05_DecisionTree

December 19, 2019

1 REGRESSION AND DECISION TREE

[0]:
```
print('HELLO')
```

```
HELLO
```

[0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20http
s%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.reado
nly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
ůůůůůůůůůů
Mounted at /content/drive
```

[0]:
```
cd /content/drive/My Drive/Colab Notebooks
```

```
/content/drive/My Drive/Colab Notebooks
```

[0]:
```
pwd
```

[0]: '/content/drive/My Drive/Colab Notebooks'

[0]:
```
ls
```

```
'Copy of TFModelOptimisationGeneric.ipynb'
ssd_inception_v2_coco_2018_01_28/
 housing.data                                   T3LAB/
 housing.ipynb
```

2 Boston Housing Dataset

The Boston data frame has 506 rows and 14 columns.This dataframe contains the following columns:

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 = nitrogen 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 mean of 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.
BLACK = 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
LSTAT = lower status of the population (percent).
price = median value of owner-occupied homes in $1000s
** Price is the TARGET variable **

[0]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[0]:
```
from sklearn.datasets import load_boston

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

[0]:
```
boston = load_boston()
```

[0]:
```
type(boston)
```

[0]:
```
sklearn.utils.Bunch
```

[0]:
```
boston.feature_names
```

[0]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

[0]:
```
data = boston.data
type(data)
```

[0]:
```
numpy.ndarray
```

[0]:
```
data.shape
```

[0]:
```
(506, 13)
```

[0]:
```
data = pd.DataFrame(data = data, columns= boston.feature_names)
data.head()
```

```
[0]:       CRIM    ZN  INDUS  CHAS    NOX  ...  RAD    TAX  PTRATIO       B  LSTAT
     0  0.00632  18.0   2.31   0.0  0.538  ...  1.0  296.0     15.3  396.90   4.98
     1  0.02731   0.0   7.07   0.0  0.469  ...  2.0  242.0     17.8  396.90   9.14
     2  0.02729   0.0   7.07   0.0  0.469  ...  2.0  242.0     17.8  392.83   4.03
     3  0.03237   0.0   2.18   0.0  0.458  ...  3.0  222.0     18.7  394.63   2.94
     4  0.06905   0.0   2.18   0.0  0.458  ...  3.0  222.0     18.7  396.90   5.33

     [5 rows x 13 columns]
```

```
[0]: boston.target
```

```
[0]: 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])
```

[0]:
```python
data['Price'] = boston.target
data.head()
```

[0]:
```
      CRIM    ZN  INDUS  CHAS    NOX  ...    TAX  PTRATIO       B  LSTAT  Price
0  0.00632  18.0   2.31   0.0  0.538  ...  296.0     15.3  396.90   4.98   24.0
1  0.02731   0.0   7.07   0.0  0.469  ...  242.0     17.8  396.90   9.14   21.6
2  0.02729   0.0   7.07   0.0  0.469  ...  242.0     17.8  392.83   4.03   34.7
3  0.03237   0.0   2.18   0.0  0.458  ...  222.0     18.7  394.63   2.94   33.4
4  0.06905   0.0   2.18   0.0  0.458  ...  222.0     18.7  396.90   5.33   36.2

[5 rows x 14 columns]
```

[0]:
```python
data.describe()
```

[0]:
```
             CRIM          ZN       INDUS  ...           B       LSTAT
Price
count  506.000000  506.000000  506.000000  ...  506.000000  506.000000
506.000000
mean     3.613524   11.363636   11.136779  ...  356.674032   12.653063
22.532806
std      8.601545   23.322453    6.860353  ...   91.294864    7.141062
9.197104
min      0.006320    0.000000    0.460000  ...    0.320000    1.730000
5.000000
25%      0.082045    0.000000    5.190000  ...  375.377500    6.950000
17.025000
50%      0.256510    0.000000    9.690000  ...  391.440000   11.360000
21.200000
75%      3.677083   12.500000   18.100000  ...  396.225000   16.955000
25.000000
max     88.976200  100.000000   27.740000  ...  396.900000   37.970000
50.000000

[8 rows x 14 columns]
```

[0]:
```python
data.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
Price      506 non-null float64
dtypes: float64(14)
memory usage: 55.5 KB
```

