House Price Prediction Model using Machine Learning by Navjoth Singh

December 25, 2023

1 Data Collection: Importing Libraries & File

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data = pd.read_csv("C:/Users/hp/OneDrive/Desktop/Data Science/Data for Practice/
↪HousingData.csv")
```

[3]:
```
from sklearn.model_selection import train_test_split
```

[4]:
```
from sklearn.linear_model import LinearRegression
```

[5]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

[6]:
```
data.head()
```

[6]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222     18.7

        B  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    NaN  36.2
```

[7]:
```
data.tail()
```

[7]:
```
        CRIM   ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
501  0.06263  0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273     21.0
```

```
502  0.04527  0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273      21.0
503  0.06076  0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273      21.0
504  0.10959  0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273      21.0
505  0.04741  0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273      21.0

          B  LSTAT  MEDV
501  391.99    NaN  22.4
502  396.90   9.08  20.6
503  396.90   5.64  23.9
504  393.45   6.48  22.0
505  396.90   7.88  11.9
```

[8]:
```
data.sample(5)
```

[8]:
```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  \
191      NaN  45.0   3.44   0.0  0.437  6.739  30.8  6.4798    5  398
358  5.20177   0.0  18.10   1.0  0.770  6.127  83.4  2.7227   24  666
129  0.88125   0.0  21.89   0.0  0.624  5.637  94.7  1.9799    4  437
88   0.05660   0.0   3.41   0.0  0.489  7.007  86.3  3.4217    2  270
106  0.17120   0.0   8.56   0.0  0.520  5.836  91.9  2.2110    5  384

     PTRATIO       B  LSTAT  MEDV
191     15.2  389.71   4.69  30.5
358     20.2  395.43  11.48  22.7
129     21.2  396.90  18.34  14.3
88      17.8  396.90   5.50  23.6
106     20.9  395.67  18.66  19.5
```

[9]:
```
data.shape
```

[9]:
```
(506, 14)
```

[10]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 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  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

[11]:
```
data.isnull().sum()
```

[11]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

[12]:
```
data.nunique()
```

[12]:
```
CRIM       484
ZN          26
INDUS       76
CHAS         2
NOX         81
RM         446
AGE        348
DIS        412
RAD          9
TAX         66
PTRATIO     46
B          357
LSTAT      438
MEDV       229
dtype: int64
```

[13]:
```
data['CRIM'].fillna(data['CRIM'].mean(),inplace=True)
data['AGE'].fillna(data['AGE'].mean(),inplace=True)
```

[14]:
```
data['ZN'].fillna(data['ZN'].mean(),inplace=True)
data['INDUS'].fillna(data['INDUS'].mean(),inplace=True)
data['CHAS'].fillna(data['CHAS'].mean(),inplace=True)
data['LSTAT'].fillna(data['LSTAT'].mean(),inplace=True)
```

[15]:
```
data.isnull().sum()
```

[15]:
```
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
MEDV       0
dtype: int64
```

