0.06  0.07 -0.08  0.02 -0.04    -0.14
NOX       0.82 -0.63   0.79  0.07  1.00 -0.31  0.80 -0.88  0.59  0.65     0.39
RM       -0.31  0.36  -0.42  0.06 -0.31  1.00 -0.28  0.26 -0.11 -0.27    -0.31
AGE       0.70 -0.54   0.68  0.07  0.80 -0.28  1.00 -0.80  0.42  0.53     0.36
```

```
DIS     -0.74  0.61  -0.76 -0.08 -0.88  0.26 -0.80  1.00 -0.50 -0.57    -0.32
RAD      0.73 -0.28   0.46  0.02  0.59 -0.11  0.42 -0.50  1.00  0.70     0.32
TAX      0.73 -0.37   0.66 -0.04  0.65 -0.27  0.53 -0.57  0.70  1.00     0.45
PTRATIO  0.47 -0.45   0.43 -0.14  0.39 -0.31  0.36 -0.32  0.32  0.45     1.00
B       -0.36  0.16  -0.29 -0.04 -0.30  0.05 -0.23  0.25 -0.28 -0.33    -0.07
LSTAT    0.63 -0.49   0.64 -0.05  0.64 -0.64  0.66 -0.56  0.39  0.53     0.47
PRICE   -0.56  0.44  -0.58  0.14 -0.56  0.63 -0.55  0.45 -0.35 -0.56    -0.56

            B  LSTAT  PRICE
CRIM    -0.36   0.63  -0.56
ZN       0.16  -0.49   0.44
INDUS   -0.29   0.64  -0.58
CHAS    -0.04  -0.05   0.14
NOX     -0.30   0.64  -0.56
RM       0.05  -0.64   0.63
AGE     -0.23   0.66  -0.55
DIS      0.25  -0.56   0.45
RAD     -0.28   0.39  -0.35
TAX     -0.33   0.53  -0.56
PTRATIO -0.07   0.47  -0.56
B        1.00  -0.21   0.19
LSTAT   -0.21   1.00  -0.85
PRICE    0.19  -0.85   1.00
KENDALL CORRELATION
         CRIM    ZN  INDUS  CHAS   NOX    RM   AGE   DIS   RAD   TAX  PTRATIO  \
CRIM     1.00 -0.46   0.52  0.03  0.60 -0.21  0.50 -0.54  0.56  0.54     0.31
ZN      -0.46  1.00  -0.54 -0.04 -0.51  0.28 -0.43  0.48 -0.23 -0.29    -0.36
INDUS    0.52 -0.54   1.00  0.08  0.61 -0.29  0.49 -0.57  0.35  0.48     0.34
CHAS     0.03 -0.04   0.08  1.00  0.06  0.05  0.06 -0.07  0.02 -0.04    -0.12
NOX      0.60 -0.51   0.61  0.06  1.00 -0.22  0.59 -0.68  0.43  0.45     0.28
RM      -0.21  0.28  -0.29  0.05 -0.22  1.00 -0.19  0.18 -0.08 -0.19    -0.22
AGE      0.50 -0.43   0.49  0.06  0.59 -0.19  1.00 -0.61  0.31  0.36     0.25
DIS     -0.54  0.48  -0.57 -0.07 -0.68  0.18 -0.61  1.00 -0.36 -0.38    -0.22
RAD      0.56 -0.23   0.35  0.02  0.43 -0.08  0.31 -0.36  1.00  0.56     0.25
TAX      0.54 -0.29   0.48 -0.04  0.45 -0.19  0.36 -0.38  0.56  1.00     0.29
PTRATIO  0.31 -0.36   0.34 -0.12  0.28 -0.22  0.25 -0.22  0.25  0.29     1.00
B       -0.26  0.13  -0.19 -0.03 -0.20  0.03 -0.15  0.17 -0.21 -0.24    -0.04
LSTAT    0.45 -0.39   0.47 -0.04  0.45 -0.47  0.49 -0.41  0.29  0.38     0.33
PRICE   -0.40  0.34  -0.42  0.12 -0.39  0.48 -0.39  0.31 -0.25 -0.41    -0.40

            B  LSTAT  PRICE
CRIM    -0.26   0.45  -0.40
ZN       0.13  -0.39   0.34
INDUS   -0.19   0.47  -0.42
CHAS    -0.03  -0.04   0.12
NOX     -0.20   0.45  -0.39
RM       0.03  -0.47   0.48
AGE     -0.15   0.49  -0.39
```

