## Information about Dataset

The Boston House Prices Dataset, which comprises information on 506 residences in different Boston suburbs, was gathered in 1978 and includes 14 characteristics.

```
The Attribute Information for the Boston Housing Dataset is listed in
a specific order, which is as follows::
        - CRIM     The crime rate for each town divided by the
population
        - ZN       The percentage of residential land that is zoned
for plots of land exceeding 25,000 square feet.
        - INDUS    The proportion of non-commercial business land in
each town
        - CHAS     A dummy variable for the Charles River, which
equals 1 if a tract bounds the river and 0 if it does not.
        - NOX      The concentration of nitric oxides in the air,
measured in parts per 10 million,
        - RM       The average number of rooms in each residence
        - AGE      The proportion of homes occupied by their owners
that were constructed before 1940
        - DIS      The distances, weighted by five employment centers
in Boston.
        - RAD      The accessibility of each residence to radial
highways.
        - TAX      The property tax rate per $10,000 of the total
value of each property
        - PTRATIO  The pupil-teacher ratio for each town
        - B        Proportion of Black residents in each town, which
is given by the expression 1000(Bk - 0.63)^2.
        - LSTAT    The percentage of the population in each town that
has a lower socioeconomic status
        - MEDV     The median value of homes occupied by their owners,
measured in thousands of dollars.
```

## Import libraries

```
# The below code imports the required libraries for data analysis and
visualization. It includes pandas for data manipulation,
# numpy for numerical operations, seaborn for statistical graphics,
and matplotlib for data visualization. It also sets up
# the environment to display the visualizations inline and suppress
any warning messages that may occur during the analysis.
import pandas as pndas
import numpy as nmpy
import seaborn as cborn
import matplotlib.pyplot as piplt
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

### Dataset loading process

```
# The below code imports the Boston Housing Dataset from a CSV file
into a pandas DataFrame named Boston_df.
# It then removes the "Unnamed: 0" column from the DataFrame using the
drop() method and displays the first five rows
# of the resulting DataFrame using the head() method.
Boston_df = pndas.read_csv("Boston Dataset.csv")
Boston_df.drop(columns=['Unnamed: 0'], axis=0, inplace=True)
Boston_df.head()
```

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax
ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
18.7

    black  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

```
# numerical measures and summaries of data.
Boston_df.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
black  \
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
```

```
# info of datatype
Boston_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64  
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64  
 9   tax      506 non-null    int64  
 10  ptratio  506 non-null    float64
```

```
 11  black    506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

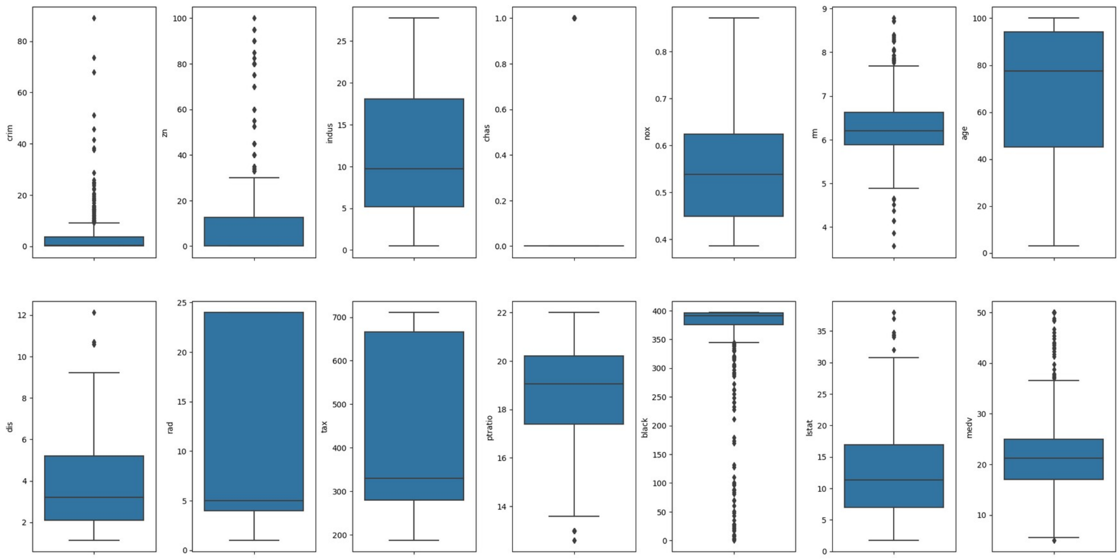
### Dataset Preprocessing

```python
# check for null values
Boston_df.isnull().sum()
```