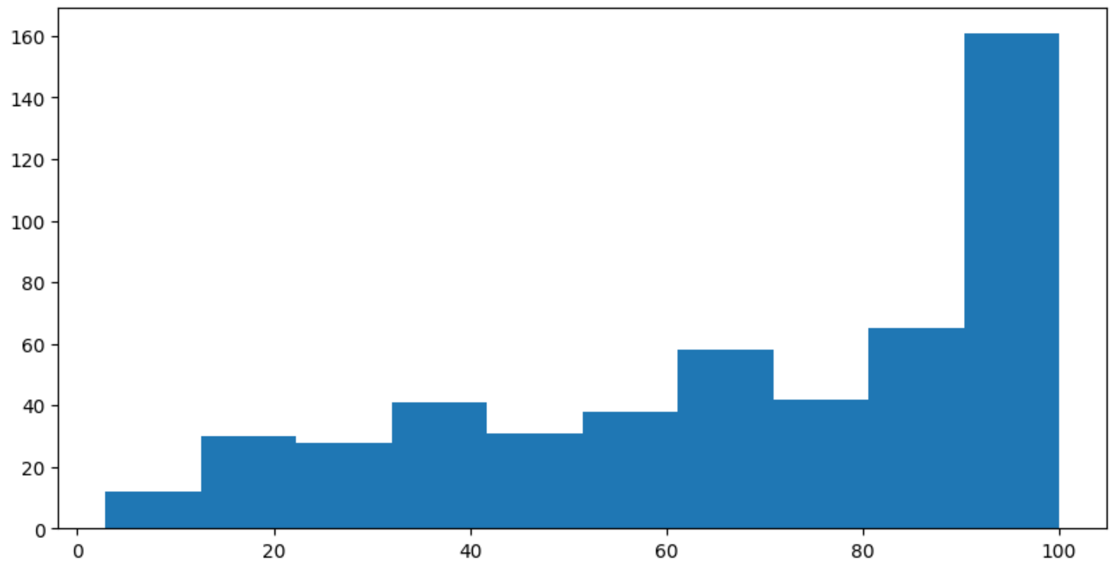
[16]:
```
data.describe()
```

[16]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.611874   11.211934   11.083992    0.069959    0.554695    6.284634
std      8.545770   22.921051    6.699165    0.250233    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.083235    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.290250    0.000000    9.900000    0.000000    0.538000    6.208500
75%      3.611874   11.211934   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    68.518519    3.795043    9.549407  408.237154   18.455534  356.674032
std     27.439466    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.925000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     74.450000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     93.575000    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     12.715432   22.532806
std       7.012739    9.197104
min       1.730000    5.000000
25%       7.230000   17.025000
50%      11.995000   21.200000
75%      16.570000   25.000000
max      37.970000   50.000000
```

2 Exploratory Data Analysis (EDA)

[17]:
```
data['AGE'].hist(bins=10, figsize = [10,5])
plt.grid(False)
plt.show()
```

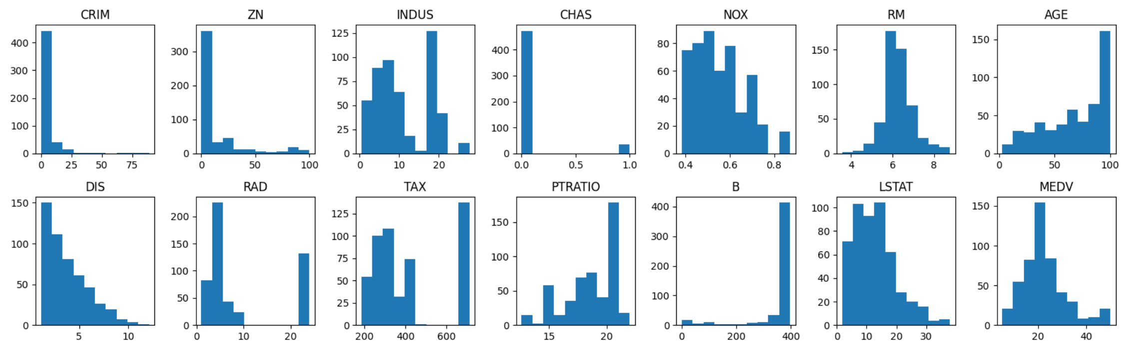


[18]:
```
data.columns
```

[18]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

[19]:
```
rows=2
cols=7

fig, axes = plt.subplots(rows, cols, figsize=(16, 5))

col = data.columns
index = 0
```

```python
for i in range(rows):
    for j in range(cols):
        axes[i][j].hist(data[col[index]])
        axes[i][j].set_title(col[index])
        index = index + 1

