Boston house prices dataset

Data Set Characteristics:

```
    :Number of Instances: 506

    :Number of Attributes: 12 numeric/categorical predictive. Median
    Value (attribute 13) 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
        - 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.

In [ ]:
```
# Importando os módulos necessários
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
%matplotlib inline
```

Carregando o dataset

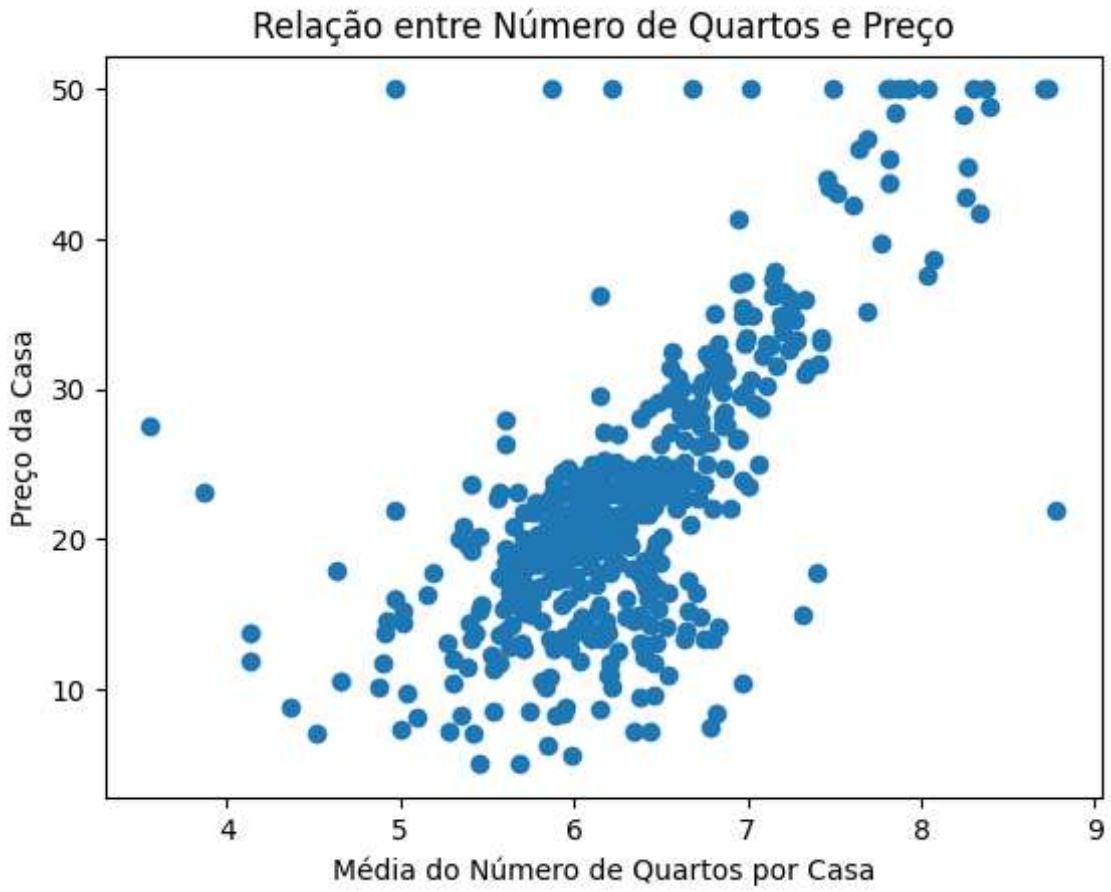
In [ ]:
```
df  = pd.read_csv('boston_house_prices.csv')
df = df.drop('B', axis=1)
df.rename(columns={'MEDV': 'PRICE'}, inplace=True)
df.head(5)
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTA
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	4.9
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	9.1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	4.0
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	2.9
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	5.3

Mostrando gráfico de correlação entre número de quartos por casa e preço da casa

In [ ]:
```
plt.scatter(df.RM, df.PRICE)
plt.xlabel("Média do Número de Quartos por Casa")
plt.ylabel("Preço da Casa")
plt.title("Relação entre Número de Quartos e Preço")
plt.show()
```



