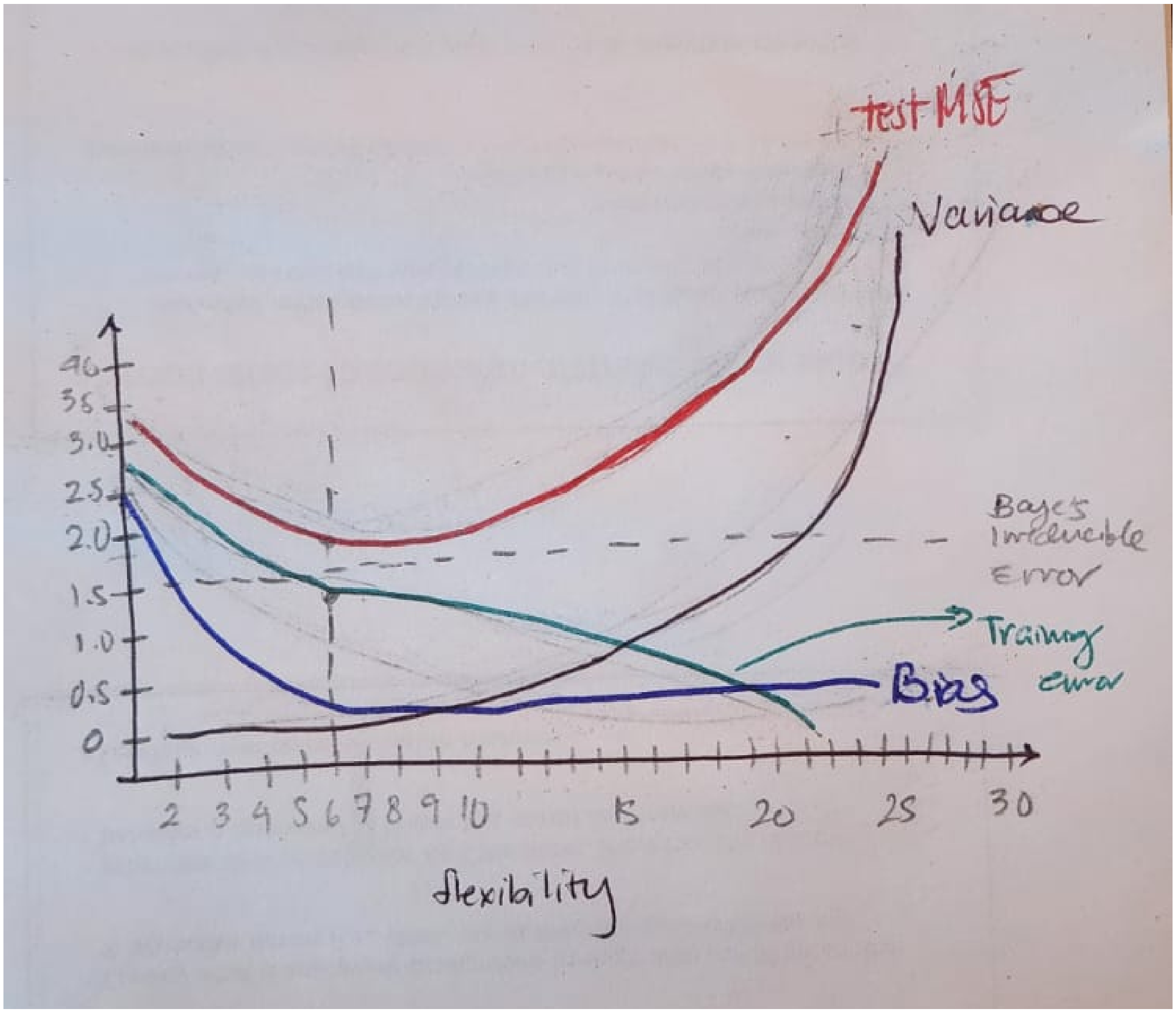
ML_PS1

Clarice Tee

2025-01-21

Part 1

```
picture =r'C:\Users\clari\OneDrive\Documents\Machine Learning\ps1'
```

1.a. 

1.b. squared bias-this goes down monotonically until it flattens out because greater flexibility leads to the model better being able to capture the true relationship in the data, until a certain point (diminishing returns) where it doesn't add much else.

irreducible error- this is the lower limit of the test MSE, thus it is a straight line. The test MSE is above tis line and the part of the training MSE below this line is when the data has been overfitted.

test MSE- its shape is concave, with the curve going upwards because more flexibility yields a better fit, until it overfits.

training MSE- this error goes down monotonically since more flexibility leads to better fitting data.

variance- this increases monotonically as flexibility increases, up to a point where there is overfitting.

2. Advantages (flexible): You have less bias and a better fit for non-linear models.

Disdvantages (flexible): Because we are estimating more parameters, it's more likey that we have overfitting from having too much noise. THis also means greater variance observed in the model.

A more flexible approach would be better if we want to capture more complex patterns from a larger sample size. This is suitable when we care more about the prediction we can make with the data, although there may be more variance.

A less flexible is better when we have a less data and are more interested in trying to interpret or make sense of our results. However, we are more likely to have biased restuls.

3.a.

```
# Import the libraries
import numpy as np
import os
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
```

3.b. There are 506 rows and 14 columns. The rows are the census tracts in Boston and the columns are the predictors. Based on the text file, the variables reporesent these things specifically:

```
# Load the dataset
directory = r"C:\Users\clari\OneDrive\Documents\Machine
↪ Learning\ps1\Data-Boston\Boston"
boston_path = os.path.join(directory, "Boston.csv")
boston_df = pd.read_csv(boston_path)
print(boston_df.dtypes)
print(boston_df.shape)
```

```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
```

```
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
MDEV       float64
dtype: object
(506, 14)
```