plt.tight_layout()
plt.show()
```



[20]:
```python
data.corr()
```

[20]:
```
             CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
CRIM     1.000000 -0.182930  0.391161 -0.052223  0.410377 -0.215434  0.344934
ZN      -0.182930  1.000000 -0.513336 -0.036147 -0.502287  0.316550 -0.541274
INDUS    0.391161 -0.513336  1.000000  0.058035  0.740965 -0.381457  0.614592
CHAS    -0.052223 -0.036147  0.058035  1.000000  0.073286  0.102284  0.075206
NOX      0.410377 -0.502287  0.740965  0.073286  1.000000 -0.302188  0.711461
RM      -0.215434  0.316550 -0.381457  0.102284 -0.302188  1.000000 -0.241351
AGE      0.344934 -0.541274  0.614592  0.075206  0.711461 -0.241351  1.000000
DIS     -0.366523  0.638388 -0.699639 -0.091680 -0.769230  0.205246 -0.724353
RAD      0.608886 -0.306316  0.593176  0.001425  0.611441 -0.209847  0.449989
TAX      0.566528 -0.308334  0.716062 -0.031483  0.668023 -0.292048  0.500589
PTRATIO  0.273384 -0.403085  0.384806 -0.109310  0.188933 -0.355501  0.262723
B       -0.370163  0.167431 -0.354597  0.050055 -0.380051  0.128069 -0.265282
LSTAT    0.434044 -0.407549  0.567354 -0.046166  0.572379 -0.602962  0.574893
MEDV    -0.379695  0.365943 -0.478657  0.179882 -0.427321  0.695360 -0.380223

              DIS       RAD       TAX   PTRATIO         B     LSTAT      MEDV
CRIM    -0.366523  0.608886  0.566528  0.273384 -0.370163  0.434044 -0.379695
ZN       0.638388 -0.306316 -0.308334 -0.403085  0.167431 -0.407549  0.365943
INDUS   -0.699639  0.593176  0.716062  0.384806 -0.354597  0.567354 -0.478657
CHAS    -0.091680  0.001425 -0.031483 -0.109310  0.050055 -0.046166  0.179882
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.572379 -0.427321
```

RM	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.602962	0.695360
AGE	-0.724353	0.449989	0.500589	0.262723	-0.265282	0.574893	-0.380223
DIS	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.483429	0.249929
RAD	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.468440	-0.381626
TAX	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.524545	-0.468536
PTRATIO	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.373343	-0.507787
B	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.368886	0.333461
LSTAT	-0.483429	0.468440	0.524545	0.373343	-0.368886	1.000000	-0.721975
MEDV	0.249929	-0.381626	-0.468536	-0.507787	0.333461	-0.721975	1.000000

[21]:
```
plt.subplots(figsize=(18,10))
sns.heatmap(data.corr(),annot=True,annot_kws={'size':14})
plt.show()
```

3 Train Test Split & Model Training

[22]:
```
X = data.drop(['MEDV'], axis=1)
X.head()
```

[22]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	

```
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222     18.7

        B      LSTAT
0  396.90   4.980000
1  396.90   9.140000
2  392.83   4.030000
3  394.63   2.940000
4  396.90  12.715432
```

[23]:
```python
y = data['MEDV']
y.head()
```

[23]:
```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: MEDV, dtype: float64
```

[24]:
```python
X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2,␣
 ↪random_state=0)

# The line of code you've mentioned performs a train-test split on your dataset␣
 ↪using the train_test_split function from the scikit-learn library.
# Let's break down what each part of this line does:

# X: This typically represents your feature data, the input variables that your␣
 ↪model will learn from.
# y: This usually represents the target or output variable you want to predict␣
 ↪based on your features.

# test_size=0.2: This parameter specifies that 20% of the data will be␣
 ↪allocated for testing, while 80% will be used for training.
# Adjusting this value changes the proportion of data used for training and␣
 ↪testing.

# random_state=0: This parameter sets the seed for the random number generator,␣
 ↪ensuring that the split is reproducible.
# Setting a specific random_state ensures that every time you run this code,␣
 ↪the data split will be the same.

# X_train: This variable stores the training data for the features (X) after␣
 ↪the split.
# This is the portion of your feature data that the model will learn from.

# X_test: This variable stores the testing data for the features (X) after the␣
 ↪split.
```

```
# This is the portion of your feature data that will be used to evaluate the␣
↪model's performance.

# y_train: This variable stores the training data for the target variable (y)␣
↪after the split.
# This corresponds to the target data associated with the X_train.