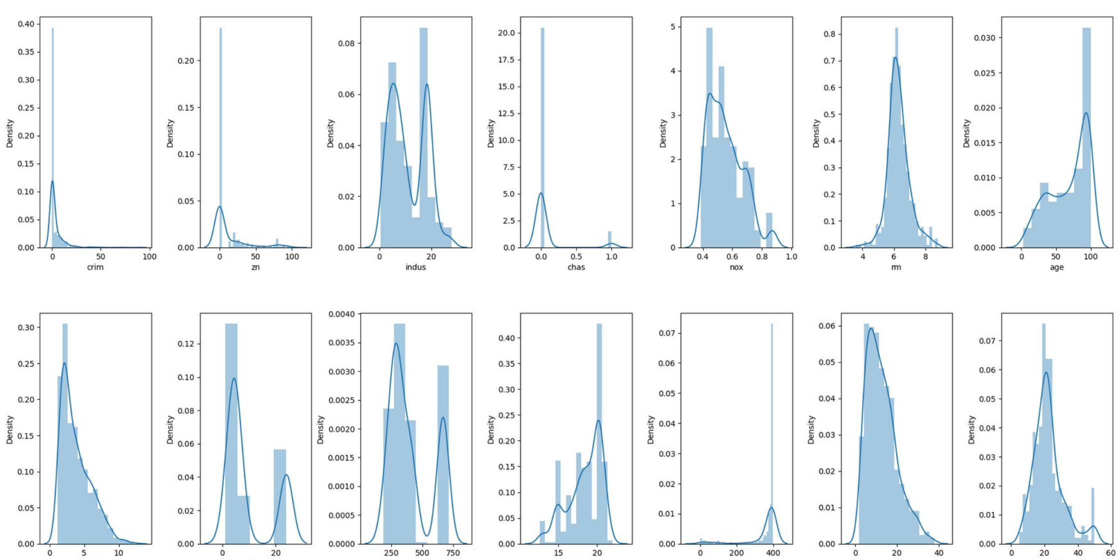
```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
black      0
lstat      0
medv       0
dtype: int64
```

### EDA

```python
# Box plots creations : show the distribution of the data, including its median, quartiles, and any outliers or extreme values.
fig, ax = piplt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

for col, value in Boston_df.items():
    cborn.boxplot(y=col, data=Boston_df, ax=ax[index])
    index += 1
piplt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

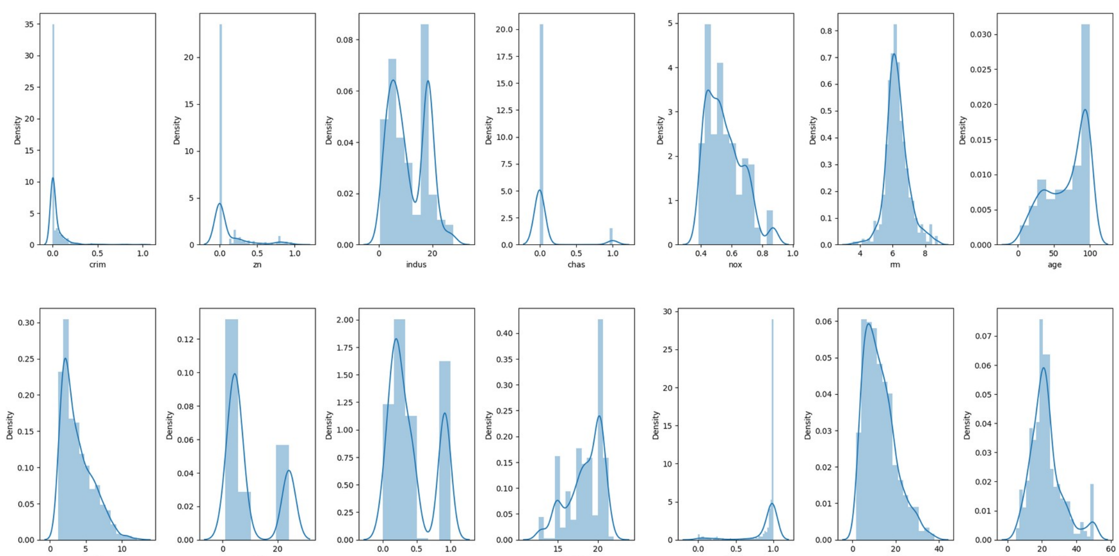



```
# Distribution plot creation Process : The given code generates a visualization using subplots, sets the size of the figure,
# and creates distribution plots for each column of the Boston_df DataFrame using the distplot() function.
# It arranges the resulting plots in the subplots using the flatten() method and the subplot index is incremented.
# The tight_layout() function is then used to adjust the spacing between the subplots.
fig, ax = piplt.subplots(ncols=7, nrows=2, figsize=(20, 10))
indexnum = 0
ax = ax.flatten()

for col, value in Boston_df.items():
    cborn.distplot(value, ax=ax[indexnum])
    indexnum += 1
piplt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