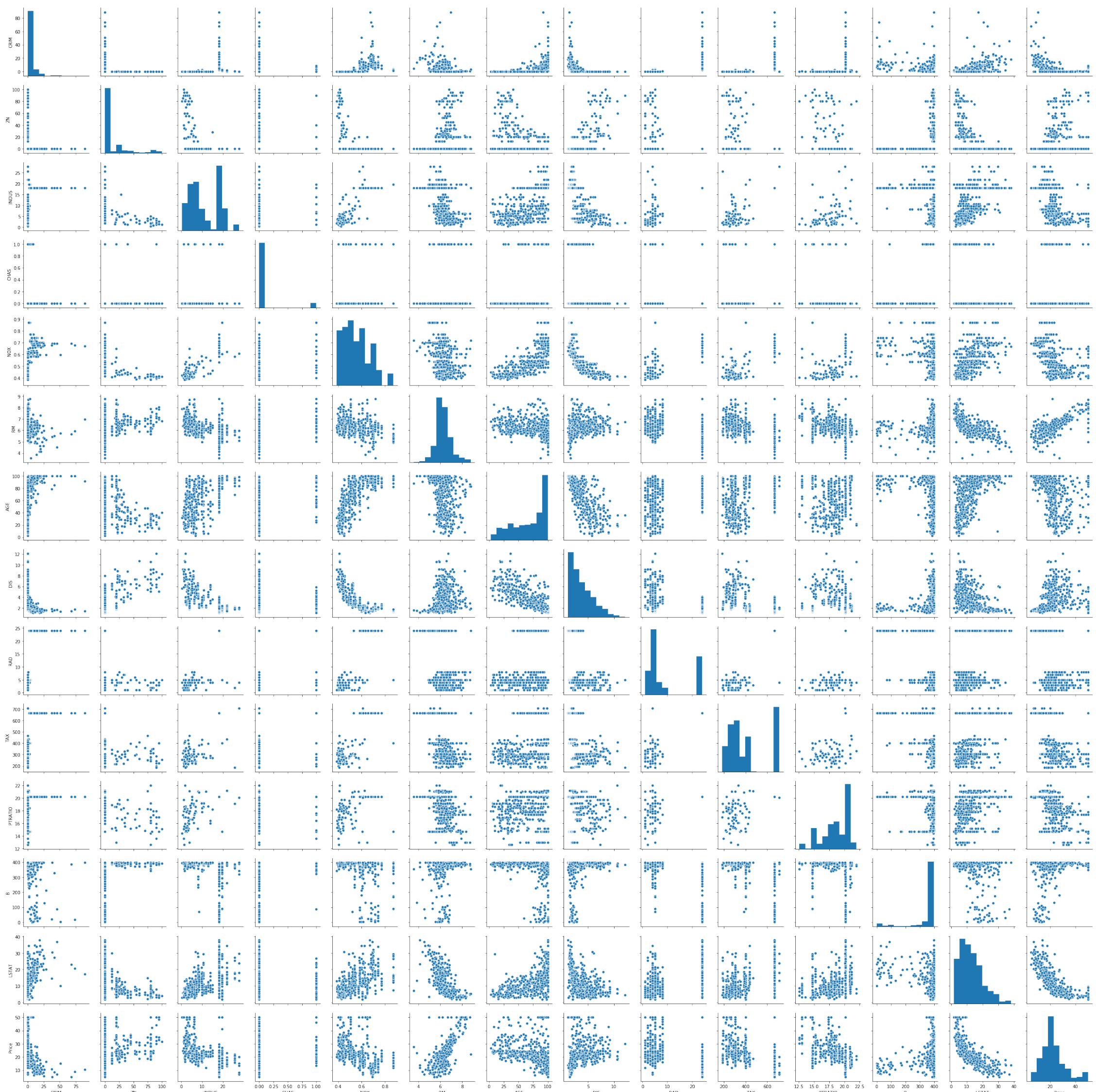
[0]:
```
data.isnull().sum()
```

[0]:
```
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
Price      0
dtype: int64
```