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

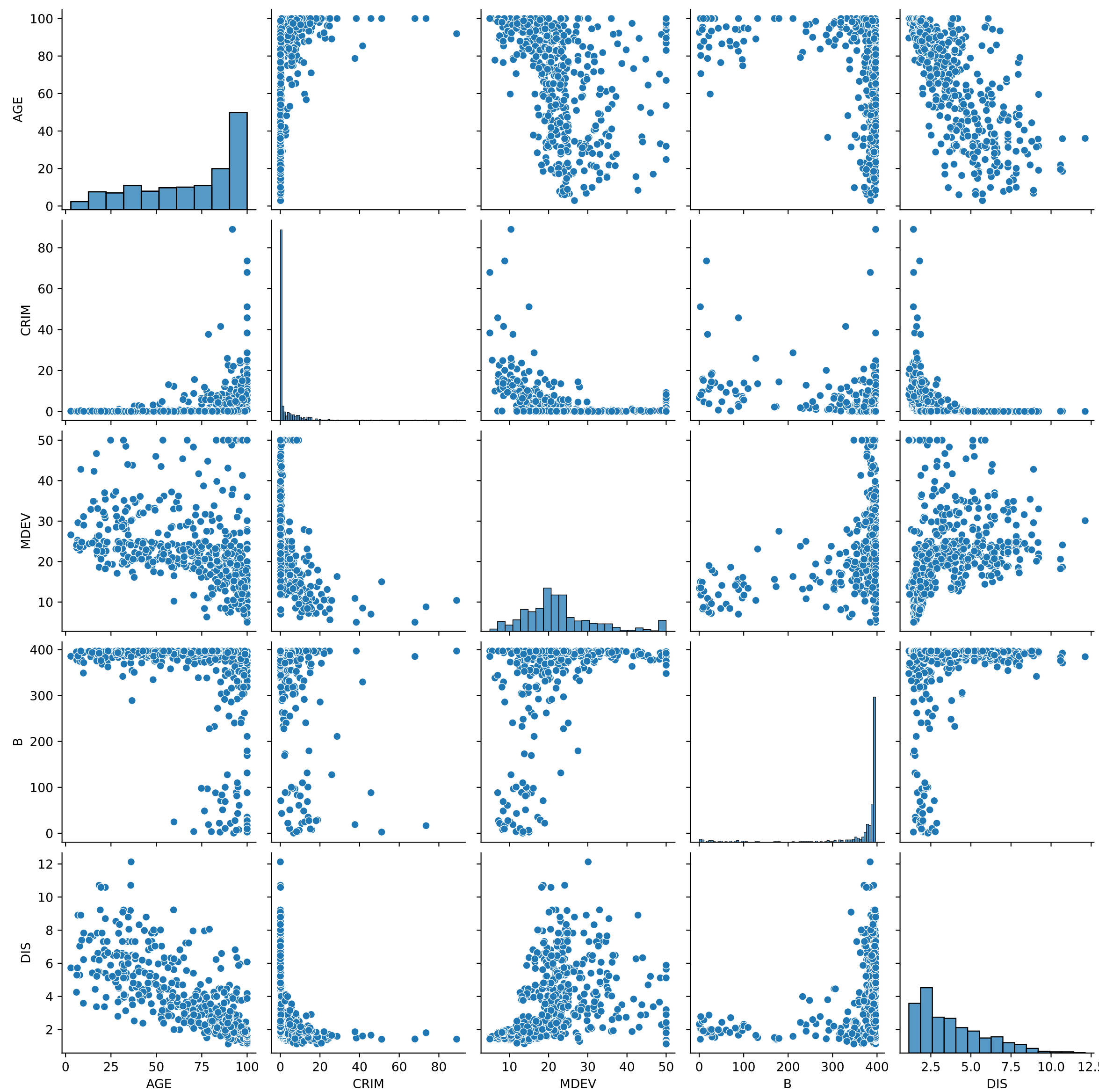
```python
# Lookup the missing values
print(boston_df.isna().sum())
```

```
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
MDEV       0
dtype: int64
```

3.c.

```python
pairwise = sns.pairplot(boston_df, vars = ['AGE', 'CRIM','MDEV', 'B', 'DIS'])
plt.show()
```

```
C:\Users\clari\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.

C:\Users\clari\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.

C:\Users\clari\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.

C:\Users\clari\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.

C:\Users\clari\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
```



Age vs CRIM: More old buildings, more observations of crime and higher per capita crime rates by town

B vs DIS: Towns with a higher proportion of Black people are father from the Boston employment centers.

MDEV vs CRIM In homes not occuppied by owners, more observations of crime and higher per capita crime rates by town

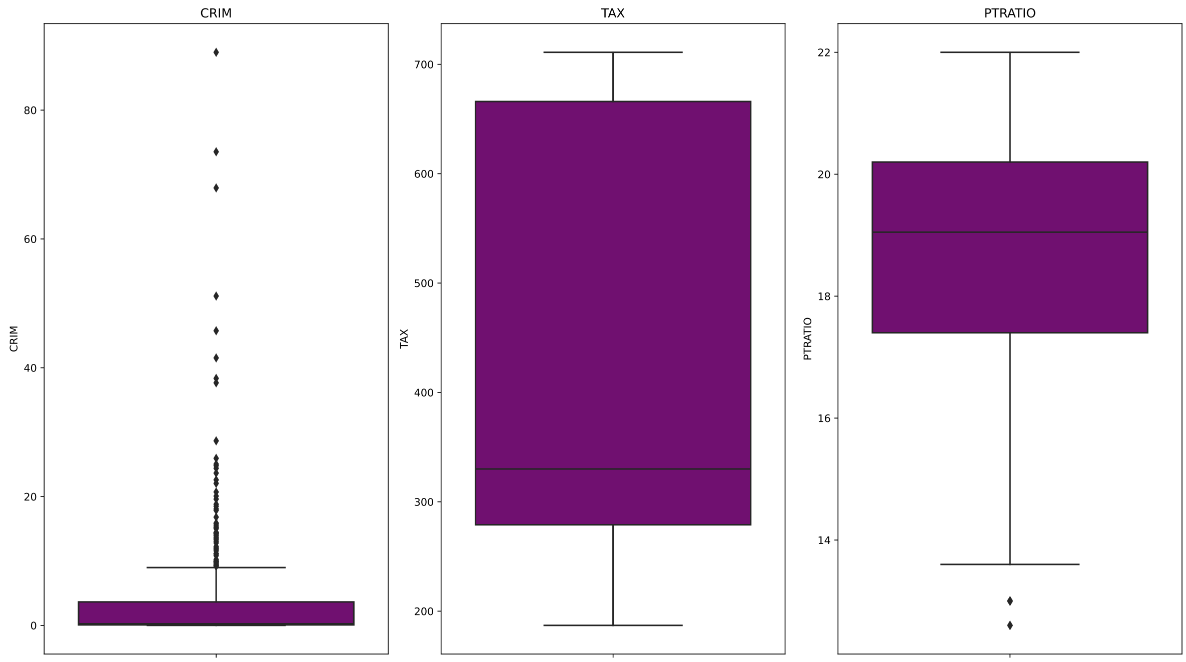
DIS vs CRIM: There seems to be more and higher per capita crime rates by town when closer to the Boston employment centers.

DIS vs AGE: It looks like a negative lniear relationship between the two. Towns with more pre 1940's buildings are closer to the Boston employment centers.

3.d. All the others that I chose seem to have a somewhat similar pattern. AGE and B, it is more like a positive correlation, meaning there are more observations and higher per capite crime rates in places with a higher proportion of older homes and Black population. For MDEV and DIS, it looks like a negative correlation. However, none of these look like a clear linear relationship with crime. The data is even highly varied at some points.

3.e.

```
#Box plot columns
box_values = ["CRIM", "TAX", "PTRATIO"]

fig, axs = plt.subplots(1, 3, figsize=(16, 9))

for i, col in enumerate(box_values):
    sns.boxplot(y=boston_df[col], ax=axs[i], color='purple')
    axs[i].set_title(f"{col}")
    axs[i].set_ylabel(col)
    axs[i].set_xlabel("")


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



Reference Lab and https://stackoverflow.com/questions/48176920/how-to-iteratively-plot-different-data-as-boxplots-in-seaborn-without-them-over

CRIM has the largest range. There is a small interquartiel range (most cities have low per capita crime rates) and a lot of outliers (values that fall outside of the min max whixkers). This may indicate some data measurement or encoding errors or something else that is happening that must be investigated.

TAX has the narrowest range of the three. It seems that the broad IQR caputes most of the data, leaving no outliers.

Pupil-teacher ratios have a small range, the highest median among the three and have a large whixker range relative to the IQR(the min and max values are far from the middle values) and there are outliers. It is also more skewed to higher ratios

3.f. There are 35 census tracts that bound the Charles River.

```
chas_subset = boston_df[boston_df['CHAS'] == 1]
chas_count = chas_subset.shape[0]

print(f"There are {chas_count} tracts that bound the Charles River")
```

```
There are 35 tracts that bound the Charles River
```

3.g.

