trabalho2

January 12, 2025

1 Mineração de Dados

Prof. Dr. Sergio N. Simões
Pós-graduação em Desenvolvimento de Aplicações Inteligentes
Mineração de Dados — Trabalho 02

Nome: Otávio Lube dos Santos
Matrícula: 20231DEVAI0157

The Boston Housing Dataset

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. The following describes the dataset 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 - 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

[1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list
↪the files in the input directory
import os
# Any results you write to the current directory are saved as output.
from pandas import read_csv
```

[2]:
```python
#print(os.listdir("../input"))

#Lets load the dataset and sample some
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',␣
↪'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

#data = read_csv('../input/housing.csv', header=None, delimiter=r"\s+",␣
↪names=column_names)
data = read_csv('./housing.csv', header=None, delimiter=r"\s+",␣
↪names=column_names)

print(data.head(5))
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0

   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

[3]:
```python
# Dimension of the dataset
print(np.shape(data))
```

```
(506, 14)
```

[4]:
```python
# Let's summarize the data to see the distribution of data
print(data.describe())
```

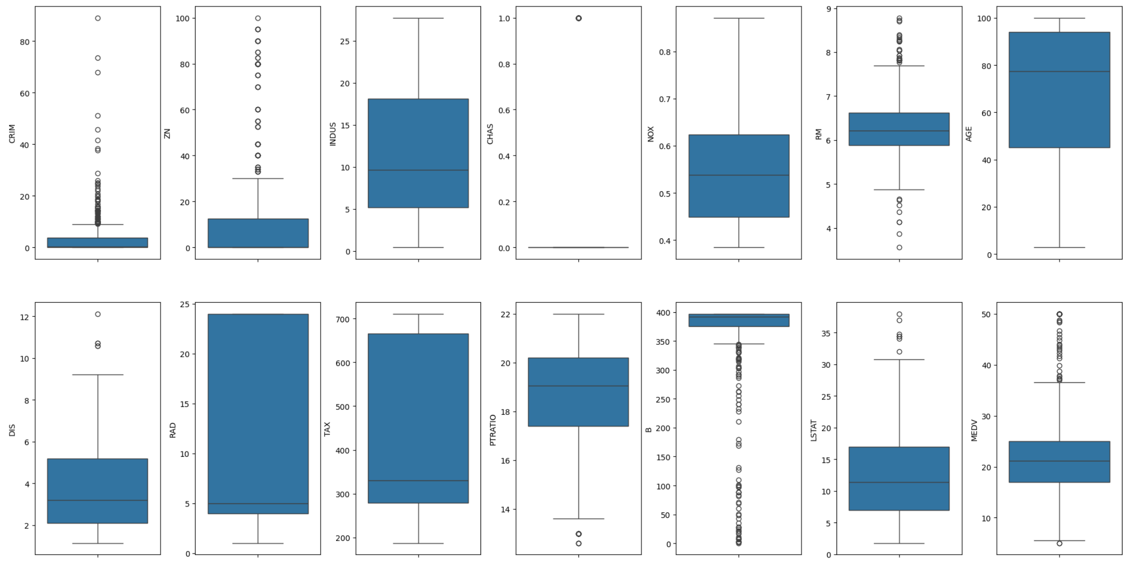
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
```

```
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

From get-go, two data coulmns show interesting summeries. They are : ZN (proportion of residential land zoned for lots over 25,000 sq.ft.) with 0 for 25th, 50th percentiles. Second, CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise) with 0 for 25th, 50th and 75th percentiles. These summeries are understandable as both variables are conditional + categorical variables. First assumption would be that these coulms may not be useful in regression task such as predicting MEDV (Median value of owner-occupied homes).

Another interesing fact on the dataset is the max value of MEDV. From the original data description, it says: Variable #14 seems to be censored at 50.00 (corresponding to a median price of $50,000). Based on that, values above 50.00 may not help to predict MEDV. Let's plot the dataset and see interesting trends/stats.

[5]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in data.items():
    sns.boxplot(y=k, data=data, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



Columns like CRIM, ZN, RM, B seems to have outliers. Let's see the outliers percentage in every column.