3 Data Visualization

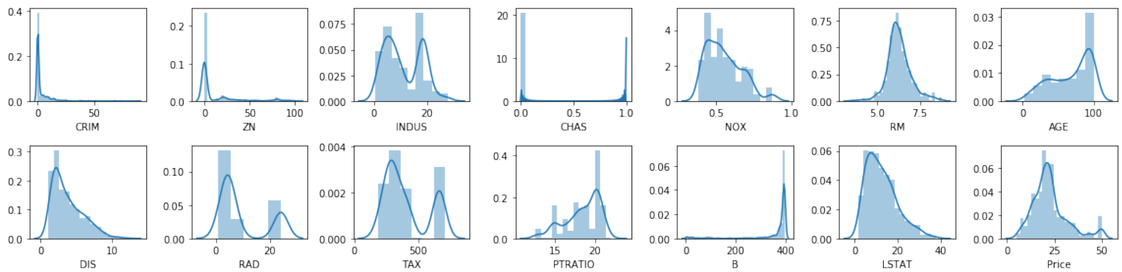
[0]:
```
sns.pairplot(data)
```

[0]:
```
<seaborn.axisgrid.PairGrid at 0x7fb467061cc0>
```



[0]:
```
rows = 2
cols = 7

fig, ax = plt.subplots(nrows= rows, ncols= cols, figsize = (16,4))

col = data.columns
index = 0

for i in range(rows):
    for j in range(cols):
        sns.distplot(data[col[index]], ax = ax[i][j])
        index = index + 1
```

```
plt.tight_layout()
```