```
md_ptratio = boston_df["PTRATIO"].median()

print(f"The median PTRATIO is {md_ptratio}.")
```

```
The median PTRATIO is 19.05.
```

3.h. Get the min MDEV

```
boston_df_sort = boston_df.sort_values("MDEV")

min_MDEV = boston_df_sort.head(5)

# Print the row
print(min_MDEV)
```

```
         CRIM   ZN  INDUS  CHAS    NOX     RM    AGE     DIS   RAD    TAX  \
398  38.35180  0.0   18.1   0.0  0.693  5.453  100.0  1.4896  24.0  666.0
405  67.92080  0.0   18.1   0.0  0.693  5.683  100.0  1.4254  24.0  666.0
400  25.04610  0.0   18.1   0.0  0.693  5.987  100.0  1.5888  24.0  666.0
399   9.91655  0.0   18.1   0.0  0.693  5.852   77.8  1.5004  24.0  666.0
414  45.74610  0.0   18.1   0.0  0.693  4.519  100.0  1.6582  24.0  666.0

     PTRATIO       B  LSTAT  MDEV
398     20.2  396.90  30.59   5.0
405     20.2  384.97  22.98   5.0
400     20.2  396.90  26.77   5.6
399     20.2  338.16  29.97   6.3
414     20.2   88.27  36.98   7.0
```

Tracts 398 and 405 have the minimum MDEV value of 5.

```
min_MDEV_rows = boston_df.loc[[398, 405]]

# Getting the summary statistics for all predictors
summary_stats_MDEV = boston_df.describe().T[['min', 'max', 'mean']]

# Add the columns w the values from 398 and 405 to the summary
summary_stats_MDEV['Row_398'] = min_MDEV_rows.iloc[0]
summary_stats_MDEV['Row_405'] = min_MDEV_rows.iloc[1]


print(summary_stats_MDEV)
```

```
              min       max        mean   Row_398   Row_405
CRIM       0.00632   88.9762    3.593761   38.3518   67.9208
ZN         0.00000  100.0000   11.363636    0.0000    0.0000
INDUS      0.46000   27.7400   11.136779   18.1000   18.1000
CHAS       0.00000    1.0000    0.069170    0.0000    0.0000
NOX        0.38500    0.8710    0.554695    0.6930    0.6930
RM         3.56100    8.7800    6.284634    5.4530    5.6830
AGE        2.90000  100.0000   68.574901  100.0000  100.0000
DIS        1.12960   12.1265    3.795043    1.4896    1.4254
RAD        1.00000   24.0000    9.549407   24.0000   24.0000
TAX      187.00000  711.0000  408.237154  666.0000  666.0000
PTRATIO   12.60000   22.0000   18.455534   20.2000   20.2000
B          0.32000  396.9000  356.674032  396.9000  384.9700
LSTAT      1.73000   37.9700   12.653063   30.5900   22.9800
MDEV       5.00000   50.0000   22.532806    5.0000    5.0000
```

CRIM: Areas with this MDEV have per capita crime rates far above the min and mean. They are outlier values (see boxplot)

AGE: Areas with this MDEV are outliers, with their AGE values being the maximum.

DIS: Areas with this MDEV fall within the range of values, below the mean and above the minimum.

RAD:Areas with this MDEV are outliers, with their RAD values being the maximum.

CHAS: They are not bounded by the Charles river

TAX: Areas with this MDEV lie outside of the IQR (see boxplot), with their TAX values being closer to the maximum.

PTRATIO: Areas with this MDEV fall within the tange, bu are higher than the mean, closer to the max value.

LSTAT: Areas with this MDEV fall within the tange, bu are higher than the mean, closer to the max value.

3.i.

```
# RM > 7
RM_7 = boston_df[boston_df["RM"] > 7]
RM_7_count = RM_7.shape[0]

# RM > 8
RM_8 = boston_df[boston_df["RM"] > 8]
RM_8_count = RM_8.shape[0]
```

```
print(f"There are {RM_7_count} rooms with an average of more than 7 rooms per
↪  dwelling")

print(f"There are {RM_8_count} rooms with an average of more than 8 rooms per
↪  dwelling")
```

```
There are 64 rooms with an average of more than 7 rooms per dwelling
There are 13 rooms with an average of more than 8 rooms per dwelling
```

Getting the summary stats

```
# Summary stats for RM > 7
RM_8_summary = RM_8.agg(['mean', 'min', 'max']).T

# Compute overall mean, min, and max for comparison
summary_stats_RM = boston_df.describe().T[['min', 'max', 'mean']]

# Combine the two summaries
comparison_summary_RM_7 = summary_stats_RM.join(RM_8_summary,
↪  lsuffix='_Overall', rsuffix='_RM>8')

# Display the comparison
print(comparison_summary_RM_7)


# Summary stats for RM > 8
RM_8_summary = RM_8.agg(['mean', 'min', 'max']).T

# Compute overall mean, min, and max for comparison
summary_stats_RM = boston_df.describe().T[['min', 'max', 'mean']]

# Combine the two summaries
comparison_summary_RM_8 = summary_stats_RM.join(RM_8_summary,
↪  lsuffix='_Overall', rsuffix='_RM>8')

# Display the comparison
print(comparison_summary_RM_8)
```

```
         min_Overall  max_Overall  mean_Overall   mean_RM>8   min_RM>8  \
CRIM         0.00632      88.9762      3.593761    0.718795    0.02009
ZN           0.00000     100.0000     11.363636   13.615385    0.00000
INDUS        0.46000      27.7400     11.136779    7.078462    2.68000
```

```
CHAS          0.00000       1.0000      0.069170    0.153846    0.00000
NOX           0.38500       0.8710      0.554695    0.539238    0.41610
RM            3.56100       8.7800      6.284634    8.348538    8.03400
AGE           2.90000     100.0000     68.574901   71.538462    8.40000
DIS           1.12960      12.1265      3.795043    3.430192    1.80100
RAD           1.00000      24.0000      9.549407    7.461538    2.00000
TAX         187.00000     711.0000    408.237154  325.076923  224.00000
PTRATIO      12.60000      22.0000     18.455534   16.361538   13.00000
B             0.32000     396.9000    356.674032  385.210769  354.55000
LSTAT         1.73000      37.9700     12.653063    4.310000    2.47000
MDEV          5.00000      50.0000     22.532806   44.200000   21.90000

          max_RM>8
CRIM       3.47428
ZN        95.00000
INDUS     19.58000
CHAS       1.00000
NOX        0.71800
RM         8.78000
AGE       93.90000
DIS        8.90670
RAD       24.00000
TAX      666.00000
PTRATIO   20.20000
B        396.90000
LSTAT      7.44000
MDEV      50.00000
         min_Overall  max_Overall  mean_Overall   mean_RM>8   min_RM>8  \
CRIM          0.00632      88.9762      3.593761    0.718795    0.02009
ZN            0.00000     100.0000     11.363636   13.615385    0.00000
INDUS         0.46000      27.7400     11.136779    7.078462    2.68000
CHAS          0.00000       1.0000      0.069170    0.153846    0.00000
NOX           0.38500       0.8710      0.554695    0.539238    0.41610
RM            3.56100       8.7800      6.284634    8.348538    8.03400
AGE           2.90000     100.0000     68.574901   71.538462    8.40000
DIS           1.12960      12.1265      3.795043    3.430192    1.80100
RAD           1.00000      24.0000      9.549407    7.461538    2.00000
TAX         187.00000     711.0000    408.237154  325.076923  224.00000
PTRATIO      12.60000      22.0000     18.455534   16.361538   13.00000
B             0.32000     396.9000    356.674032  385.210769  354.55000
LSTAT         1.73000      37.9700     12.653063    4.310000    2.47000
MDEV          5.00000      50.0000     22.532806   44.200000   21.90000
```