Treinando nosso modelo de regressão

PRICE = a * CRIM + b * ZN + c * INDUS + d * CHAS + e * NOX + f * RM + g * AGE + h * DIS + i * RAD + j * TAX + k * PTRATIO + j * LSTAT + k

In [ ]:
```
# Importando o módulo de regressão linear
from sklearn.linear_model import LinearRegression

# Criando o objeto de regressão linear
regr = LinearRegression()

# Treinando o modelo
X = df.drop('PRICE', axis = 1)
Y = df.PRICE
regr.fit(X, Y)

print("Número de Coeficientes Ajustados: ", len(regr.coef_))
regr.coef_
```

```
Número de Coeficientes Ajustados:  12
```

Out[ ]:
```
array([-1.21388618e-01,  4.69634633e-02,  1.34676947e-02,  2.83999338e+00,
       -1.87580220e+01,  3.65811904e+00,  3.61071055e-03, -1.49075365e+00,
        2.89404521e-01, -1.26819813e-02, -9.37532900e-01, -5.52019101e-01])
```

Analisando parâmetros e predições do modelo treinado

In [ ]:
```
# Coeficientes
regr.predict(X)
```

Out[ ]:
```
array([30.03373805, 25.05683368, 30.60818602, 28.67717948, 27.928791  ,
       25.39316857, 22.80496155, 19.19898495, 11.07897982, 18.71966912,
       18.62438629, 21.39246353, 20.60745482, 19.54258612, 19.42338826,
       19.29854778, 20.56134548, 16.87747529, 17.09180043, 18.43760689,
       12.52394623, 17.63469278, 15.59842603, 13.63779912, 15.57390124,
       14.14580816, 15.56515976, 15.35858528, 19.5109376 , 20.88646225,
       11.52648291, 18.19763703,  9.85998433, 14.49395604, 14.83212976,
       23.83455965, 22.48819096, 23.07938508, 22.83020538, 31.43055097,
       34.26922099, 27.99845331, 25.27611814, 24.52743894, 22.96863585,
       22.08532214, 20.28694959, 17.918101  ,  8.74324717, 17.11021002,
       21.15393966, 23.99706202, 27.58402585, 23.97009389, 15.26850351,
       31.07547354, 24.91295459, 33.15306888, 21.73625055, 21.02325709,
       17.79903953, 18.61209277, 23.96892556, 22.3430303 , 23.35021797,
       30.37899322, 25.51127036, 21.09138792, 17.34408592, 20.76945584,
       25.20041168, 21.81872788, 24.57909925, 24.10411089, 25.33936544,
       23.95908755, 23.03803615, 23.3090221 , 21.15826976, 22.33465511,
       28.39958723, 27.07192508, 26.05135651, 25.17390175, 24.75035765,
       27.87552006, 22.10809503, 25.98875871, 30.75726777, 30.83764803,
       27.19109696, 27.51330572, 28.88606875, 29.02990799, 26.9999881 ,
       29.05937631, 24.82381183, 35.75448863, 35.06045973, 32.23033461,
       24.49866791, 25.51863012, 22.7570127 , 20.23476635, 21.41321452,
       18.4651746 , 17.02417894, 20.72416707, 22.57867997, 19.66679089,
       20.48919123, 26.30782492, 20.555527  , 20.43148278, 25.09330765,
       20.67246864, 23.18330025, 23.58512823, 20.61186064, 20.61742827,
       21.77028564, 22.45723928, 20.47346235, 16.18052679, 20.49666144,
       22.40186839, 14.51561715, 14.93268603, 18.59200038, 13.72717122,
       19.77331743, 19.1429295 , 19.93033237, 15.570102  , 14.19079769,
       16.89923715, 15.72066087, 19.05743765, 13.44030043, 16.05755411,
       13.09511731,  3.32967117, 13.88459263, 11.27447859,  7.87359254,
       13.13203747, 17.33255246,  7.6962799 ,  9.21047762, 14.4672043 ,
       20.44690679, 18.34159397, 20.36177252, 17.93997867, 22.57833566,
       22.41762004, 15.92705934, 33.35593038, 29.30584631, 25.38768278,
       33.25274407, 36.73956621, 40.49736832, 41.66249638, 24.52460229,
       26.56656062, 37.10297334, 24.33508407, 27.0122213 , 26.93080118,
       23.19121549, 24.4652428 , 22.94172415, 29.06311055, 26.55672329,
       30.68105609, 25.53871813, 29.17702032, 31.42463769, 32.96699333,
       34.66764083, 27.83129046, 34.0380294 , 31.19061802, 22.87179761,
       24.83019858, 35.83675156, 33.34164058, 32.47619211, 34.35946734,
       30.74791856, 30.23492312, 32.827136  , 32.11549101, 31.79035541,
       40.74958036, 36.03691334, 32.88603707, 34.60268924, 30.01137099,
       30.6540026 , 29.3455338 , 36.94678876, 41.95475404, 43.12737947,
       22.52568884, 23.54902664, 17.71908196, 23.60134192, 16.96818856,
       22.43439864, 16.97554969, 22.75040716, 25.15433633, 10.93155162,
       24.42812597, 26.53719085, 28.20558333, 24.7939717 , 29.7215753 ,