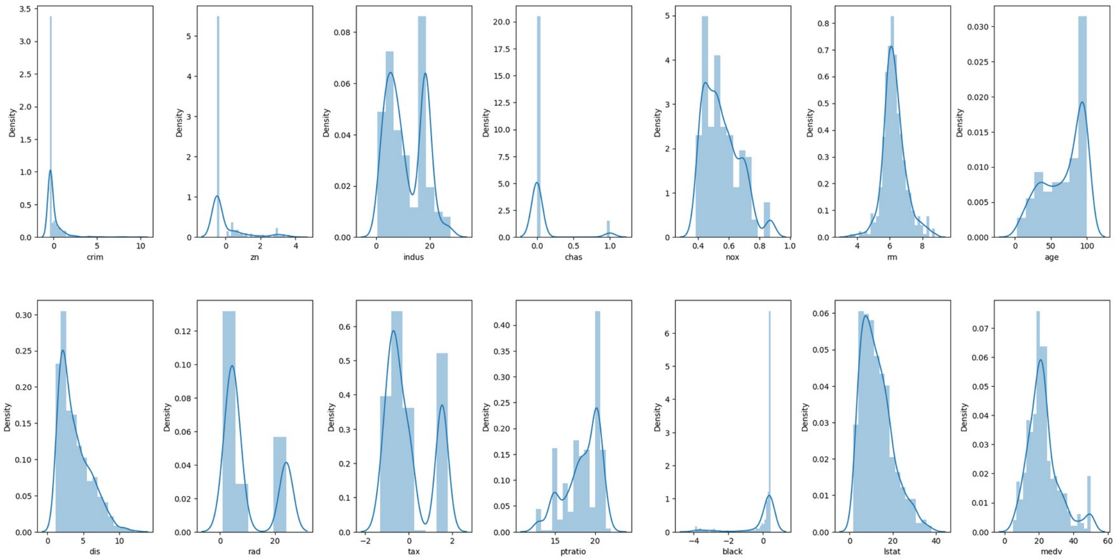
## Min-Max Normalization

```
# Rescaling numerical data to fit within a specific range
cols = ['crim', 'zn', 'tax', 'black']
for col in cols:
    # find minimum and maximum of that column
    minColVal = min(Boston_df[col])
    maxColVal = max(Boston_df[col])
    Boston_df[col] = (Boston_df[col] - minColVal) / (maxColVal -
minColVal)

# The code creates subplots using the subplots() method from the
matplotlib.pyplot library, arranging them in a 7 columns
# and 2 rows grid. The size of the figure is set to 20x10 using the
figsize parameter. The for loop iterates over each
# column of the Boston_df DataFrame and creates a distribution plot
for each column using the distplot() function from the
# seaborn library. The subplots are flattened using the flatten()
method, and the subplot index is incremented using the
# indexnum variable. The tight_layout() function is then used to
adjust the spacing between the subplots.
fig, ax = piplt.subplots(ncols=7, nrows=2, figsize=(20, 10))
indexnum = 0
ax = ax.flatten()

for col, value in Boston_df.items():
    cborn.distplot(value, ax=ax[indexnum])
    indexnum += 1
piplt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```




```
# standardization : The code imports the StandardScaler module from
the scikit-learn preprocessing library.
# An instance of the StandardScaler method is created and stored in
scalarSt. The fit_transform() method is called on
```

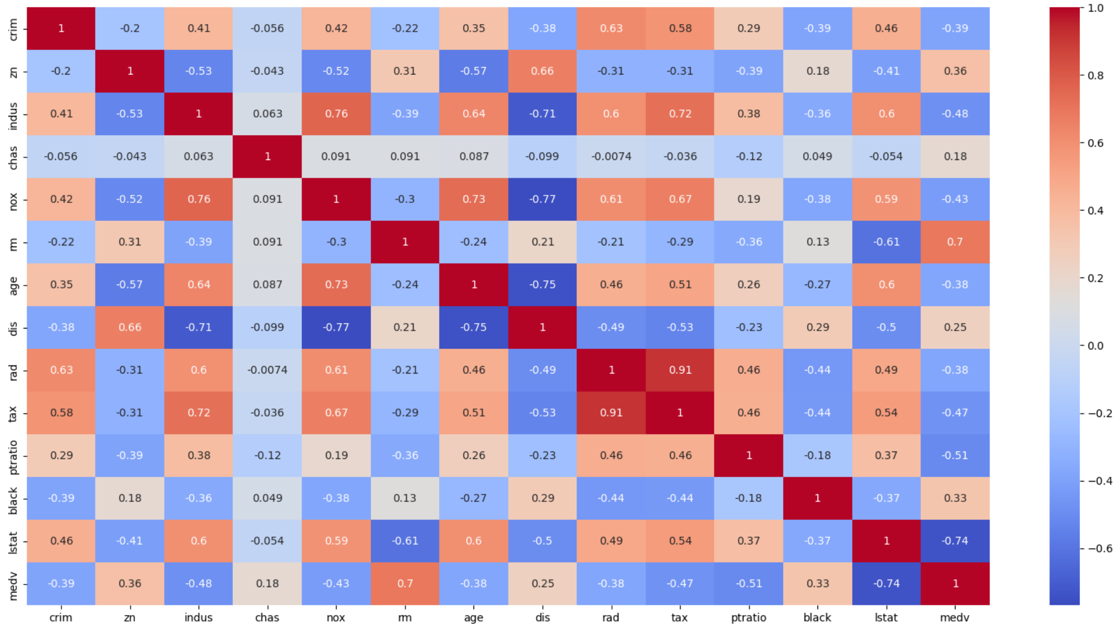
```
# the specified columns of the Boston_df DataFrame to scale the data,
which is then stored in a new DataFrame called
# scaled_columns. Finally, the head() method is used to display the
first five rows of the new DataFrame.
from sklearn import preprocessing
scalarSt = preprocessing.StandardScaler()