```
         max_RM>8
CRIM      3.47428
ZN       95.00000
INDUS    19.58000
CHAS      1.00000
NOX       0.71800
RM        8.78000
AGE      93.90000
DIS       8.90670
RAD      24.00000
TAX     666.00000
PTRATIO  20.20000
B       396.90000
LSTAT     7.44000
MDEV     50.00000
```

On average, the tracts with greater than 8 rooms have a ver low CRIM, lower than the overall CRIM mean, while the proportion of Black population is higher than the mean, closer to the maximum overall value. A thing to note is that they do have a high MDEV as well, meaning that these rooms are likely rented out.

4.a.

Y = 50 + 20(GPA) + 0.07(IQ) + 35(LVL) + 0.01(GPA * IQ) - 10 (GPA * LVL) college: (LVL = 1) 35−10 GPA highschool: (LVL = 0) 0 (35−10 GPA)−0=35−10 GPA GPA = 2, College students earn = 35 - 10 * 2 = 15K more

3, College students earn = 35 - 10 * 3 = 5k more

3.5, College students earn = 35 - 10 * 3.5 = don't earn more

4, HS students = 35 - 10 * 4 = earn 5k more

The correct answer is: iii. For a fixed value of IQ and GPA, high school graduates earn more, on average, than college graduates provided that the GPA is high enough.

4.b. Predict the salary of a college graduate with IQ of 110 and a GPA of 4.0. 50 + 20(4) + 0.07(110) + 35(1) + 0.01(4 * 110) - 10 (4 * 1) 50+ 80 + 7.7 + 35 + 4.4 -40 = $137.1K

4.c. False. We need look at the standard error of the coefficient, as well as the sigma squared (unexplained variation in Y).We learned that the magnitudes and indiv hypthesis tests aren't really good wys to asses the models.Sometimes, a small magnitutde might signify a big change (from GPA 3 to 4) vs IQ of 90 vs 130.Moreover, we want to look at it's relation to other predictors too.

5.a.

```python
# List of indevependnt variables
independent_var = boston_df.columns[1:]

# Function to fit a model and collect results
def simple_reg(x):
    reg = smf.ols(f"CRIM ~ {x}", data=boston_df).fit()
    return {
        "predictor": x,
        "coefficient": reg.params.iloc[1],
        "p-value": reg.pvalues.iloc[1],
        "adjusted R-squared (single)": reg.rsquared_adj
    }


# Apply the function to each predictor and create a DataFrame
results_simple_reg = pd.DataFrame([simple_reg(x) for x in independent_var])