```
DIS       0.17  -0.41   0.31
RAD      -0.21   0.29  -0.25
TAX      -0.24   0.38  -0.41
PTRATIO -0.04   0.33  -0.40
B         1.00  -0.15   0.13
LSTAT    -0.15   1.00  -0.67
PRICE     0.13  -0.67   1.00
```

Missing Values

Sometimes, in a dataset we will have missing values such as NaN or empty string in a cell. We need to take care of these missing values so that our machine learning model doesn't break. To handle missing values, there are three approaches followed.

Replace the missing value with a large negative number (e.g. -999). Replace the missing value with mean of the column. Replace the missing value with median of the column.

[20]:
```
#returns a boolean for each column in the dataset that tells if the column␣
 ↪contains any missing value.
print(pd.isnull(df).any())
```

```
CRIM       False
ZN         False
INDUS      False
CHAS       False
NOX        False
RM         False
AGE        False
DIS        False
RAD        False
TAX        False
PTRATIO    False
B          False
LSTAT      False
PRICE      False
dtype: bool
```

[21]:
```
#the above processes can also be implemented as below:
file_report = "plots/boston_housing.txt"
with open(file_report, "w") as f:
        f.write("Features shape : {}".format(df.drop("PRICE", axis=1).shape))
        f.write("\n")

        f.write("Target shape   : {}".format(df["PRICE"].shape))
        f.write("\n")

        f.write("\nColumn names")
        f.write("\n")
        f.write(str(df.columns))
        f.write("\n")

        f.write("\nStatistical summary")
        f.write("\n")
        f.write(str(df.describe()))
        f.write("\n")

        f.write("\nDatatypes")
        f.write("\n")
        f.write(str(df.dtypes))
        f.write("\n")

        f.write("\nPEARSON correlation")
        f.write("\n")
        f.write(str(df.corr(method="pearson")))
        f.write("\n")

        f.write("\nSPEARMAN correlation")
        f.write("\n")
        f.write(str(df.corr(method="spearman")))
        f.write("\n")

        f.write("\nKENDALL correlation")
        f.write("\n")
        f.write(str(df.corr(method="kendall")))
        f.write("\nMissing Values")
        f.write("\n")
        f.write(str(pd.isnull(df).any()))
```

4 Visualize the dataset

two types of visualization strategy namely univariate plots and bivariate plots.
As the name suggests, univariate plot is used to visualize a single column or an attribute whereas bivariate plot is used to visualize two columns or two attributes.

[22]:
```
#A box-whisker plot is a univariate plot used to visualize a data distribution.
# visualize the dataset
def warn(*args, **kwargs):
    pass


warnings.warn = warn
warnings.filterwarnings("ignore", category=FutureWarning)
```

[23]:
```
sns.set(color_codes=True)
colors = ["y", "b", "g", "r"]
```

```
cols = list(df.columns.values)
```

[25]:
```
#Density plot is another univariate plot that draws a histogram of the data␣
↪distribution
#and fits a Kernel Density Estimate (KDE).
if not os.path.exists("plots/univariate/box"):
    os.makedirs("plots/univariate/box")
```

[26]:
```
# draw a histogram and fit a kernel density estimate (KDE)
for i, col in enumerate(cols):
    sns.distplot(df[col], color=random.choice(colors))
    plt.savefig("plots/univariate/density/density_" + str(i) + ".png")
    plt.clf()
    plt.close()
```

[27]:
```
#multivariate plots
if not os.path.exists("plots/multivariate"):
    os.makedirs("plots/multivariate")

# bivariate plot between target and reason of absence
for i, col in enumerate(cols):
    if (i == len(cols) - 1):
        pass
    else:
        sns.jointplot(x=col, y="PRICE", data=df);
        plt.savefig("plots/multivariate/target_vs_" + str(i) + ".png")
        plt.clf()
        plt.close()
```

[28]:
```
#Scatter plot is used to understand relationship between two different␣
↪attributes in the dataset.
#pairplot-For each pair of features (columns) in the dataset, we can visualize␣
↪the scatter plot for each pair
#long with the feature’s histogram along the diagonal in a single image using␣
↪sns.pairplot() function.
# pairplot
sns.pairplot(df)
plt.savefig("plots/pairplot.png")
plt.clf()
plt.close()
```

We see a lot of structure in this dataset with outliers and different data distributions. Two key take aways from these visualizations are

Data is not standardized (meaning there are different data distributions). Data is not normalized (meaning there are differing scales of data).

4.1 Training regression models

By looking at the dataset, we simply can't suggest the best regression model for this problem. So, we will try out different regression models available in scikit-learn with a 10-fold cross validation method.

It means we split the training data into train and test data using a test_size parameter for 10-folds. Each fold will have different samples that are not present in other folds. By this way, we can throughly train our model on different samples in the dataset.

Before doing anything, we will split our boston housing prices dataframe df into features X and target Y.

[29]:
```
X = df.drop("PRICE", axis=1)
Y = df["PRICE"]
print(X.shape)
print(Y.shape)
```

```
(506, 13)
(506,)
```

As we see different data distributions, we will standardize the dataset using StandardScaler function in scikit-learn. This is a useful technique where the attributes are transformed to a standard gaussian distribution with a mean of 0 and a standard deviation of 1.

[38]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler = MinMaxScaler().fit(X)
scaled_X = scaler.transform(X)
```

[39]:
```
#Now, we will split the data into train and test set
from sklearn.model_selection import train_test_split

seed      = 9
test_size = 0.20

X_train, X_test, Y_train, Y_test = train_test_split(scaled_X, Y, test_size =␣
 ↪test_size, random_state = seed)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 13)
(102, 13)
(404,)
(102,)
```

Let's dive into regression. We will use different regression models offered by scikit-learn to produce a baseline accuracy for this problem. We will use the MSE (Mean Squared Error) as the performance metric for the regression models.