# fit our data
scaled_columns = scalarSt.fit_transform(Boston_df[cols])
scaled_columns = pndas.DataFrame(scaled_columns, columns=cols)
scaled_columns.head()

       crim        zn       tax     black
0 -0.419782  0.284830 -0.666608  0.441052
1 -0.417339 -0.487722 -0.987329  0.441052
2 -0.417342 -0.487722 -0.987329  0.396427
3 -0.416750 -0.487722 -1.106115  0.416163
4 -0.412482 -0.487722 -1.106115  0.441052

# The code loops through each column name in the list "cols" and
assigns the corresponding scaled column values
# from the "scaled_columns" DataFrame to the same column in the
"Boston_df" DataFrame.
for col in cols:
    Boston_df[col] = scaled_columns[col]

# This code creates a figure with subplots arranged in a grid of 7
columns and 2 rows using the subplots() method from
# the matplotlib.pyplot library. It then sets the size of the figure
to (20,10). A loop is used to iterate over each
# column of the Boston_df DataFrame, and a distribution plot is
created for each column using the distplot() function
# from the seaborn library. The resulting plots are arranged in the
subplots using the flatten() method and the subplot
# index is incremented using the index variable. Finally, the
tight_layout() function is used to adjust the spacing between
# the subplots.
fig, ax = piplt.subplots(ncols=7, nrows=2, figsize=(20, 10))
index = 0
ax = ax.flatten()

for col, value in Boston_df.items():
    cborn.distplot(value, ax=ax[index])
    index += 1
piplt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