print(results_simple_reg)
```

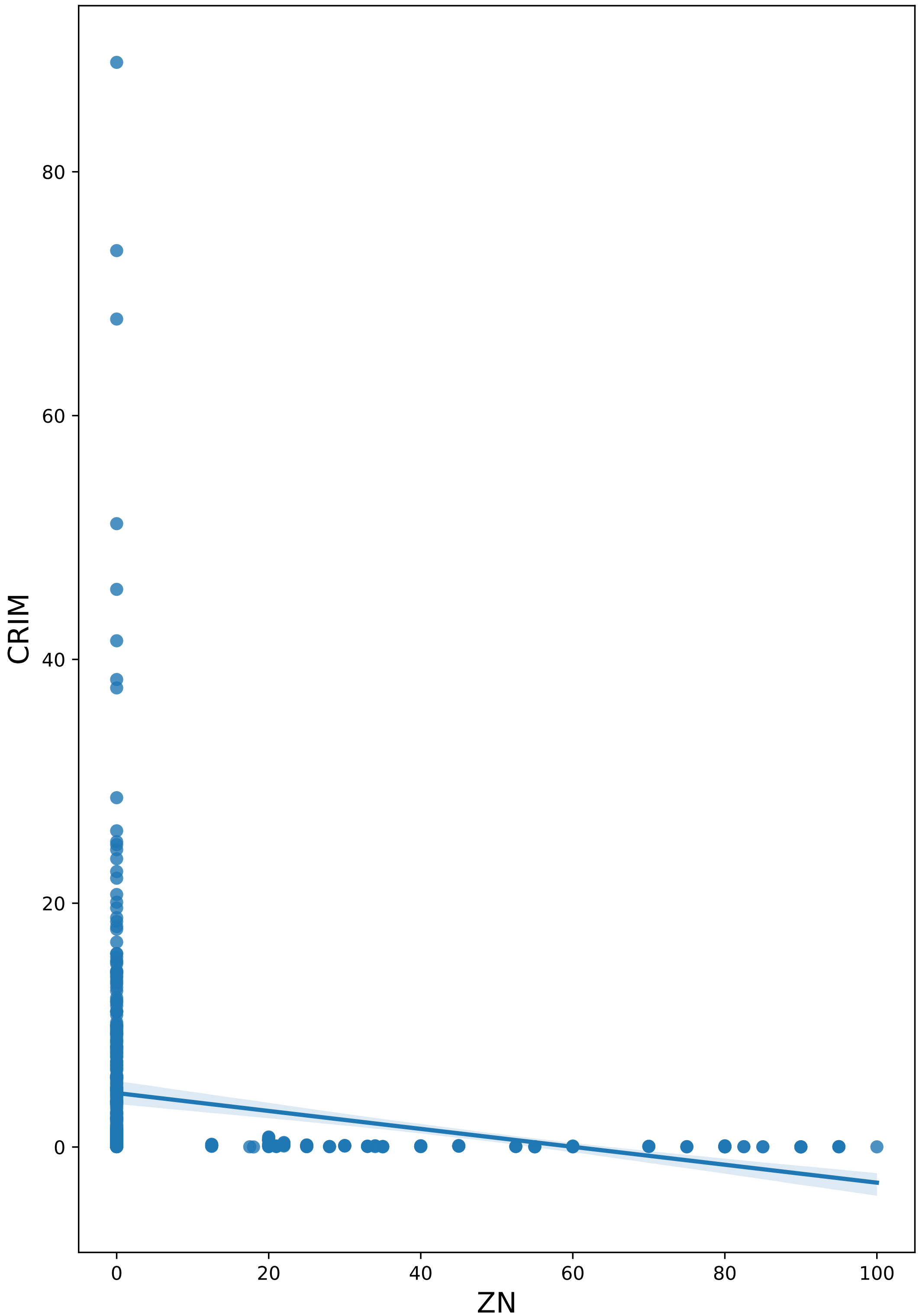
```
   predictor  coefficient       p-value  adjusted R-squared (single)
0         ZN    -0.073521  6.151722e-06                     0.037878
1      INDUS     0.506847  2.444137e-21                     0.161937
2       CHAS    -1.871545  2.143436e-01                     0.001080
3        NOX    30.975259  9.159490e-23                     0.172686
4         RM    -2.691045  5.838094e-07                     0.046485
5        AGE     0.107131  4.259064e-16                     0.121309
6        DIS    -1.542831  1.268832e-18                     0.141110
7        RAD     0.614137  1.620605e-55                     0.385704
8        TAX     0.029563  9.759521e-47                     0.334577
9    PTRATIO     1.144613  3.875122e-11                     0.081269
10         B    -0.035535  1.432088e-18                     0.140702
11     LSTAT     0.544406  7.124778e-27                     0.202925
12      MDEV    -0.360647  2.083550e-19                     0.147177
```

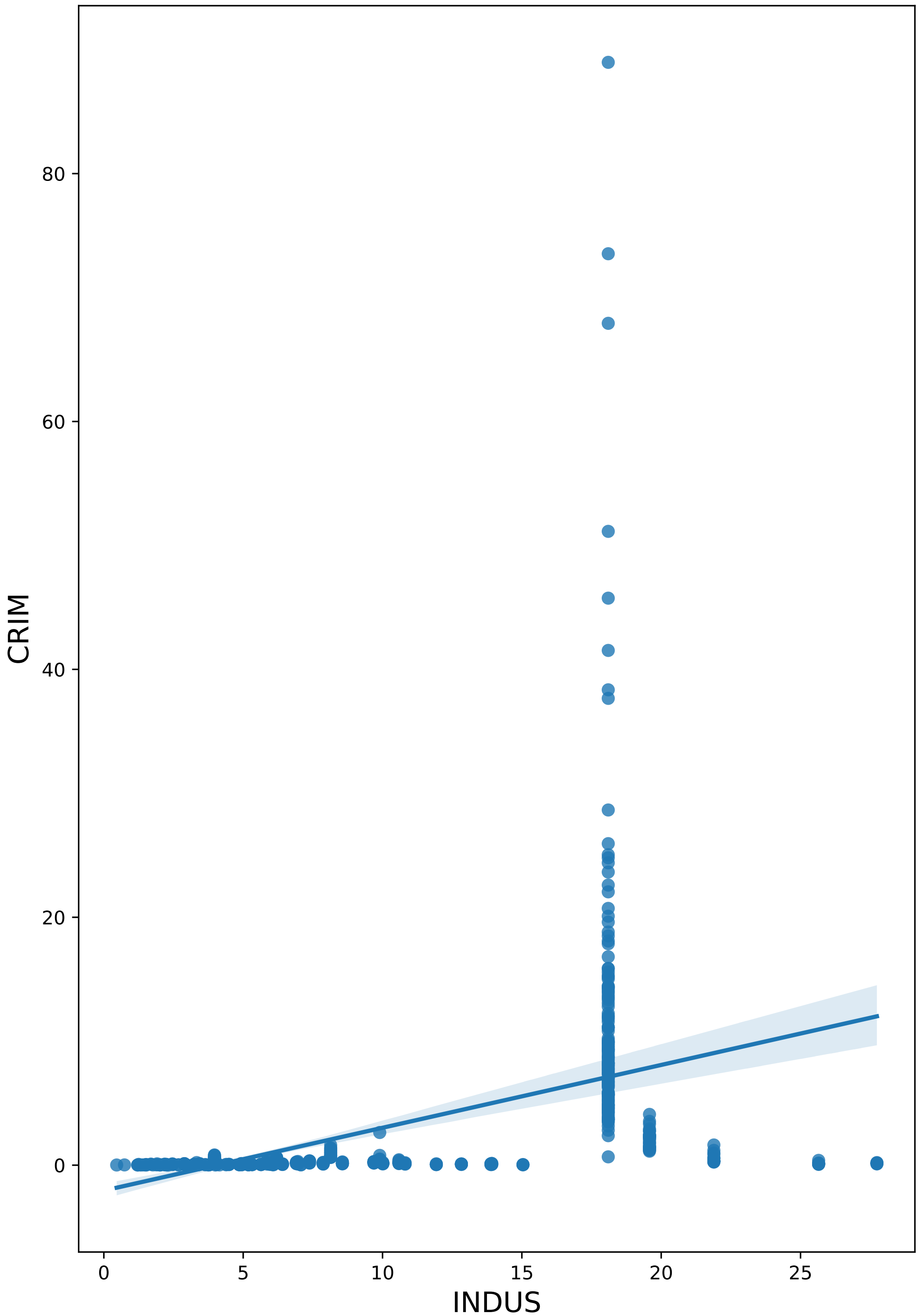
Sources: https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLSResults.rsquare
https://stackoverflow.com/questions/41075098/how-to-get-the-p-value-in-a-variable-from-olsresults-in-python

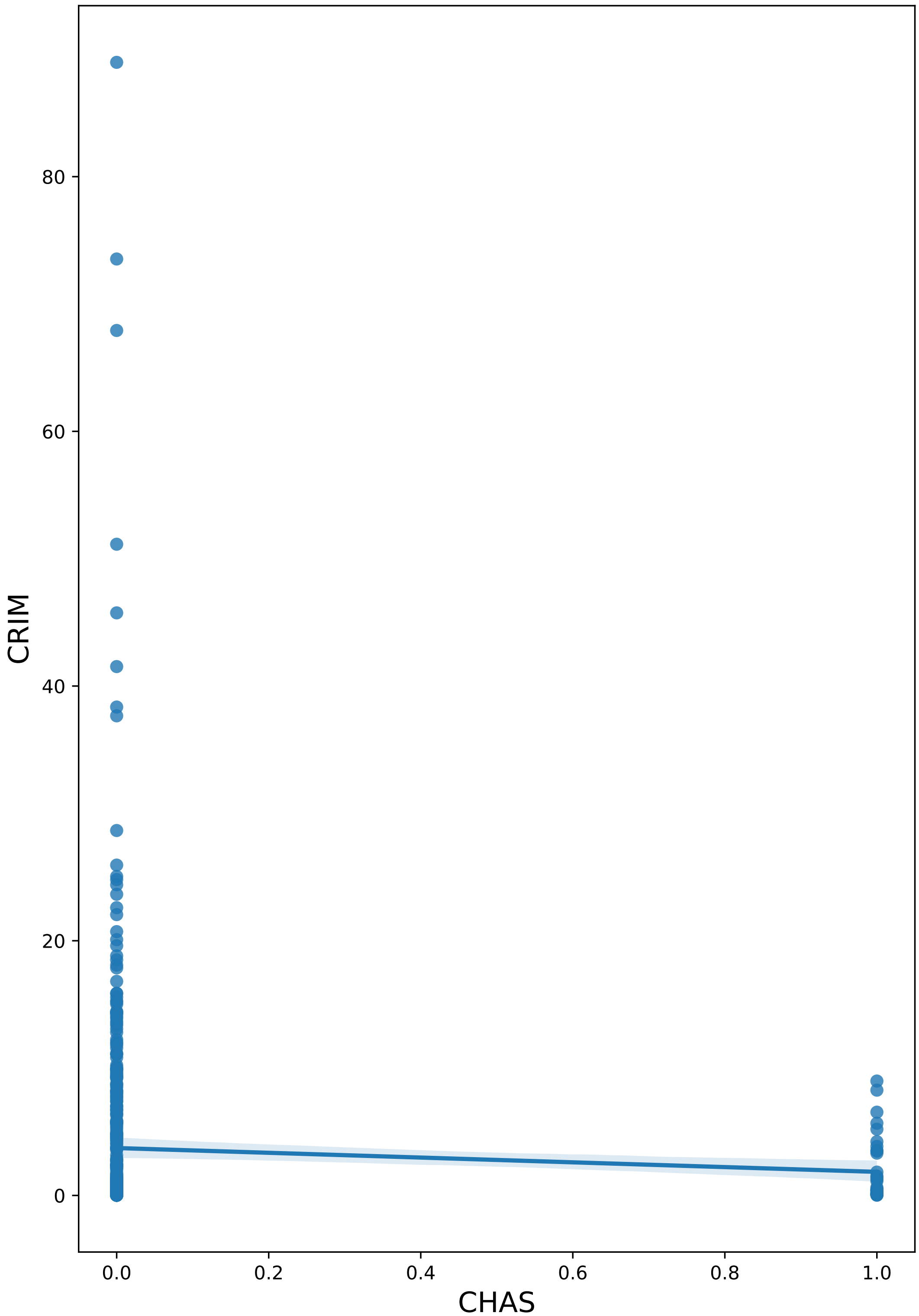
PLotting 5 predictors