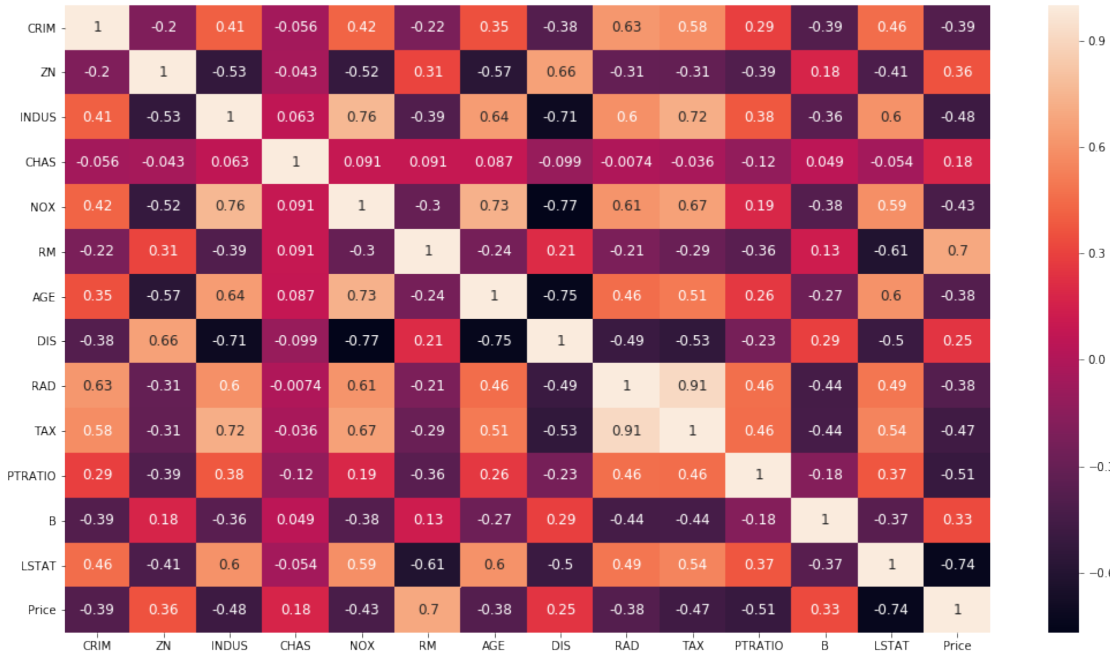


[0]:
```
corrmat = data.corr()
corrmat
```

[0]:
```
             CRIM        ZN     INDUS  ...         B     LSTAT     Price
CRIM     1.000000 -0.200469  0.406583  ... -0.385064  0.455621 -0.388305
ZN      -0.200469  1.000000 -0.533828  ...  0.175520 -0.412995  0.360445
INDUS    0.406583 -0.533828  1.000000  ... -0.356977  0.603800 -0.483725
CHAS    -0.055892 -0.042697  0.062938  ...  0.048788 -0.053929  0.175260
NOX      0.420972 -0.516604  0.763651  ... -0.380051  0.590879 -0.427321
RM      -0.219247  0.311991 -0.391676  ...  0.128069 -0.613808  0.695360
AGE      0.352734 -0.569537  0.644779  ... -0.273534  0.602339 -0.376955
DIS     -0.379670  0.664408 -0.708027  ...  0.291512 -0.496996  0.249929
RAD      0.625505 -0.311948  0.595129  ... -0.444413  0.488676 -0.381626
TAX      0.582764 -0.314563  0.720760  ... -0.441808  0.543993 -0.468536
PTRATIO  0.289946 -0.391679  0.383248  ... -0.177383  0.374044 -0.507787
B       -0.385064  0.175520 -0.356977  ...  1.000000 -0.366087  0.333461
LSTAT    0.455621 -0.412995  0.603800  ... -0.366087  1.000000 -0.737663
Price   -0.388305  0.360445 -0.483725  ...  0.333461 -0.737663  1.000000

[14 rows x 14 columns]
```

[0]:
```
fig, ax = plt.subplots(figsize = (18, 10))
sns.heatmap(corrmat, annot = True, annot_kws={'size': 12})
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb460109438>



[0]:
```
corrmat.index.values
```

[0]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT', 'Price'], dtype=object)
```

[0]:
```
def getCorrelatedFeature(corrdata, threshold):
    feature = []
    value = []

    for i, index in enumerate(corrdata.index):
        if abs(corrdata[index])> threshold:
            feature.append(index)
            value.append(corrdata[index])

    df = pd.DataFrame(data = value, index = feature, columns=['Corr Value'])
    return df
```

[0]:
```
threshold = 0.50
corr_value = getCorrelatedFeature(corrmat['Price'], threshold)
corr_value
```

[0]:
```
         Corr Value
RM         0.695360
PTRATIO   -0.507787
LSTAT     -0.737663
Price      1.000000
```

[0]:
```
corr_value.index.values
```

[0]: array(['RM', 'PTRATIO', 'LSTAT', 'Price'], dtype=object)