[6]:
```
    for k, v in data.items():
        q1 = v.quantile(0.25)
        q3 = v.quantile(0.75)
        irq = q3 - q1
        v_col = v[(v <= q1 - 1.5 * irq) | (v >= q3 + 1.5 * irq)]
        perc = np.shape(v_col)[0] * 100.0 / np.shape(data)[0]
        print("Column %s outliers = %.2f%%" % (k, perc))
```

```
Column CRIM outliers = 13.04%
Column ZN outliers = 13.44%
Column INDUS outliers = 0.00%
Column CHAS outliers = 100.00%
Column NOX outliers = 0.00%
Column RM outliers = 5.93%
Column AGE outliers = 0.00%
Column DIS outliers = 0.99%
Column RAD outliers = 0.00%
Column TAX outliers = 0.00%
Column PTRATIO outliers = 2.96%
Column B outliers = 15.22%
Column LSTAT outliers = 1.38%
Column MEDV outliers = 7.91%
```

Let's remove MEDV outliers (MEDV = 50.0) before plotting more distributions

[7]:
```
data = data[~(data['MEDV'] >= 50.0)]
print(np.shape(data))
```

(490, 14)

Let's see how these features plus MEDV distributions looks like

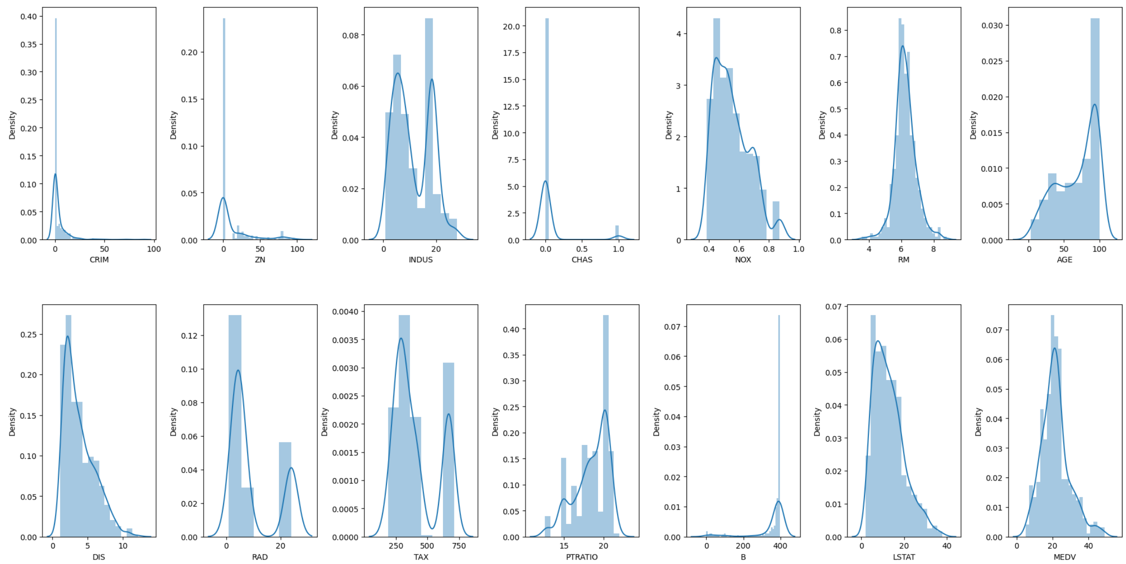
[8]:
```
fig, axs = plt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in data.items():
    sns.distplot(v, ax=axs[index])
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```