       33.21671167, 23.57790683, 32.14885689, 29.65256765, 38.23777897,
       39.63765076, 37.51110207, 32.49240849, 35.3016529 , 31.24012589,
       24.51111139, 33.31956385, 37.90622207, 37.05715456, 32.0416543 ,
       25.29445543, 30.15934744, 32.61738815, 28.45952588, 28.44360171,
       27.11773792, 23.660733  , 24.18790795, 27.47877959, 16.53060743,
       13.26351402, 19.94187143, 20.00437518, 21.34598434, 23.90522898,
       24.05513543, 25.12729055, 24.8163637 , 29.54468588, 24.07078864,
       21.69765852, 37.59775716, 42.95323757, 36.36552257, 34.88843935,
       34.52135821, 36.9540611 , 40.69871984, 34.11949742, 35.66033413,
       28.12807857, 30.90876487, 40.54129164, 39.15983262, 25.88059697,
       22.30741736, 27.17405749, 28.55625698, 35.54159872, 36.27929516,
       33.84223479, 35.74462627, 34.99001238, 30.31007898, 35.25134128,
       38.79273883, 34.29920449, 40.57902278, 44.75976429, 31.48757529,
       27.35744126, 20.56022884, 26.98708013, 27.17167234, 26.95023815,
       33.51024601, 34.42711797, 31.89256898, 25.62968965, 24.27973293,
       28.27835023, 27.2692216 , 19.30674937, 29.3256931 , 31.96333873,
```

```
       30.74370235, 28.75063318, 28.77489774, 32.656176  , 33.09301556,
       30.75434071, 35.50450239, 32.70344201, 28.65584615, 23.5500484 ,
       18.8729942 , 26.85051923, 23.23096013, 25.56098598, 25.39831377,
       20.48802604, 17.41438191, 18.16883617, 24.15287647, 21.12155145,
       24.83301905, 24.8378026 , 22.84830911, 19.49625938, 25.0544708 ,
       24.57772771, 23.58947502, 19.14737695, 21.16592768, 24.28758127,
       21.70456362, 19.92667289, 23.63832484, 22.16760629, 21.5580741 ,
       20.55442555, 20.11932836, 19.27109562, 22.11860281, 21.19550563,
       21.42823263, 30.28888722, 21.97616043, 27.61307252, 28.48693981,
       16.57608924, 14.98044611, 25.24144708, 27.5418748 , 22.18348537,
       20.52303385, 20.72649866, 16.85249263, 25.49621734, 14.51447709,
       16.86576736, 18.9249723 , 22.01370294, 21.505635  , 18.37887073,
       22.09550903, 18.45361601, 17.7333695 , 19.93456612, 37.03036895,
       14.40977286, 15.65010111, 12.82666462, 23.74739099, 32.45297992,
       34.22327905, 24.56172871, 25.95754857,  4.90728023, -0.51023431,
       24.17932289, 16.80583219, 19.10395701, 14.50760846, 15.70316925,
       12.2447456 , 17.30767055, 12.40534499, 12.03239296,  3.10736201,
        6.77543587,  4.91094167,  4.23983611,  5.49213058, 13.29082726,
       16.37872796, 16.49487626,  8.8443811 , 19.31841045, 16.99643964,
       19.3702223 , 18.29484553, 15.44086659,  4.96950183, 10.12680097,
       10.49892814, 16.69585463, 17.39107284, 11.87446052,  6.41278908,
        6.5181009 ,  7.4041174 , 19.87401018, 13.48433235, 20.79406086,
       17.75005096, 19.24044364,  3.94186679, 12.51008005, -3.12350834,
       11.61658666, 15.55875131,  8.09936153,  7.84209989, 16.53213856,
       19.35901925, 17.95515455, 18.72756673, 15.81135498, 17.4639624 ,
       12.36266561, 18.90991539, 16.29333179, 15.9631735 , 15.00931008,
       20.17178196, 20.60041585, 23.53791255, 18.82543027, 17.72471974,
       14.76505361, 16.81358096, 11.11041776,  6.41563996, 11.98561496,
       11.49407045, 16.29194222, 17.81786431, 17.06072501, 11.79240653,
       14.01520753, 17.4130874 , 17.20050596, 16.51771346, 17.04663537,
       19.16432431, 18.78845003, 17.73288762, 21.55253231, 17.79530896,
       18.0132957 , 15.3185462 , 15.56268165, 17.35071573, 17.60628756,
       19.36540575, 19.32896893, 18.84904157, 21.63444902, 19.41828093,
       17.62156825, 17.00876086, 16.67577256, 16.24257405, 17.70187712,
       19.36331001, 22.34203147, 21.68268327, 24.90615709, 16.05102789,
       15.99239285, 19.62247578, 10.9528481 , 18.4488214 , 21.1234434 ,
       22.77041185, 26.47357803, 27.90139952, 20.46299172, 18.91032596,
       21.52695113, 18.89410437, 20.67281902, 11.44734989,  8.15251024,
        3.72203885, 13.33411349, 15.5232582 , 20.42201855, 20.29689558,
       16.48770678, 13.65509137, 18.87317724, 21.0529801 , 18.25418756,
       20.22485547, 23.49549557, 22.40532826, 27.66426716, 26.19267498,
       22.42882569])
```