```python
# Loop
for x in independent_var[:5]:
```

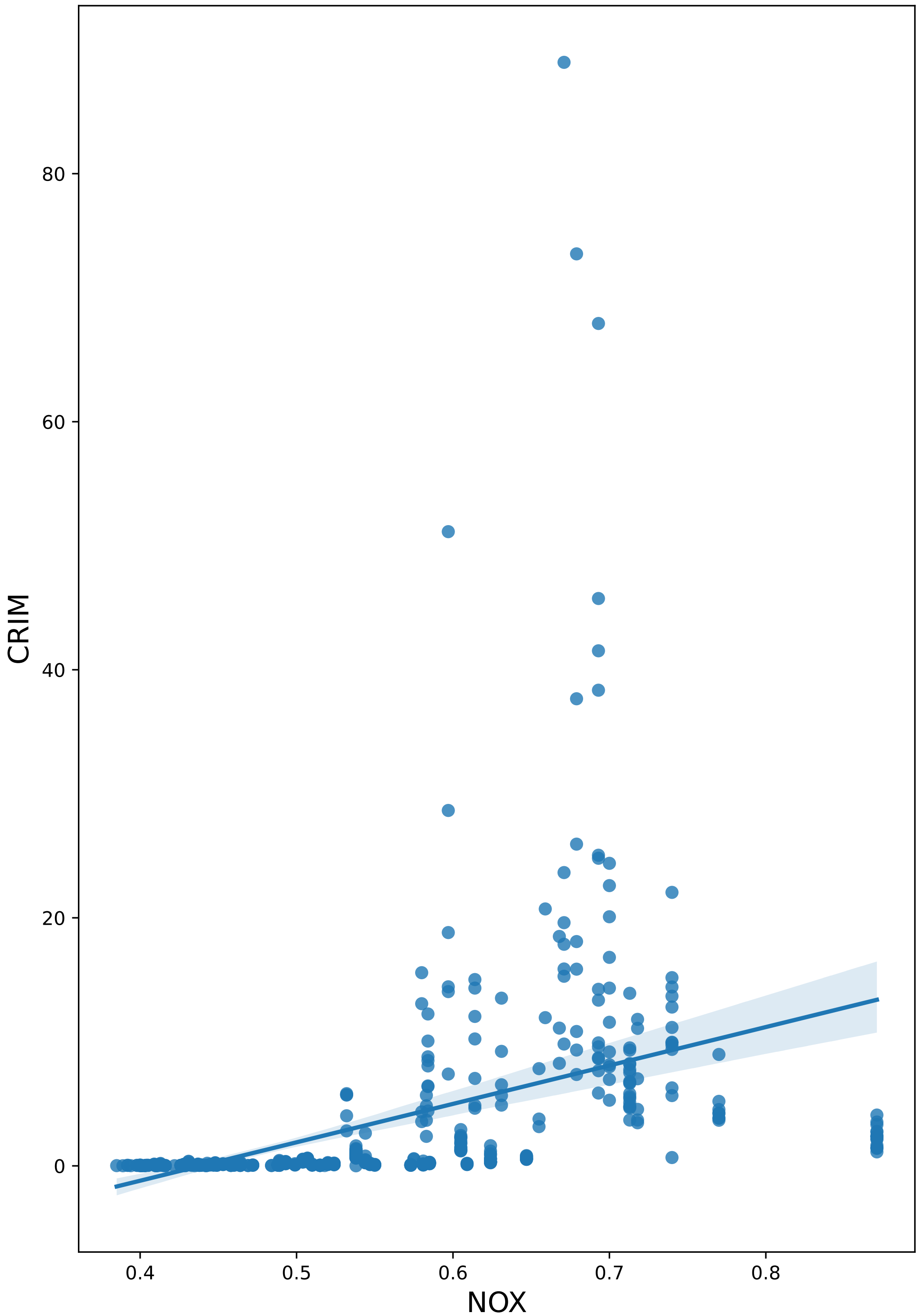
```
fig, ax = plt.subplots(figsize=(8, 12))

# regression
sns.regplot(x=x, y='CRIM', data=boston_df, ax=ax)

ax.set_xlabel(x, fontsize=15)
ax.set_ylabel("CRIM", fontsize=15)

plt.show()
```

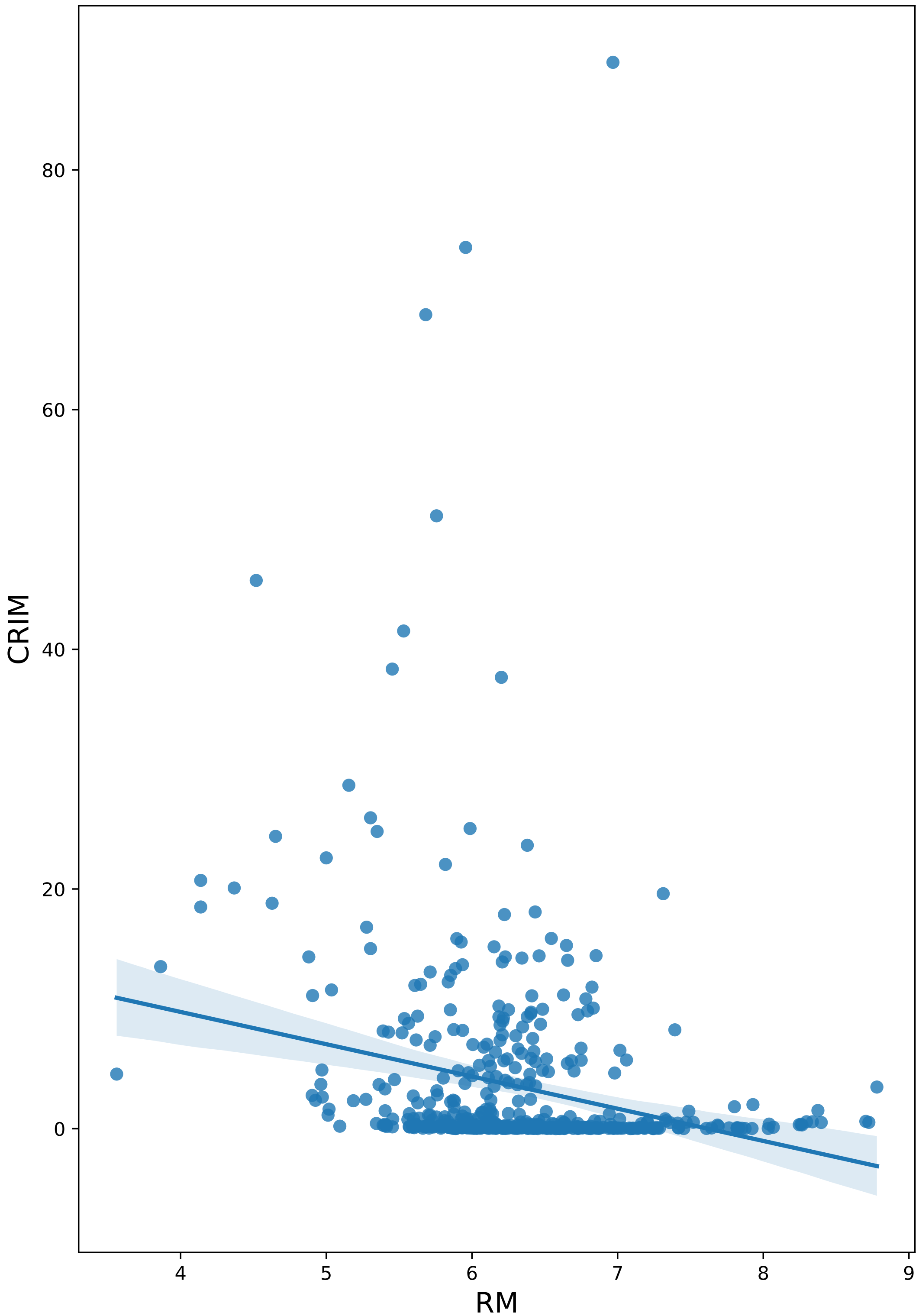

CRIM
80
60
40
20
0
0
20
40
60
80
100
ZN


CRIM
INDUS
0
20
40
60
80
0
5
10
15
20
25



80
60
40
20
0
CRIM
0.0
0.2
0.4
0.6
0.8
1.0
CHAS


80
60
40
20
0
CRIM
0.4
0.5
0.6
0.7
0.8
NOX



80
60
40
20
0
CRIM
4
5
6
7
8
9
RM

5.b.