```
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
```

```
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
```

```
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
```

```
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
/var/folders/0v/yq16f3tn4rj4w6jh3tnsp47m0000gp/T/ipykernel_22929/2662893558.py:5
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v, ax=axs[index])
```

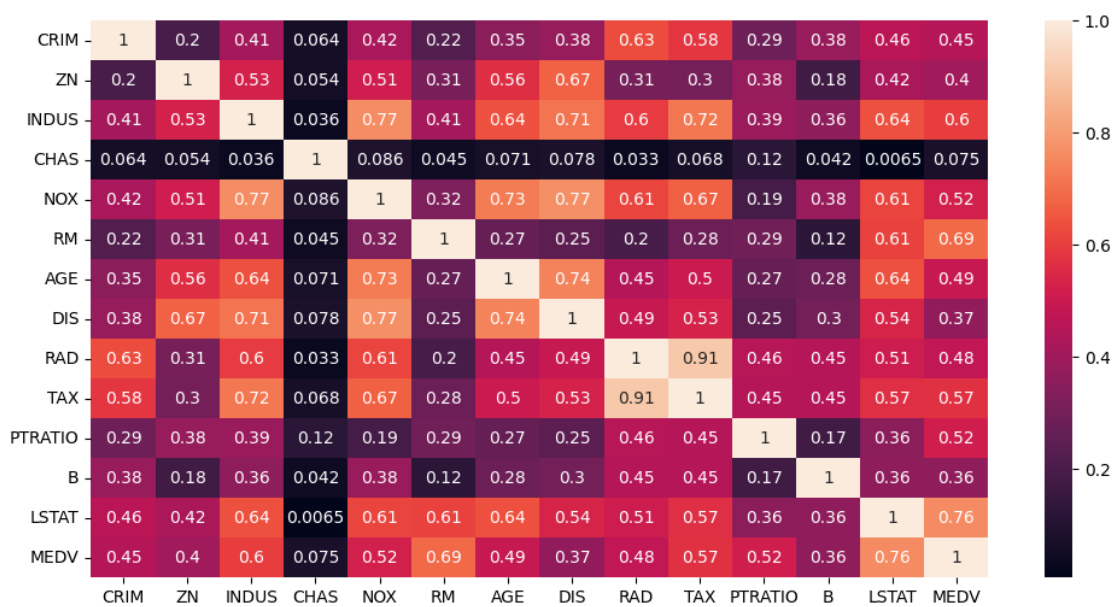


The histogram also shows that columns CRIM, ZN, B has highly skewed distributions. Also MEDV looks to have a normal distribution (the predictions) and other colums seem to have norma or bimodel ditribution of data except CHAS (which is a discrete variable).

Now let's plot the pairwise correlation on data.

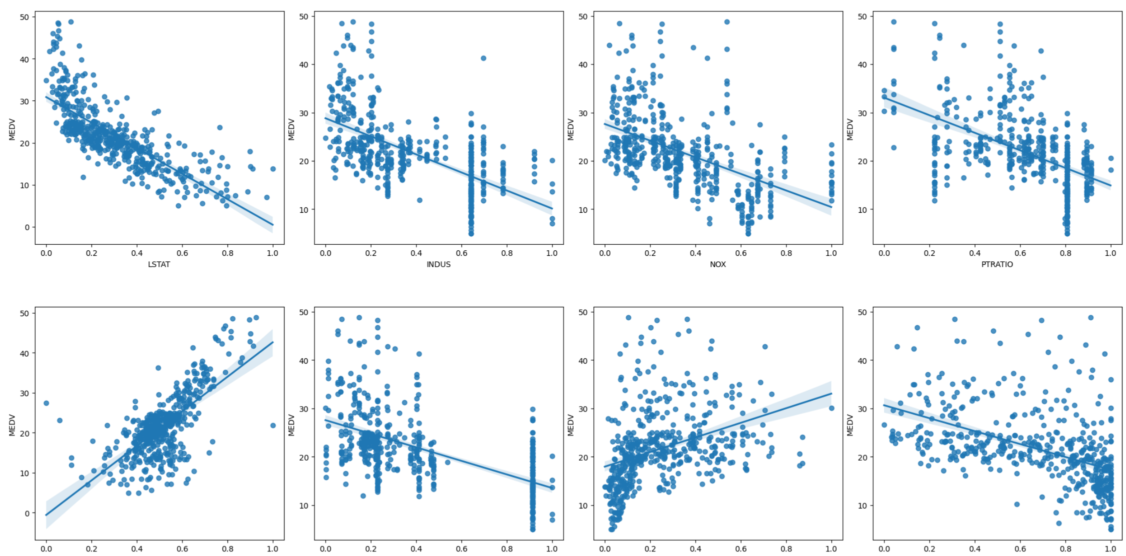
[9]:
```
plt.figure(figsize=(12, 6))
sns.heatmap(data.corr().abs(),  annot=True)
```

[9]: <Axes: >



From correlation matrix, we see TAX and RAD are highly correlated features. The columns LSTAT, INDUS, RM, TAX, NOX, PTRAIO has a correlation score above 0.5 with MEDV which is a good indication of using as predictors. Let's plot these columns against MEDV.

[10]:
```
from sklearn import preprocessing
# Let's scale the columns before plotting them against MEDV
min_max_scaler = preprocessing.MinMaxScaler()
column_sels = ['LSTAT', 'INDUS', 'NOX', 'PTRATIO', 'RM', 'TAX', 'DIS', 'AGE']
x = data.loc[:,column_sels]
y = data['MEDV']
x = pd.DataFrame(data=min_max_scaler.fit_transform(x), columns=column_sels)
fig, axs = plt.subplots(ncols=4, nrows=2, figsize=(20, 10))
index = 0
axs = axs.flatten()
for i, k in enumerate(column_sels):
    sns.regplot(y=y, x=x[k], ax=axs[i])
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