Erro Absoluto Médio:

MAE = ∑(|Preço Original - Preço Predito|)

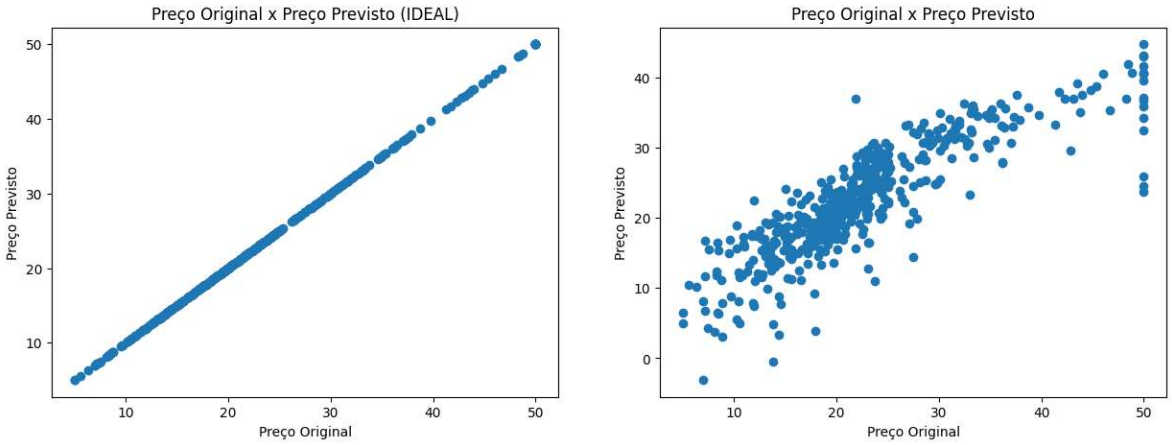
In [ ]:
```
# Vamos calcular o MAE (Mean Squared Error)
mse = np.mean(np.abs(df.PRICE - regr.predict(X)))

print(f"Erro médio absoluto: {mse:.2f}")
```

```
Erro médio absoluto: 3.36
```