```
mult_reg = smf.ols(
    "CRIM ~ AGE + B + CHAS + DIS+ INDUS + NOX + RM + RAD + ZN + TAX + PTRATIO
     ↪  + LSTAT + MDEV", data=boston_df).fit()

print(mult_reg.summary())
```

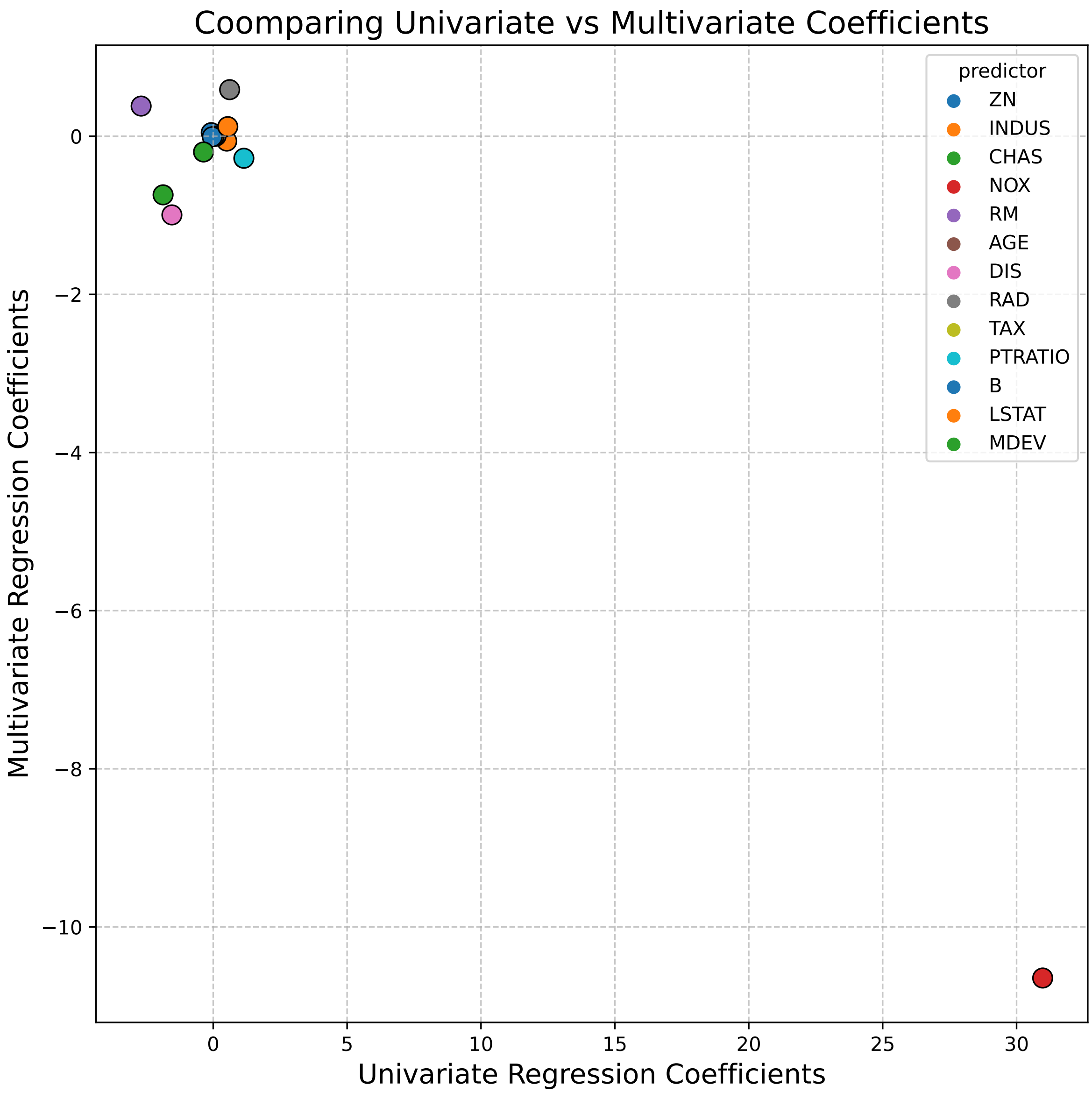
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   CRIM   R-squared:
0.448
Model:                            OLS   Adj. R-squared:
0.434
Method:                 Least Squares   F-statistic:
30.73
Date:                Thu, 23 Jan 2025   Prob (F-statistic):
2.04e-55
Time:                        23:16:30   Log-Likelihood:
-1655.7
No. Observations:                 506   AIC:
3339.
Df Residuals:                     492   BIC:
3399.
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
Intercept     17.4184      7.270      2.396      0.017       3.135
31.702
AGE            0.0020      0.018      0.112      0.911      -0.033
0.037
B             -0.0069      0.004     -1.857      0.064      -0.014
0.000
CHAS          -0.7414      1.186     -0.625      0.532      -3.071
1.588
DIS           -0.9950      0.283     -3.514      0.000      -1.551
-0.439
INDUS         -0.0616      0.084     -0.735      0.463      -0.226
0.103
```

```
NOX          -10.6455      5.301     -2.008      0.045     -21.061
-0.230
RM             0.3811      0.616      0.619      0.536      -0.829
1.591
RAD            0.5888      0.088      6.656      0.000       0.415
0.763
ZN             0.0449      0.019      2.386      0.017       0.008
0.082
TAX           -0.0037      0.005     -0.723      0.470      -0.014
0.006
PTRATIO       -0.2787      0.187     -1.488      0.137      -0.647
0.089
LSTAT          0.1213      0.076      1.594      0.112      -0.028
0.271
MDEV          -0.1992      0.061     -3.276      0.001      -0.319
-0.080
==============================================================================
Omnibus:                      662.271   Durbin-Watson:
1.515
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
82701.666
Skew:                           6.544   Prob(JB):
0.00
Kurtosis:                      64.248   Cond. No.
1.58e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.58e+04. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

The coefficients represent the estimated effect of a one unit change in each of the independent variables, holding all other predictors constant. The intercept is the per capita crime rate when all the other predictors are 0. We can reject the null hypothesis at the 5% significance level for the ff predictors which have a p-value less than .05.: DIS, NOX, ZN, RAD, and MDEV for having low p-values.

5.c. Create a plot displaying the univariate regression coefficients from Question (5a) on the x-axis, and the multiple regression coefficients from Question (5b) on the y-axis.

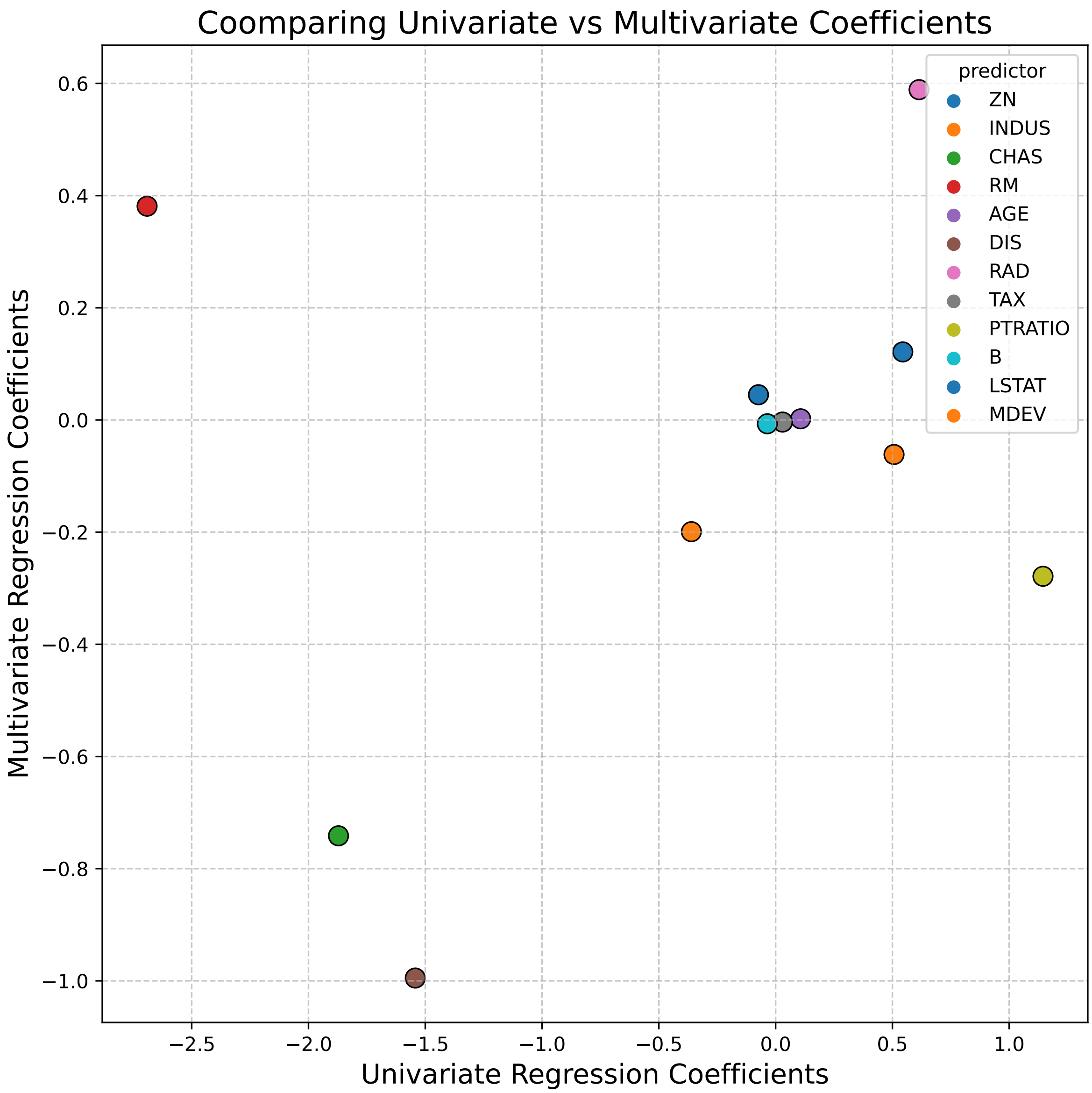
```python
# Univariate regression coefficients
coefs_df = pd.DataFrame({
    "predictor": results_simple_reg["predictor"],
    "uni_coefs": results_simple_reg["coefficient"]
})