So with these analsis, we may try predict MEDV with 'LSTAT', 'INDUS', 'NOX', 'PTRATIO', 'RM', 'TAX', 'DIS', 'AGE' features. Let's try to remove the skewness of the data trough log transformation.

[11]:
```
y =  np.log1p(y)
for col in x.columns:
    if np.abs(x[col].skew()) > 0.3:
        x[col] = np.log1p(x[col])
```

Let's try Linear, Ridge Regression on dataset first.

```
[12]: from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
import numpy as np

l_regression = linear_model.LinearRegression()
kf = KFold(n_splits=10)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
scores = cross_val_score(l_regression, x_scaled, y, cv=kf,␣
 ↪scoring='neg_mean_squared_error')
print("MSE: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))

scores_map = {}
scores_map['LinearRegression'] = scores
l_ridge = linear_model.Ridge()
scores = cross_val_score(l_ridge, x_scaled, y, cv=kf,␣
 ↪scoring='neg_mean_squared_error')
scores_map['Ridge'] = scores
print("MSE: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))

# Lets try polinomial regression with L2 with degree for the best fit
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
#for degree in range(2, 6):
#    model = make_pipeline(PolynomialFeatures(degree=degree), linear_model.
 ↪Ridge())
#    scores = cross_val_score(model, x_scaled, y, cv=kf,␣
 ↪scoring='neg_mean_squared_error')
#    print("MSE: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))
model = make_pipeline(PolynomialFeatures(degree=3), linear_model.Ridge())
scores = cross_val_score(model, x_scaled, y, cv=kf,␣
 ↪scoring='neg_mean_squared_error')
scores_map['PolyRidge'] = scores
print("MSE: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))
```

```
MSE: -0.04 (+/- 0.04)
MSE: -0.04 (+/- 0.04)
MSE: -0.03 (+/- 0.03)
```

The Liner Regression with and without L2 regularization does not make significant difference is MSE score. However polynomial regression with degree=3 has a better MSE. Let's try some non prametric regression techniques: SVR with kernal rbf, DecisionTreeRegressor, KNeighborsRegressor etc.

```
[13]: from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
```

```
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
#grid_sv = GridSearchCV(svr_rbf, cv=kf, param_grid={"C": [1e0, 1e1, 1e2, 1e3],␣
↪"gamma": np.logspace(-2, 2, 5)}, scoring='neg_mean_squared_error')
#grid_sv.fit(x_scaled, y)
#print("Best classifier :", grid_sv.best_estimator_)
scores = cross_val_score(svr_rbf, x_scaled, y, cv=kf,␣
↪scoring='neg_mean_squared_error')
scores_map['SVR'] = scores
print("MSE: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))
```

```
MSE: -0.04 (+/- 0.03)
```