Analisando a correlação entre os preços preditos e os preços reais

In [ ]:
```
# Comparando preços originais x preços previstos
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
```

```
ax[0].scatter(Y,Y)
ax[0].set_xlabel("Preço Original")
ax[0].set_ylabel("Preço Previsto")
ax[0].set_title("Preço Original x Preço Previsto (IDEAL)")

ax[1].scatter(Y, regr.predict(X))
ax[1].set_xlabel("Preço Original")
ax[1].set_ylabel("Preço Previsto")
ax[1].set_title("Preço Original x Preço Previsto")



plt.show()
```



E se treinarmos com apenas uma feature? Vamos testar para número de quartos

In [ ]:
```
# Aplicando regressão linear para apenas uma variável e calculando o MSE
regr = LinearRegression()
regr.fit(X[['RM']], df.PRICE)
mse = np.mean(np.abs((df.PRICE - regr.predict(X[['RM']])) ))
print(f"Erro médio absoluto: {mse:.2f}")
```

```
Erro médio absoluto: 4.45
```

Vamos analisar as correlações de cada variável com o preço das casas

In [ ]:
```
plt.figure(figsize = (12,6))
sns.heatmap(df.corr()['PRICE'].to_frame(), annot=True, cmap = 'BrBG')
plt.title("Correlação entre as variáveis e o preço da casa ")
plt.show()
```

Vamos tentar usar apenas a variável com maior correlação para ver o que acontece

In [ ]:
```
# Aplicando regressão linear para apenas uma variável e calculando o MSE
regr = LinearRegression()
regr.fit(X[['LSTAT']], df.PRICE)
mse = np.mean(np.abs((df.PRICE - regr.predict(X[['LSTAT']])) ))
print(f"Erro médio absoluto: {mse:.2f}")
```

```
Erro médio absoluto: 4.51
```

Vamos tentar retirar as duas variáveis que quase não tem correlação com o preço das casas

In [ ]:
```
# Criando o objeto de regressão linear
regr = LinearRegression()

# Treinando o modelo
X = df.drop(['PRICE', 'CHAS', 'DIS'], axis = 1)
Y = df.PRICE
regr.fit(X, Y)
# Coeficientes
print("Número de Coeficientes: ", len(regr.coef_))

# Vamos calcular o MAE (Mean Squared Error)
mse = np.mean(np.abs(df.PRICE - regr.predict(X)))

print(f"Erro médio absoluto: {mse:.2f}")
```

```
Número de Coeficientes:  10
Erro médio absoluto: 3.55
```

Desafio!

Existe um erro na estratégia de validação do modelo! Qual??

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Nós obtivemos nossas métricas com os mesmos dados utilizados no treinamento, isso torna nossas métricas de validação enviesadas e fracas

Para resolver isso, iremos separar 1/4 do nosso dataset para validação e iremos treiná-lo com apenas 3/4 dos dados

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

# Dividindo X e Y em dados de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = 0.25,


print(f"Quantidade de ítems de treinamento: {X_treino.shape[0]}")
print(f"Quantidade de ítems de teste: {X_teste.shape[0]}")
```