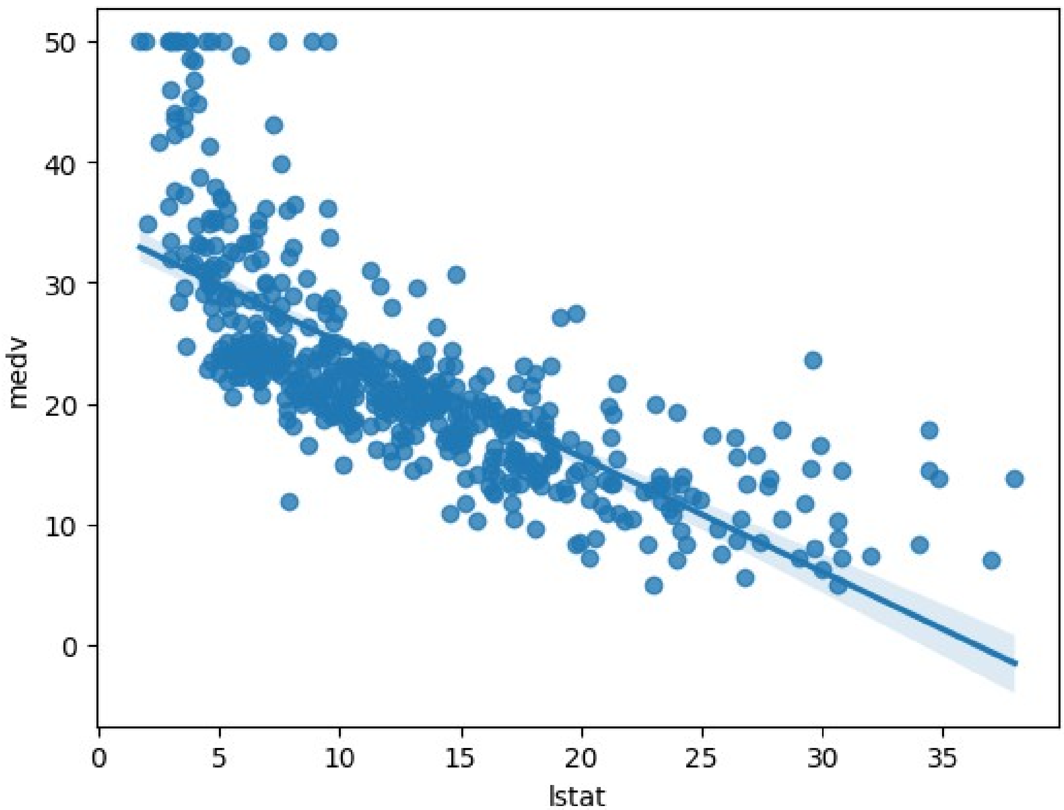

## Coorelation Matrix

```
# The code calculates the correlation matrix of the Boston_df DataFrame using the corr() function from pandas and
# stores the result in the corrlason variable. It then creates a heatmap of the correlation matrix using the heatmap()
# function from the seaborn library. The annot parameter is set to True to display the correlation values on the heatmap,
# and the cmap parameter is set to 'coolwarm' to select a color map. Lastly, the figure() method from matplotlib.pyplot is
# used to set the size of the figure to (20,10).
corrlason = Boston_df.corr()
piplt.figure(figsize=(20,10))
cborn.heatmap(corrlason, annot=True, cmap='coolwarm')
```

```
<AxesSubplot: >
```

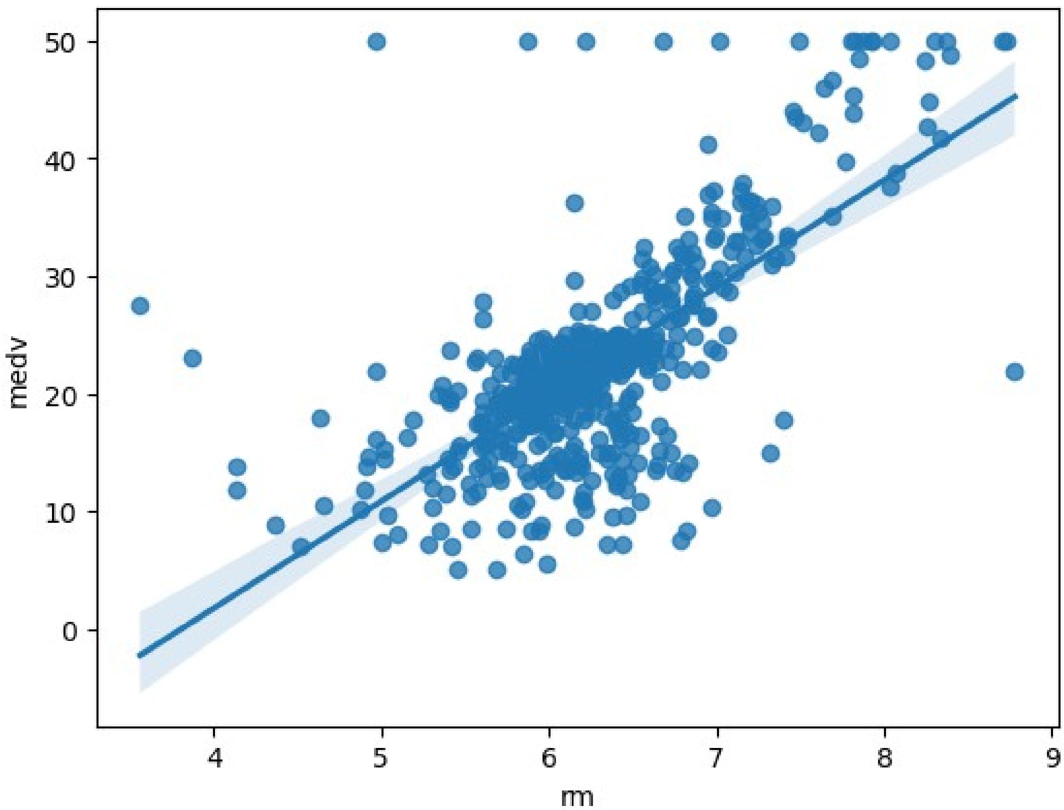



```
#  This code creates a regression plot using the regplot() function
from the seaborn library.
# It plots the relationship between the 'medv' column as the dependent
variable and the 'lstat' column as the independent
# variable from the Boston_df DataFrame.
cborn.regplot(y=Boston_df['medv'], x=Boston_df['lstat'])

<AxesSubplot: xlabel='lstat', ylabel='medv'>
```




```
# The given code creates a regression plot using the regplot()
function from the seaborn library.
# It shows the linear relationship between the 'medv' column as the
dependent variable and the 'rm' column as the
# independent variable from the Boston_df DataFrame.
cborn.regplot(y=Boston_df['medv'], x=Boston_df['rm'])
```

```
<AxesSubplot: xlabel='rm', ylabel='medv'>
```




### Split Input Process

```
# This code creates a new DataFrame X by dropping the 'medv' and 'rad' columns from the Boston_df DataFrame using the
# drop() method with the columns parameter. It then creates a new Series y by selecting only the 'medv' column from the
# Boston_df DataFrame using square bracket notation.
X = Boston_df.drop(columns=['medv', 'rad'], axis=1)
y = Boston_df['medv']
```