# y_test: This variable stores the testing data for the target variable (y)␣
↪after the split.
# This corresponds to the target data associated with the X_test.
```

[25]:
```
X_train.shape, X_test.shape
# 404 will undergo training and 506*20% = 102 will undergoes testing
```

[25]: ((404, 13), (102, 13))

[26]:
```
y_train.shape, y_test.shape
```

[26]: ((404,), (102,))

[27]:
```
model = LinearRegression()
model.fit(X_train,y_train)
```

[27]: LinearRegression()

[28]:
```
model.predict(X_test)
```

[28]:
```
array([26.175296  , 22.64747588, 29.1456294 , 11.52971235, 21.65312134,
       19.42320699, 20.18413017, 21.46914355, 19.1985363 , 19.98228162,
        4.32483046, 16.16891668, 16.87682404,  5.31232373, 39.36827861,
       33.09358732, 21.9152876 , 36.61918436, 31.52676377, 23.52713482,
       24.96022461, 23.69866912, 20.88033802, 30.55074901, 22.74081741,
        8.66805959, 17.65119072, 17.93088633, 36.01223185, 21.16299556,
       17.83464361, 17.43306603, 19.5240167 , 23.50605522, 28.97262793,
       19.21808862, 11.23997435, 23.94256597, 17.86786717, 15.40849806,
       26.3630836 , 21.5193299 , 23.78733694, 14.84041522, 23.9445175 ,
       24.97067627, 20.11366175, 23.08636158, 10.42208266, 24.52832122,
       21.60847326, 18.66228165, 24.53362832, 31.03502944, 12.97457826,
       22.38536236, 21.34822822, 16.10928673, 12.37477824, 22.78596712,
       18.28714824, 21.91802045, 32.49771603, 31.21256855, 17.47867791,
       33.18861907, 19.17896285, 19.94662594, 20.17142015, 23.90228857,