```
Quantidade de ítems de treinamento: 379
Quantidade de ítems de teste: 127
```

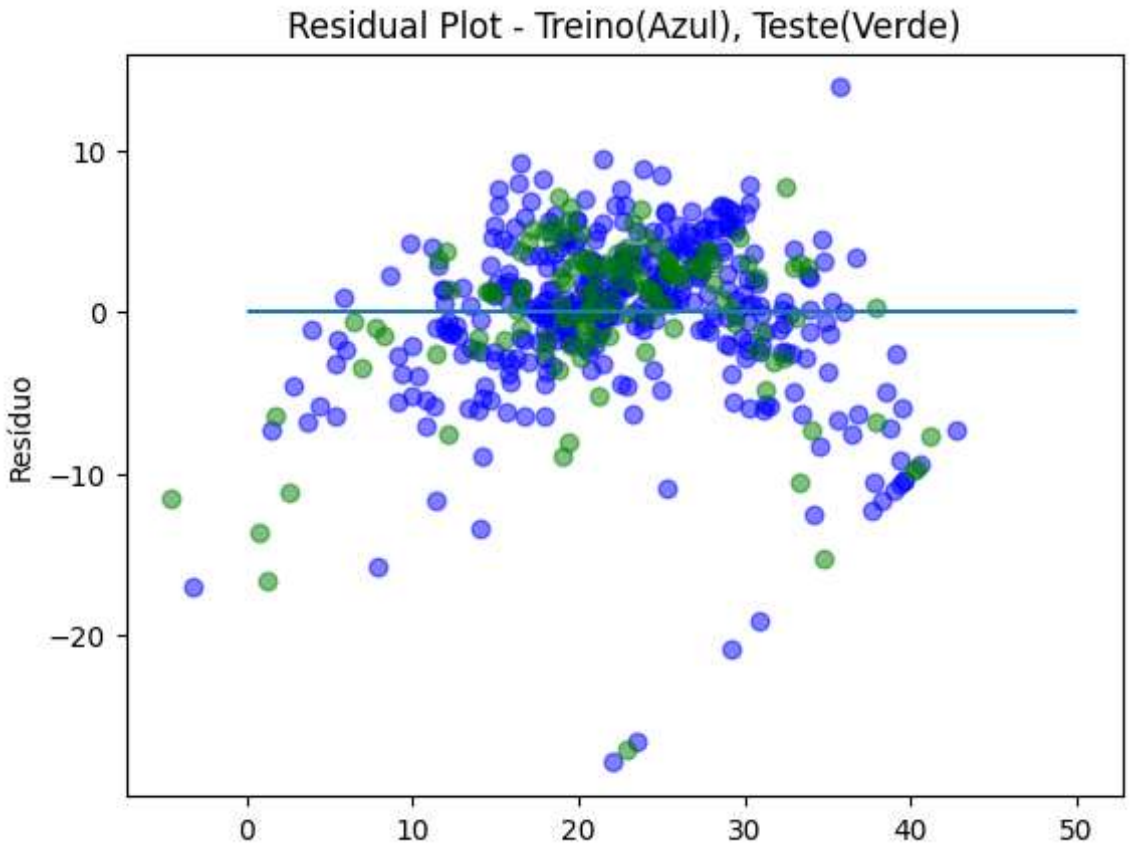
In [ ]:
```
# Construindo um modelo de regressão
regr = LinearRegression()

# Treinando o modelo
regr.fit(X_treino, Y_treino)
```

Out[ ]:
```
▾ LinearRegression

LinearRegression()
```

In [ ]:
```
# Vamos calcular o MAE (Mean Squared Error)
mse = np.mean(np.abs(Y_teste - regr.predict(X_teste)))

print(f"Erro médio absoluto: {mse:.2f}")
```

```
Erro médio absoluto: 3.58
```

In [ ]:
```
# Definindo os dados de treino e teste
pred_treino = regr.predict(X_treino)
pred_teste = regr.predict(X_teste)
# Comparando preços originais x preços previstos
plt.scatter(regr.predict(X_treino), regr.predict(X_treino) - Y_treino, c = 'b',
plt.scatter(regr.predict(X_teste), regr.predict(X_teste) - Y_teste, c = 'g', s =
plt.hlines(y = 0, xmin = 0, xmax = 50)
plt.ylabel("Resíduo")
plt.title("Residual Plot - Treino(Azul), Teste(Verde)")
plt.show()
```


Residual Plot - Treino(Azul), Teste(Verde)
Resíduo
10
0
−10
−20
0
10
20
30
40
50