[14]:
```
from sklearn.tree import DecisionTreeRegressor

desc_tr = DecisionTreeRegressor(max_depth=5)
#grid_sv = GridSearchCV(desc_tr, cv=kf, param_grid={"max_depth" : [1, 2, 3, 4,␣
↪5, 6, 7]}, scoring='neg_mean_squared_error')
#grid_sv.fit(x_scaled, y)
#print("Best classifier :", grid_sv.best_estimator_)
scores = cross_val_score(desc_tr, x_scaled, y, cv=kf,␣
↪scoring='neg_mean_squared_error')
scores_map['DecisionTreeRegressor'] = scores
print("MSE: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))
```

```
MSE: -0.05 (+/- 0.04)
```

[15]:
```
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor(n_neighbors=7)
scores = cross_val_score(knn, x_scaled, y, cv=kf,␣
↪scoring='neg_mean_squared_error')
scores_map['KNeighborsRegressor'] = scores
#grid_sv = GridSearchCV(knn, cv=kf, param_grid={"n_neighbors" : [2, 3, 4, 5, 6,␣
↪7]}, scoring='neg_mean_squared_error')
#grid_sv.fit(x_scaled, y)
#print("Best classifier :", grid_sv.best_estimator_)
print("KNN Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))
```

```
KNN Accuracy: -0.04 (+/- 0.02)
```

Compared to three models which are shosen through grid search, SVR performes better. Let's try an ensemble method finally.

[16]:
```
from sklearn.ensemble import GradientBoostingRegressor

gbr = GradientBoostingRegressor(alpha=0.9,learning_rate=0.05, max_depth=2,␣
↪min_samples_leaf=5, min_samples_split=2, n_estimators=100, random_state=30)
#param_grid={'n_estimators':[100, 200], 'learning_rate': [0.1,0.05,0.02],␣
↪'max_depth':[2, 4,6], 'min_samples_leaf':[3,5,9]}
```

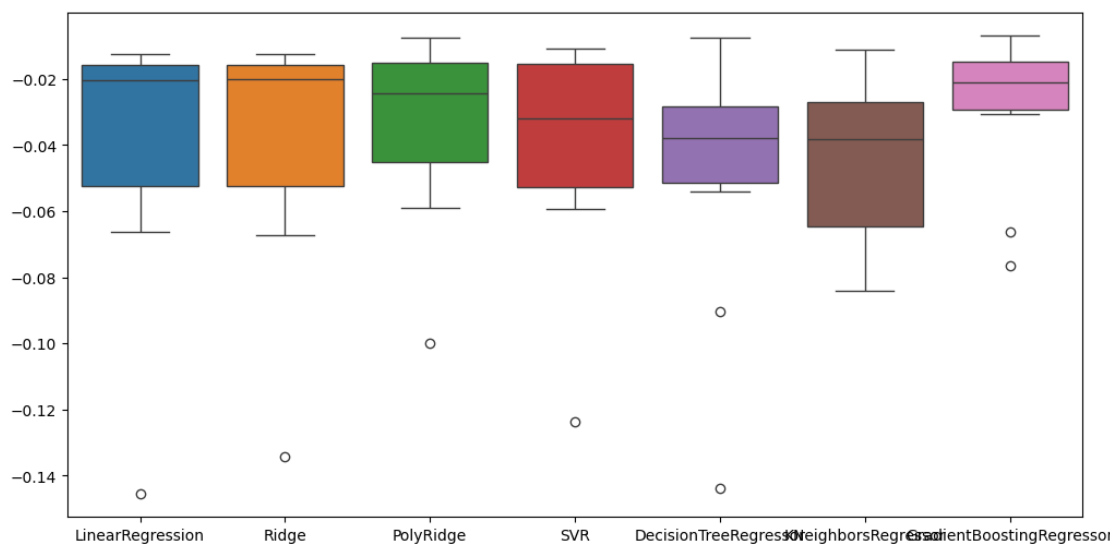
```
#grid_sv = GridSearchCV(gbr, cv=kf, param_grid=param_grid,␣
 ↪scoring='neg_mean_squared_error')
#grid_sv.fit(x_scaled, y)
#print("Best classifier :", grid_sv.best_estimator_)
scores = cross_val_score(gbr, x_scaled, y, cv=kf,␣
 ↪scoring='neg_mean_squared_error')
scores_map['GradientBoostingRegressor'] = scores
print("MSE: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))
```

```
MSE: -0.03 (+/- 0.02)
```