       22.81288844, 24.17911208, 30.83402844, 28.87481037, 25.14581721,
        5.55072029, 37.0183454 , 24.15428003, 27.67587636, 19.63884644,
       28.74874123, 18.83204358, 17.63305678, 37.97947167, 39.49507972,
       24.17228966, 25.33605088, 16.75044819, 25.43224687, 16.65089426,
       16.49186628, 13.37283452, 24.81689254, 31.21188699, 22.0891919 ,
       20.49360168,  0.8229737 , 25.5004737 , 15.5481509 , 17.72901193,
```

25.77663998, 22.43131323])

[29]:
```python
y_predict = model.predict(X_test)
```

4 Regression Evaluation Metrics

[30]:
```python
from sklearn.metrics import r2_score
```

[31]:
```python
r2_score(y_test,y_predict)
```

[31]: 0.5703296053895559

[32]:
```python
mean_absolute_error(y_test,y_predict)
```

[32]: 3.9616211239591177

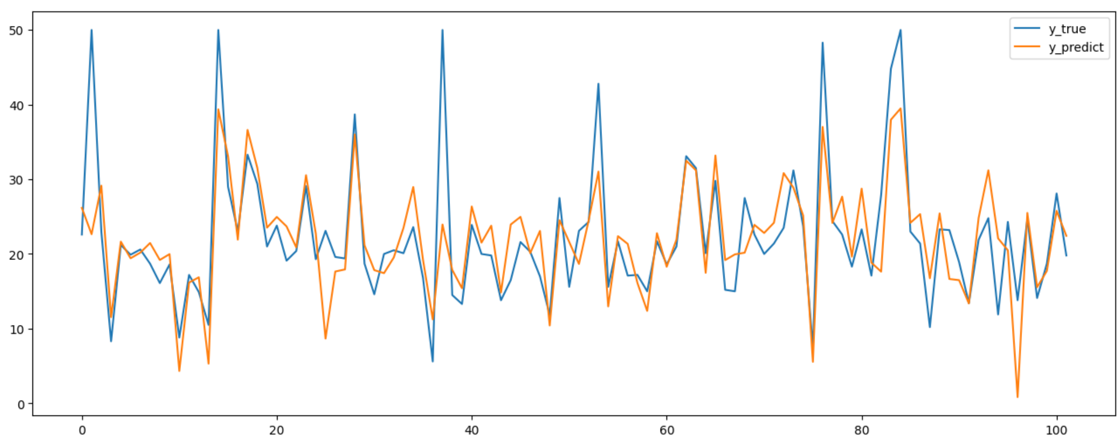
[33]:
```python
mean_squared_error(y_test,y_predict)
```

[33]: 34.987389544238766

[34]:
```python
np.sqrt(mean_squared_error(y_test,y_predict))
# root_mean_squared_error
```

[34]: 5.915013909048631

[35]:
```python
plt.figure(figsize=(16,6))
x_points = list(range(len(y_test)))
plt.plot(x_points,y_test,label='y_true')
plt.plot(x_points,y_predict,label='y_predict')
plt.legend()
plt.show()
```

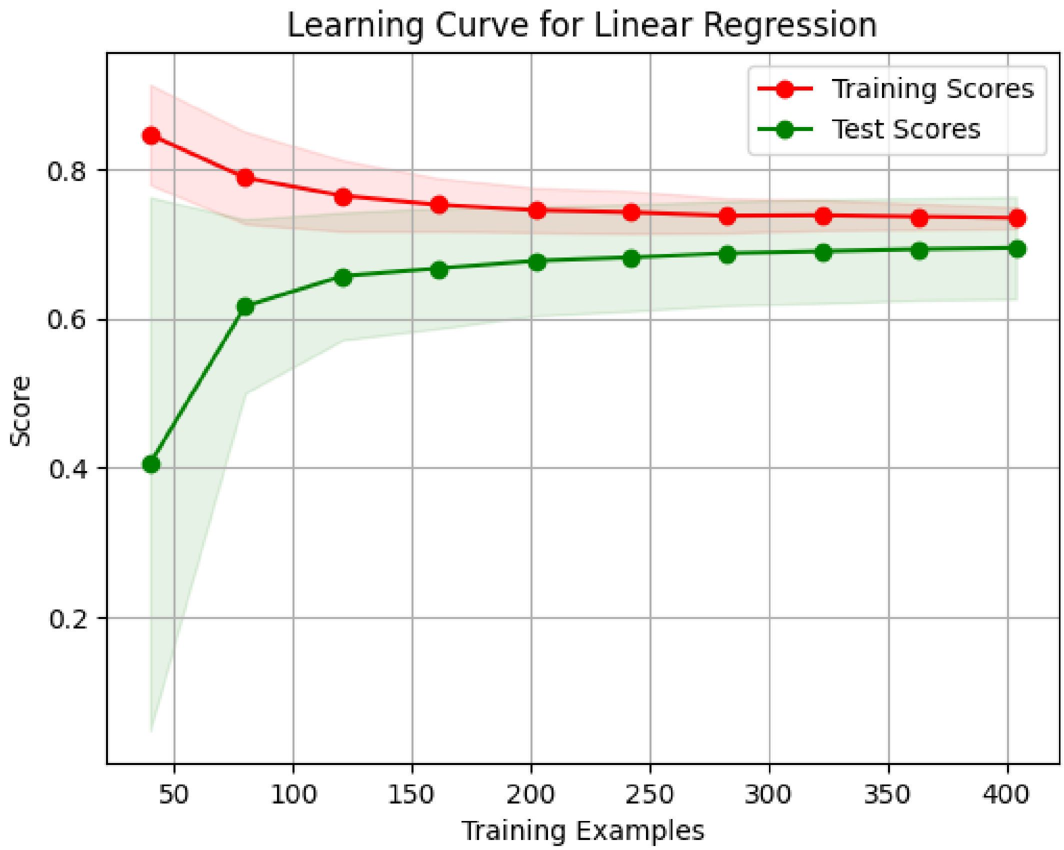


5 Plot Learning Curve

[36]:
```
from sklearn.model_selection import learning_curve, ShuffleSplit
```

[37]:
```
def plot_learning_curves(estimator, title, X, y, ylim=None, cv=None,␣
↪train_size=np.linspace(0.1, 1.0, 10)):
    plt.figure()
    plt.title(title)
    plt.xlabel('Training Examples')
    plt.ylabel('Score')