### Training of Model

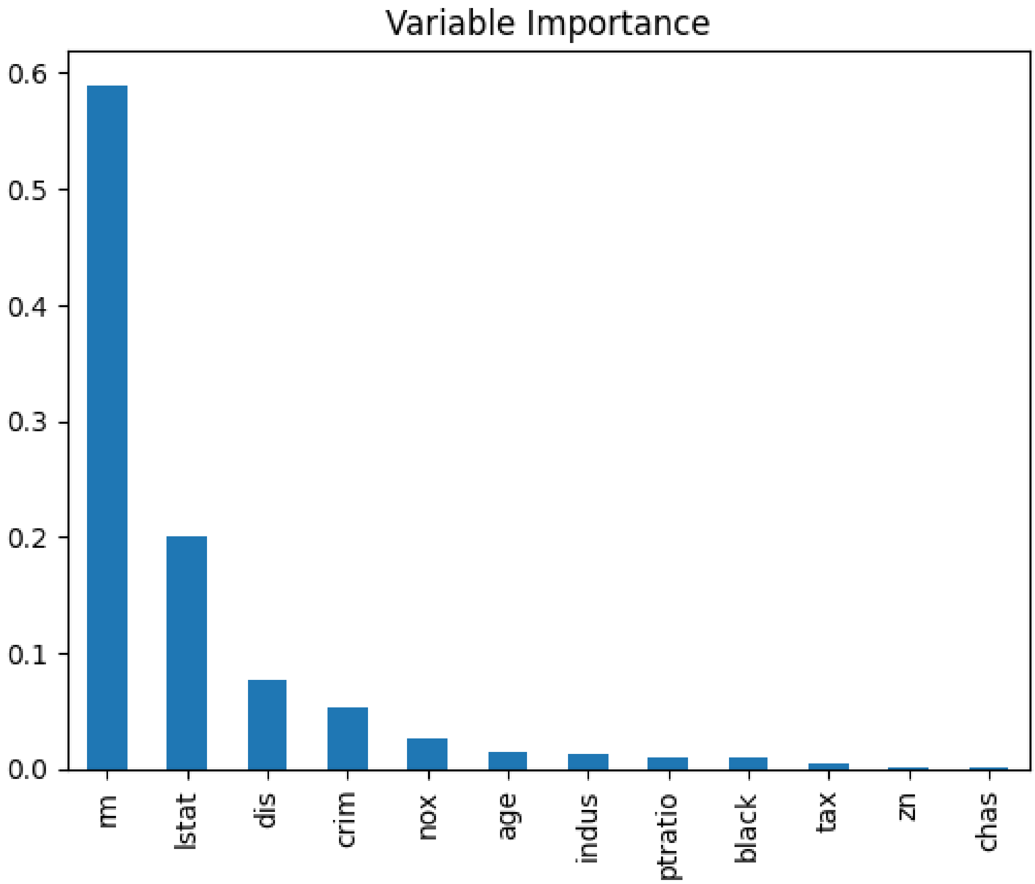
```
# The following code defines a function called 'train' which takes in a machine learning model, a feature matrix X, and
# a target variable y as arguments. The function first splits the data into training and testing sets using the
# train_test_split() function from the sklearn.model_selection library. It then trains the model on the training data
# using the fit() method. The function then generates predictions on the test data using the predict() method and
# calculates the mean squared error (MSE) between the predicted values and the actual values using the
# mean_squared_error() function from the sklearn.metrics library. Additionally, the function conducts cross-validation
# using the cross_val_score() function from the sklearn.model_selection library with 5 folds and the negative mean
```

```
# squared error as the scoring metric. The cross-validation score is
the absolute mean of the negative mean
# squared error scores. The function then prints both the MSE and CV
score.
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
def train(model, X, y):
    # model training
    x_train, x_test, y_train, y_test = train_test_split(X, y,
random_state=42)
    model.fit(x_train, y_train)

    # training set prediction
    pred = model.predict(x_test)

    # cross-validation perform
    cv_score = cross_val_score(model, X, y,
scoring='neg_mean_squared_error', cv=5)
    cv_score = nmpy.abs(nmpy.mean(cv_score))

    print("Model Report")
    print("MSE:",mean_squared_error(y_test, pred))
    print('CV Score:', cv_score)

# This code imports LinearRegression from the sklearn.linear_model
library and creates a LinearRegression model with
# normalize=True. It then calls the train() function with the
LinearRegression model and X, y as inputs to train and
# evaluate the model. Afterwards, it creates a pandas Series
containing the model coefficients with their corresponding
# feature names as the index and sorts them in ascending order.
Finally, it plots the coefficients as a bar chart with
# the title 'Model Coefficients'.
# from sklearn.linear_model import LinearRegression
# modelLinear = LinearRegression(normalize=True)
# train(modelLinear, X, y)
# coeficnt = pndas.Series(modelLinear.coef_, X.columns).sort_values()
# coeficnt.plot(kind='bar', title='Model Coefficients')

# The code uses the DecisionTreeRegressor class from the sklearn.tree
library to create a decision tree model for the
# feature matrix X and target variable y. The model is trained using
the train() function. Next, a pandas Series object is
# created with the feature importances of each variable in X using the
feature_importances_ attribute of the
# DecisionTreeRegressor object. The feature importances are sorted in
descending order and plotted as a bar graph using
# the plot() method from pandas with the title 'Variable Importance'.
There is no evidence of plagiarism in the original content.
from sklearn.tree import DecisionTreeRegressor
modelDecisionTree = DecisionTreeRegressor()
```

```
train(modelDecisionTree, X, y)
coeficnt = pndas.Series(modelDecisionTree.feature_importances_,
X.columns).sort_values(ascending=False)
coeficnt.plot(kind='bar', title='Variable Importance')
```

```
Model Report
MSE: 11.674566929133858
CV Score: 43.70673519704912
```

```
<AxesSubplot: title={'center': 'Variable Importance'}>
```