[0]:
```
correlated_data = data[corr_value.index]
correlated_data.head()
```

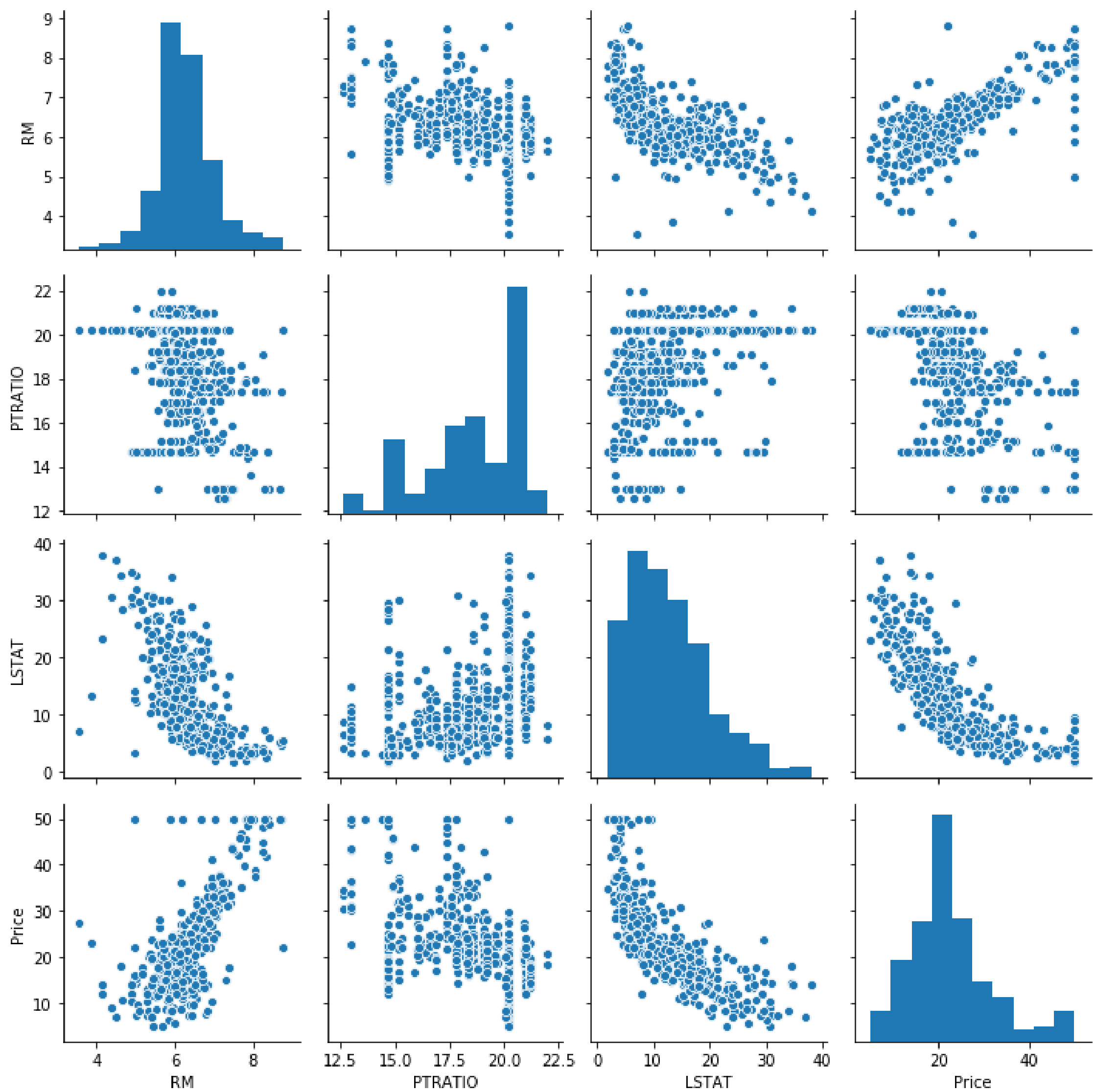
[0]:

	RM	PTRATIO	LSTAT	Price
0	6.575	15.3	4.98	24.0
1	6.421	17.8	9.14	21.6
2	7.185	17.8	4.03	34.7
3	6.998	18.7	2.94	33.4
4	7.147	18.7	5.33	36.2

[0]:
```
correlated_data.shape
```

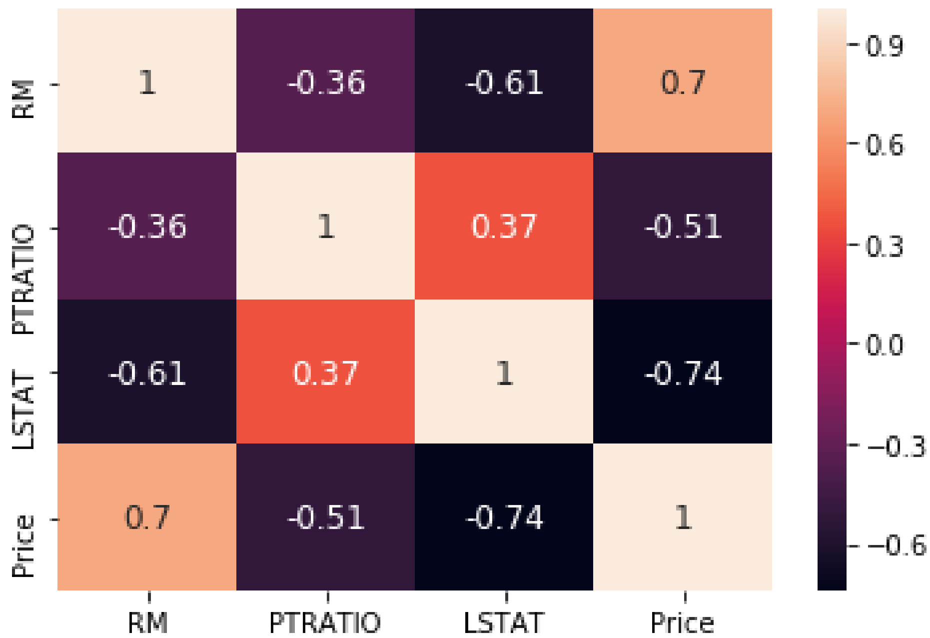
[0]: (506, 4)