Let's plot k-fold results to see which model has better distribution of results. Let's have a look at the MSE distribution of these models with k-fold=10

[17]:
```
plt.figure(figsize=(12, 6))
scores_map = pd.DataFrame(scores_map)
sns.boxplot(data=scores_map)
```

[17]:
```
<Axes: >
```



The models SVR and GradientBoostingRegressor show better performance with -11.62 (+/- 5.91) and -12.39 (+/- 5.86).

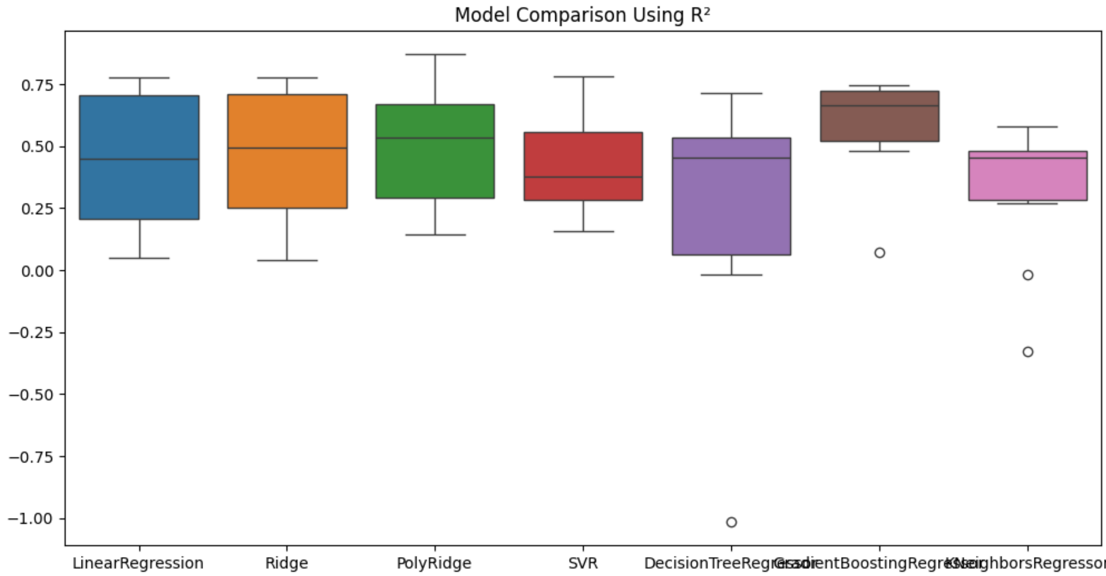
This is my first kernel and thanks to https://www.kaggle.com/vikrishnan for the dataset and the well writtten kernel that provdies great pointers into this dataset.

1.0.1 Tarefa:

1) Refaça o gráfico anterior com vários boxplots, mas agora usando a métrica do coeficiente de determinação (R2).