    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,␣
↪train_sizes=train_size, cv=cv)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,␣
↪color='red')
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,␣
↪color='green')

    plt.plot(train_sizes, train_scores_mean, 'o-', color='red', label='Training␣
↪Scores')
    plt.plot(train_sizes, test_scores_mean, 'o-', color='green', label='Test␣
↪Scores')

    plt.legend(loc='best')
    return plt
```

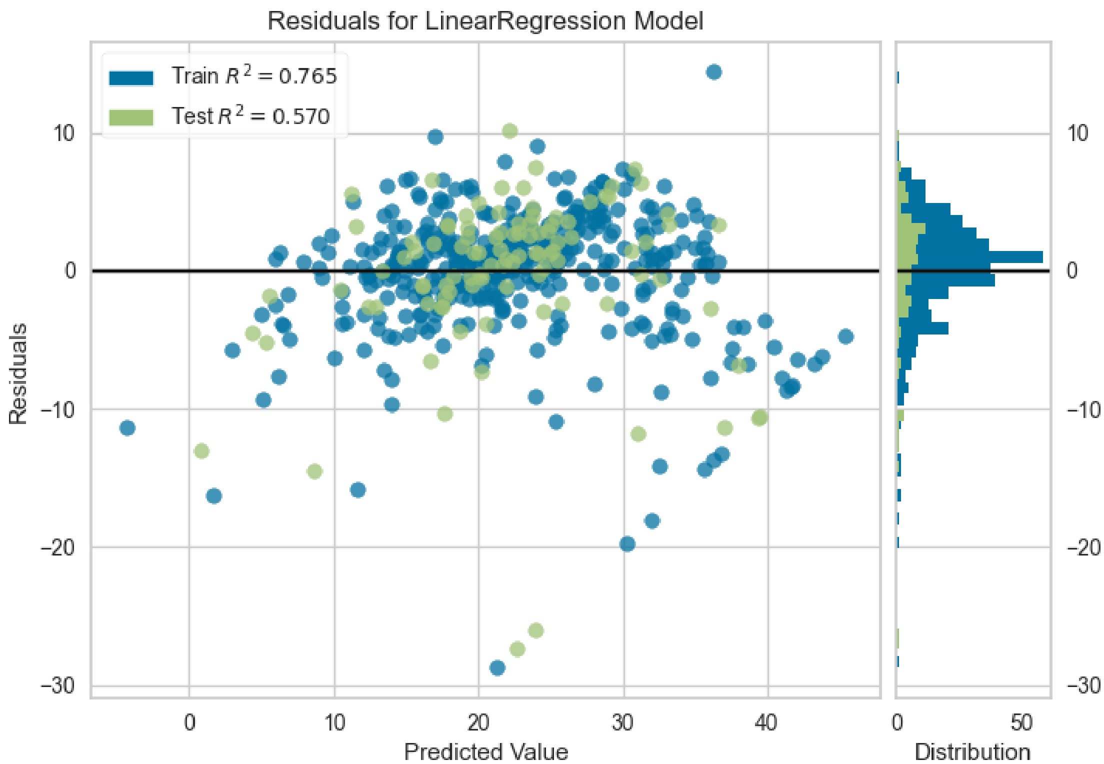
[38]:
```
title = 'Learning Curve for Linear Regression'
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)
model = LinearRegression()
plot_learning_curves(model, title, X, y, ylim=(0.7, 1.01), cv=cv)
plt.show()
```



6 Residuals Plot (Errors Plot)

[39]:
```
from yellowbrick.regressor import ResidualsPlot
```

[40]:
```
viz = ResidualsPlot(model)
viz.fit(X_train, y_train)
viz.score(X_test,y_test)
viz.show()
plt.show()
```



7 Prediction Error Plot

[41]:
```
from yellowbrick.regressor import PredictionError
```

[42]:
```
viz = PredictionError(model)
viz.fit(X_train, y_train)
viz.score(X_test,y_test)
viz.show()
plt.show()
```

```
C:\Users\hp\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:464: UserWarning: X does not have valid feature names,
but LinearRegression was fitted with feature names
  warnings.warn(
```

8 Thank You by Navjoth Singh