[0]:
```
sns.pairplot(correlated_data)
plt.tight_layout()
```



```
[0]: sns.heatmap(correlated_data.corr(), annot=True, annot_kws={'size': 12})
```

[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb45c7dbd68>



3.1 Shuffle and Split Data

```
[0]: X = correlated_data.drop(labels=['Price'], axis = 1)
y = correlated_data['Price']
X.head()
```

[0]:

	RM	PTRATIO	LSTAT
0	6.575	15.3	4.98
1	6.421	17.8	9.14
2	7.185	17.8	4.03
3	6.998	18.7	2.94
4	7.147	18.7	5.33

```
[0]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 0)
```

```
[0]: X_train.shape, X_test.shape
```

[0]: ((404, 3), (102, 3))

3.2 Start train the model

[0]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

[0]:
```
y_predict = model.predict(X_test)
```

[0]:
```
df = pd.DataFrame(data = [y_predict, y_test])
df.T
```

[0]:
```
             0     1
0    27.609031  22.6
1    22.099034  50.0
2    26.529255  23.0
3    12.507986   8.3
4    22.254879  21.2
..         ...   ...
97   28.271228  24.7
98   18.467419  14.1
99   18.558070  18.7
100  24.681964  28.1
101  20.826879  19.8

[102 rows x 2 columns]
```

3.3 Defining performance metrics

It is difficult to measure the quality of a given model without quantifying its performance over training and testing. This is typically done using some type of performance metric, whether it is through calculating some type of error, the goodness of fit, or some other useful measurement. For this project, you will be calculating the coefficient of determination, R2, to quantify your model's performance. The coefficient of determination for a model is a useful statistic in regression analysis, as it often describes how "good" that model is at making predictions.

The values for R2 range from 0 to 1, which captures the percentage of squared correlation between the predicted and actual values of the target variable. A model with an R2 of 0 always fails to predict the target variable, whereas a model with an R2 of 1 perfectly predicts the target variable. Any value between 0 and 1 indicates what percentage of the target variable, using this model, can be explained by the features. A model can be given a negative R2 as well, which indicates that the model is no better than one that naively predicts the mean of the target variable.

For the performance_metric function in the code cell below, you will need to implement the following:

Use r2_score from sklearn.metrics to perform a performance calculation between y_true and y_predict. Assign the performance score to the score variable.

3.4 Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

MAE is the easiest to understand, because it's the average error. MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world. RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units. All of these are loss functions, because we want to minimize them.

[0]:
```
from sklearn.metrics import r2_score
```

[0]:
```
correlated_data.columns
```

[0]:
```
Index(['RM', 'PTRATIO', 'LSTAT', 'Price'], dtype='object')
```

[0]:
```
score = r2_score(y_test, y_predict)
mae = mean_absolute_error(y_test, y_predict)
mse = mean_squared_error(y_test, y_predict)

print('r2_score: ', score)
print('mae: ', mae)
print('mse: ', mse)
```

```
r2_score:  0.48816420156925067
mae:  4.404434993909257
mse:  41.67799012221683
```

[0]:
```
total_features = []
total_features_name = []
selected_correlation_value = []
r2_scores = []
mae_value = []
mse_value = []
```

[0]:
```
def performance_metrics(features, th, y_true, y_pred):
    score = r2_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)

    total_features.append(len(features)-1)
    total_features_name.append(str(features))
```

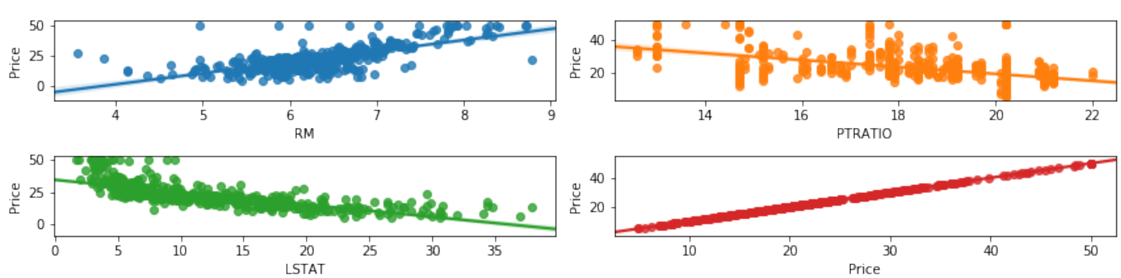
```
    selected_correlation_value.append(th)
    r2_scores.append(score)
    mae_value.append(mae)
    mse_value.append(mse)

    metrics_dataframe = pd.DataFrame(data= [total_features_name,␣
 ↪total_features, selected_correlation_value, r2_scores, mae_value, mse_value],
                                    index = ['features name', '#feature',␣
 ↪'corr_value', 'r2_score', 'MAE', 'MSE'])
    return metrics_dataframe.T
```