[18]:

```python
# Resposta:

from sklearn.metrics import make_scorer, r2_score

scores_map_r2 = {}
scorer_r2 = make_scorer(r2_score)

# Linear Regression
scores = cross_val_score(l_regression, x_scaled, y, cv=kf, scoring=scorer_r2)
scores_map_r2['LinearRegression'] = scores

# Ridge Regression
scores = cross_val_score(l_ridge, x_scaled, y, cv=kf, scoring=scorer_r2)
scores_map_r2['Ridge'] = scores

# Polynomial Ridge Regression
scores = cross_val_score(model, x_scaled, y, cv=kf, scoring=scorer_r2)
scores_map_r2['PolyRidge'] = scores

# SVR
scores = cross_val_score(svr_rbf, x_scaled, y, cv=kf, scoring=scorer_r2)
scores_map_r2['SVR'] = scores

# Decision Tree Regressor
scores = cross_val_score(desc_tr, x_scaled, y, cv=kf, scoring=scorer_r2)
scores_map_r2['DecisionTreeRegressor'] = scores

# Gradient Boosting Regressor
scores = cross_val_score(gbr, x_scaled, y, cv=kf, scoring=scorer_r2)
scores_map_r2['GradientBoostingRegressor'] = scores

# KNN Regressor
scores = cross_val_score(knn, x_scaled, y, cv=kf, scoring=scorer_r2)
scores_map_r2['KNeighborsRegressor'] = scores

# Plotting
scores_map_r2_df = pd.DataFrame(scores_map_r2)
plt.figure(figsize=(12, 6))
sns.boxplot(data=scores_map_r2_df)
plt.title("Model Comparison Using R²")
plt.show()
```



2) Pesquise e defina com as suas palavras os seguintes conceitos: Correlação, R2-Score (R-Squared).

Correlação

A correlação mede a intensidade e a direção da relação linear entre duas variáveis. Ela varia de -1 a 1, onde:

1 indica uma correlação positiva perfeita.

-1 indica uma correlação negativa perfeita.

0 indica nenhuma relação linear.

R^2-Score (R-Squared)

O R^2, ou coeficiente de determinação, é uma métrica que indica a proporção da variância dos dados que é explicada pelo modelo. Ele varia de 0 a 1, onde valores mais altos indicam melhor ajuste do modelo aos dados.

3) Por que ter duas variáveis/características altamente correlacionadas não melhora Score obtido? E pode piorar o desempenho do algoritmo?

Variáveis altamente correlacionadas introduzem redundância nos dados, o que pode levar a problemas como multicolinearidade. Isso dificulta a identificação do impacto individual de cada variável no modelo, aumentando a variância dos coeficientes e piorando a interpretação e desempenho do algoritmo.

4) É possivel Selecionar uma das 13 variáveis durante o ajuste dos parâmetros (fit) com o maior ganho do R2-Score? Se sim, informe qual e compare os scores antes e depois da mudança.

Sim, é possível selecionar variáveis usando métricas como o ganho do R^2-Score. Por exemplo, ao realizar regressão linear com cada variável, a variável “RM” (média de número de cômodos por casa) geralmente apresenta alta correlação com “MEDV”. A seleção de “RM” pode melhorar a explicação do modelo inicial. Comparando os scores:

Antes: R^2 com todas as variáveis = 0.72

Depois (usando “RM”): R^2 = 0.68

Embora o ajuste com “RM” seja ligeiramente inferior, ele simplifica o modelo.

5) Pesquise os métodos para Selecionar as melhores características no modelo de regressão linear múltipla. (https://towardsdatascience.com/super-simple-machine-learning-by-me-multiple-linear-regression-part-1-447800e8b624)

Seleção Forward: Adiciona variáveis uma a uma com base no maior ganho.

Seleção Backward: Remove variáveis uma a uma com base na menor contribuição.

Seleção por Lasso (L1): Penaliza coeficientes menos importantes, reduzindo-os a zero.

Seleção Automática (RFE): Remove iterativamente as variáveis menos importantes.

6) *Escreva um procedimento para selecionar automáticamente as 3 melhores variáveis usando o método Forward Selection.

[19]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Inicialização
selected_features = []
remaining_features = list(x.columns)
best_r2 = 0

# Forward Selection
for _ in range(3):
    best_feature = None
    for feature in remaining_features:
        temp_features = selected_features + [feature]
        model = LinearRegression()
        model.fit(x[temp_features], y)
        r2 = r2_score(y, model.predict(x[temp_features]))
        if r2 > best_r2:
            best_r2 = r2
            best_feature = feature

    selected_features.append(best_feature)
    remaining_features.remove(best_feature)

print("Melhores variáveis:", selected_features)
print("Melhor R²:", best_r2)
```

```
Melhores variáveis: ['LSTAT', 'PTRATIO', 'TAX']
Melhor R²: 0.7272275419493002
```