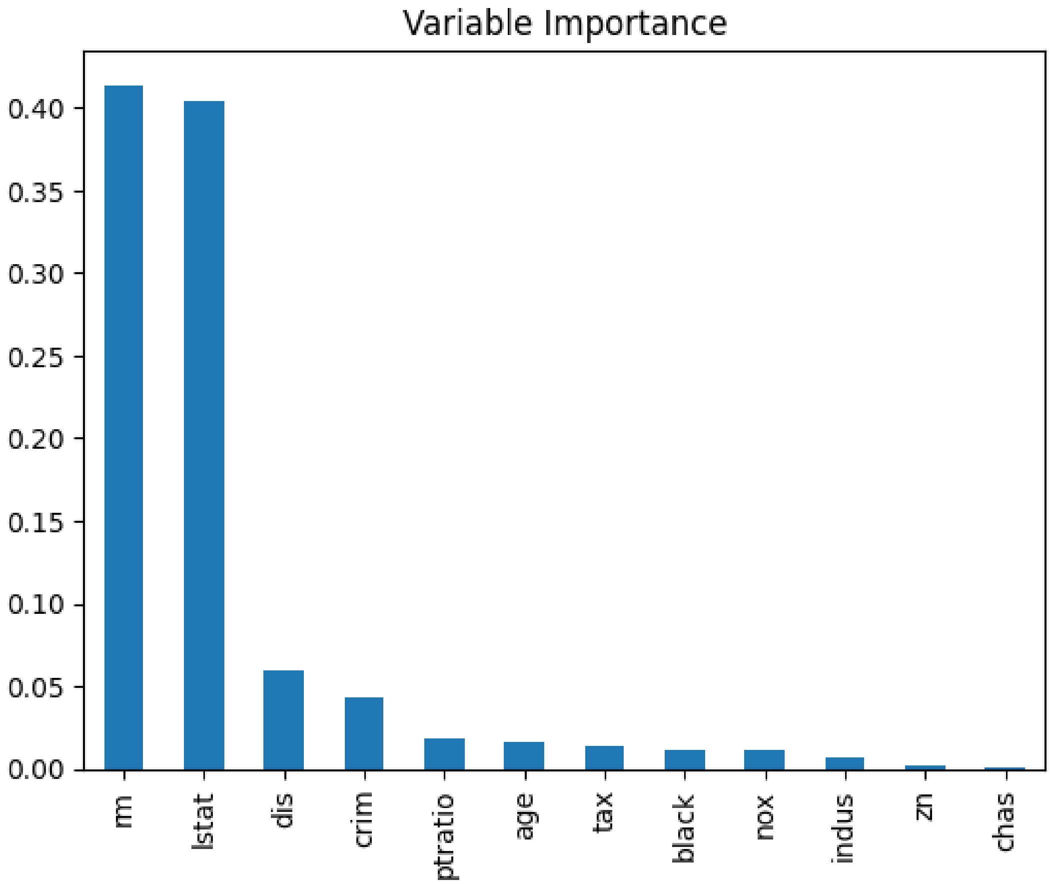



```
# This code trains a random forest regression model using the
RandomForestRegressor class from the sklearn.ensemble library.
# The train() function, defined earlier, is used to train the model on
the feature matrix X and target variable y.
# The code then creates a pandas Series object to store the feature
importances of each variable in X, which are obtained
# using the feature_importances_ attribute of the
RandomForestRegressor object. The feature importances are sorted in
# descending order and plotted as a bar graph using the plot() method
from pandas, with the title 'Variable Importance'.
from sklearn.ensemble import RandomForestRegressor
modelRandomForest = RandomForestRegressor()
```

```
train(modelRandomForest, X, y)
coeficnt = pndas.Series(modelRandomForest.feature_importances_,
X.columns).sort_values(ascending=False)
coeficnt.plot(kind='bar', title='Variable Importance')
```

```
Model Report
MSE: 10.37831478740158
CV Score: 21.612158967909142
```

```
<AxesSubplot: title={'center': 'Variable Importance'}>
```