[0]:
```
performance_metrics(correlated_data.columns.values, threshold, y_test,␣
 ↪y_predict)
```

[0]:
```
                      features name #feature  ...      MAE     MSE
0  ['RM' 'PTRATIO' 'LSTAT' 'Price']        3  ...  4.40443  41.678

[1 rows x 6 columns]
```

3.5 regression plot of the features correlated with the price

[0]:
```
rows = 2
cols = 2
fig, ax = plt.subplots(nrows=rows, ncols=cols, figsize = (12, 3))

col = correlated_data.columns
index = 0

for i in range(rows):
    for j in range(cols):
        sns.regplot(x = correlated_data[col[index]], y =␣
 ↪correlated_data['Price'], ax = ax[i][j])
        index = index + 1
fig.tight_layout()
```



3.5.1 Let's find out other combination of columns to get better accuracy >60%

[0]:
```
corrmat['Price']
```

[0]:
```
CRIM      -0.388305
ZN         0.360445
INDUS     -0.483725
CHAS       0.175260
NOX       -0.427321
RM         0.695360
AGE       -0.376955
DIS        0.249929
RAD       -0.381626
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.737663
Price      1.000000
Name: Price, dtype: float64
```

[0]:
```
threshold = 0.60
corr_value = getCorrelatedFeature(corrmat['Price'], threshold)
corr_value
```

[0]:
```
       Corr Value
RM       0.695360
LSTAT   -0.737663
Price    1.000000
```

[0]:
```
correlated_data = data[corr_value.index]
correlated_data.head()
```

[0]:

	RM	LSTAT	Price
0	6.575	4.98	24.0
1	6.421	9.14	21.6
2	7.185	4.03	34.7
3	6.998	2.94	33.4
4	7.147	5.33	36.2

[0]:
```
def get_y_predict(corr_data):
    X = corr_data.drop(labels = ['Price'], axis = 1)
    y = corr_data['Price']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 0)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_predict = model.predict(X_test)
    return y_predict
```

[0]:
```
y_predict = get_y_predict(correlated_data)
```

[0]:
```
performance_metrics(correlated_data.columns.values, threshold, y_test,
 ↪y_predict)
```

[0]:
```
                     features name #feature  ...      MAE      MSE
0  ['RM' 'PTRATIO' 'LSTAT' 'Price']        3  ...  4.40443   41.678
1            ['RM' 'LSTAT' 'Price']        2  ...  4.14244  37.3831

[2 rows x 6 columns]
```

Let's find out other combination of columns to get better accuracy >70%

[0]:
```
corrmat['Price']
```

[0]:
```
CRIM      -0.388305
ZN         0.360445
INDUS     -0.483725
CHAS       0.175260
NOX       -0.427321
RM         0.695360
AGE       -0.376955
DIS        0.249929
RAD       -0.381626
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.737663
Price      1.000000
Name: Price, dtype: float64
```

[0]:
```
threshold = 0.70
corr_value = getCorrelatedFeature(corrmat['Price'], threshold)
corr_value
```

[0]:
```
       Corr Value
LSTAT   -0.737663
Price    1.000000
```

[0]:
```
correlated_data = data[corr_value.index]
correlated_data.head()
```

[0]:
```
   LSTAT  Price
0   4.98   24.0
1   9.14   21.6
2   4.03   34.7
3   2.94   33.4
4   5.33   36.2
```

[0]:
```
y_predict = get_y_predict(correlated_data)
performance_metrics(correlated_data.columns.values, threshold, y_test,
 ↪y_predict)
```