[40]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import mean_squared_error
from sklearn.utils import shuffle
```

[41]:
```
# user variables to tune
folds   = 10
metric  = "neg_mean_squared_error"

# hold different regression models in a single dictionary
models = {}
models["Linear"]        = LinearRegression()
models["Lasso"]         = Lasso()
models["ElasticNet"]    = ElasticNet()
models["KNN"]           = KNeighborsRegressor()
models["DecisionTree"]  = DecisionTreeRegressor()
models["SVR"]           = SVR()
models["AdaBoost"]      = AdaBoostRegressor()
models["GradientBoost"] = GradientBoostingRegressor()
models["RandomForest"]  = RandomForestRegressor()
models["ExtraTrees"]    = ExtraTreesRegressor()
# 10-fold cross validation for each model
```

[ ]:
```
# 10-fold cross validation for each model
model_results = []
model_names   = []
for model_name in models:
        model   = models[model_name]
        k_fold  = KFold(n_splits=folds, random_state=True, shuffle=True)
        results = cross_val_score(model, X_train, Y_train, cv=k_fold,␣
 ↪scoring=metric)

        model_results.append(results)
        model_names.append(model_name)
        print("{}: {}, {}".format(model_name, round(results.mean(), 3),␣
 ↪round(results.std(), 3)))
# box-whisker plot to compare regression models
```

```
figure = plt.figure()
figure.suptitle('Regression models comparison')
axis = figure.add_subplot(111)
plt.boxplot(model_results)
axis.set_xticklabels(model_names, rotation = 45, ha="right")
#mean squared deviation (MSD) of an estimator (of a procedure for estimating an␣
 ↪unobserved quantity)
#measures the average of the squares of the errors-that is,
#the average squared difference between the estimated values and the actual␣
 ↪value.
axis.set_ylabel("Mean Squared Error (MSE)")
plt.margins(0.05, 0.1)
plt.savefig("plots/model_mse_scores.png")
plt.clf()
plt.close()
```

```
Linear: -23.509, 10.778
Lasso: -63.674, 18.484
ElasticNet: -69.188, 19.888
KNN: -26.868, 12.775
DecisionTree: -22.076, 12.918
SVR: -34.825, 16.588
AdaBoost: -15.434, 3.389
GradientBoost: -9.802, 3.255
```

5 Choosing the best model

Based on the above comparison, we can see that Gradient Boosting Regression model outperforms all the other regression models. So, we will choose it as the best regression model for this problem

[ ]:
```
# create and fit the best regression model
best_model = GradientBoostingRegressor(random_state=seed)
best_model.fit(X_train, Y_train)

# make predictions using the model
predictions = best_model.predict(X_test)
print("[INFO] MSE : {}".format(round(mean_squared_error(Y_test, predictions),␣
 ↪3)))
```

Finally, we can see that Gradient Boosting Regression model achieved a mean squared error of 9.961 which means our model is able to predict correct values on test data with MSE of 9.961. We can visualize the predictions made by our best model and the original targets Y_test using the below code.

[ ]:
```
# plot between predictions and Y_test
x_axis = np.array(range(0, predictions.shape[0]))
```

```
plt.plot(x_axis, predictions, linestyle="--", marker="o", alpha=0.7, color='r',␣
 ↪label="predictions")
plt.plot(x_axis, Y_test, linestyle="--", marker="o", alpha=0.7, color='g',␣
 ↪label="Y_test")
plt.xlabel('Row number')
plt.ylabel('PRICE')
plt.title('Predictions vs Y_test')
plt.legend(loc='lower right')
plt.savefig("plots/predictions_vs_ytest.png")
plt.clf()
plt.close()
```

Feature Importance Once we have a trained model, we can understand feature importance (or variable importance) of the dataset which tells us how important each feature is, to predict the target.

[ ]:
```
# plot model's feature importance
feature_importance = best_model.feature_importances_
feature_importance = 100.0 * (feature_importance / feature_importance.max())

sorted_idx = np.argsort(feature_importance)
pos        = np.arange(sorted_idx.shape[0]) + .5

plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, dataset.feature_names[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.savefig("plots/feature_importance.png")
plt.clf()
plt.close()
```