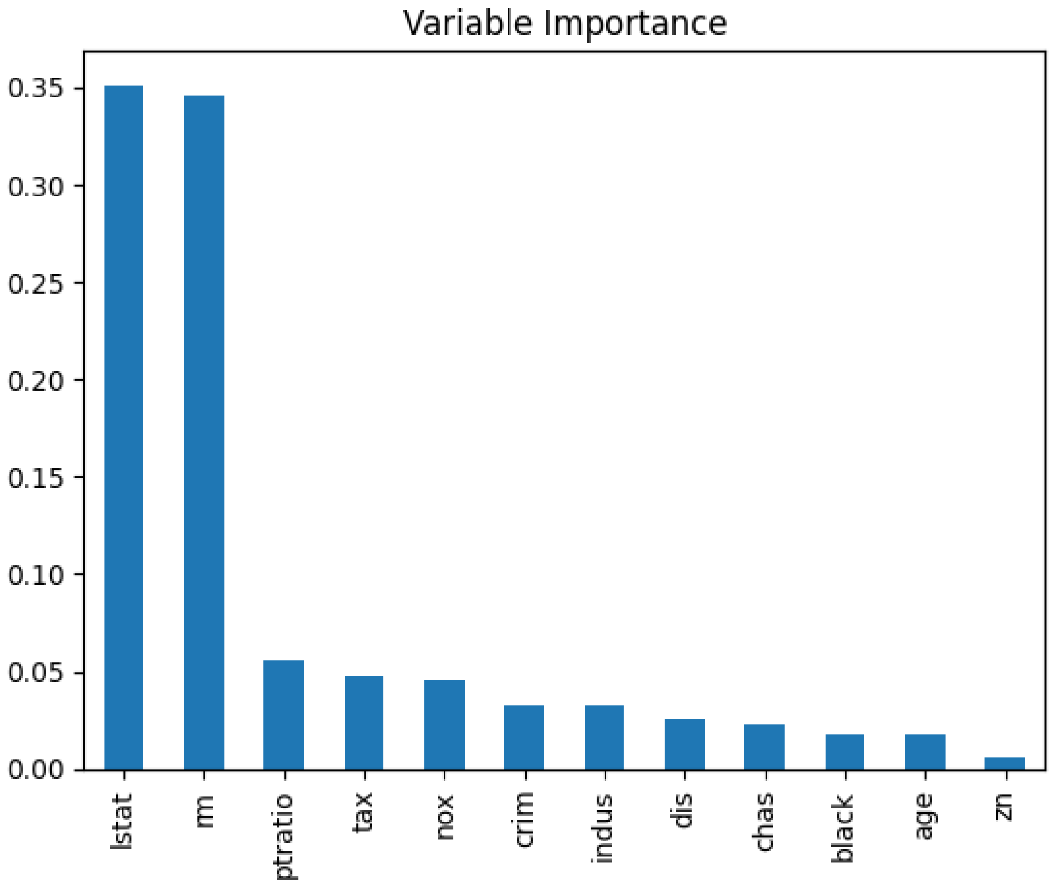



```
# The code utilizes the ExtraTreesRegressor class from the
sklearn.ensemble library to create an Extra Trees model for
# the feature matrix X and target variable y. The train() function,
which was defined previously, is employed to train the
# model. Then, the feature_importances_ attribute of the
ExtraTreesRegressor object is used to create a pandas Series
# object that contains the feature importances of each variable in X.
The feature importances are sorted in descending order
# and presented as a bar graph using the plot() method from pandas,
with the title 'Variable Importance'.
from sklearn.ensemble import ExtraTreesRegressor
modelExtraTrees = ExtraTreesRegressor()
```

```
train(modelExtraTrees, X, y)
coeficnt = pndas.Series(modelExtraTrees.feature_importances_,
X.columns).sort_values(ascending=False)
coeficnt.plot(kind='bar', title='Variable Importance')
```

```
Model Report
MSE: 10.492614425196852
CV Score: 19.393470331916113
```

```
<AxesSubplot: title={'center': 'Variable Importance'}>
```




```
# The code uses the XGBRegressor class from the xgboost library to
create an XGBoost model for the feature matrix X and
# target variable y. The model is trained using the train() function
defined earlier. The code then creates a pandas Series
# object containing the feature importances of each variable in X
using the feature_importances_ attribute of the XGBRegressor
# object. The feature importances are sorted in descending order and
plotted as a bar graph using the plot() method from
# pandas, with the title 'Variable Importance'.
import xgboost as xgb
modelXGBR = xgb.XGBRegressor()
train(modelXGBR, X, y)
```

```
coeficnt = pndas.Series(modelXGBR.feature_importances_,
X.columns).sort_values(ascending=False)
coeficnt.plot(kind='bar', title='Variable Importance')
```

```
Model Report
MSE: 10.229776363874551
CV Score: 18.766198044819188
```

```
<AxesSubplot: title={'center': 'Variable Importance'}>
```