# Add multivariate coefficients to the df
coefs_df["multi_coefs"] = mult_reg.params.loc[independent_var].values

coefs_df = coefs_df.reset_index(drop=True)

# Plotting
fig, ax = plt.subplots(figsize=(8, 8))
sns.scatterplot(
    x="uni_coefs",
    y="multi_coefs",
    hue="predictor",
    palette="tab10",
    data=coefs_df,
    ax=ax,
    s=100,
    edgecolor="black"
)

# Setlabels
ax.set_title(
    "Coomparing Univariate vs Multivariate Coefficients", fontsize=16)
ax.set_xlabel("Univariate Regression Coefficients", fontsize=14)
ax.set_ylabel("Multivariate Regression Coefficients", fontsize=14)
ax.grid(True, linestyle="--", alpha=0.7)

# Show the plot
plt.tight_layout()
plt.show()
```



source:https://stackoverflow.com/questions/42767489/add-legend-to-seaborn-point-plot
*asked ChatGPT: Please help me refine with color palette an sizing.Helped me choose the best pallete and sizing to use for htis

Most of the data is clustered at the upperleft corner of the graphm wiith NOX being the one outlier (a high univariate coef value, but low mutivariate coef value). It looks liek the coefficients do not match. To observe the other data points more closely, we can redo the graph, without NOX so it will zoom in,

```
fig, ax = plt.subplots(figsize=(8, 8))
sns.scatterplot(
    x="uni_coefs",
    y="multi_coefs",
    hue="predictor",
    palette="tab10",
    data=coefs_df[coefs_df["predictor"]!="NOX"],
    ax=ax,
    s=100,
    edgecolor="black"
)

# Setlabels
ax.set_title(
    "Coomparing Univariate vs Multivariate Coefficients", fontsize=16)
ax.set_xlabel("Univariate Regression Coefficients", fontsize=14)
ax.set_ylabel("Multivariate Regression Coefficients", fontsize=14)
ax.grid(True, linestyle="--", alpha=0.7)

# Show the plot
plt.tight_layout()
plt.show()
```



Now, we can see that the data is actually spread out–there are both positive and negative values. Strangely, some values that are positive in the univariate regressio (PTRATIO and INDUS), are negative in the multivariate, and vice versa (RM and ZN).

5.d.

```
# Extracting adjusted R2 from earlier
linear_r2 = results_simple_reg[["predictor", "adjusted R-squared
↪  (single)"]].rename(
```

```
    columns={"adjusted R-squared (single)": "Adj R2 (Linear)"}
)

# Function for polynomial models and get adjusted R2
def fit_polynomial(data, predictor, response='CRIM'):
    # Fit polynomial model
    formula = f"{response} ~ {predictor} + I({predictor}**2) +
 ↪  I({predictor}**3)"
    model = smf.ols(formula, data=boston_df).fit()
    return model.rsquared_adj


# Fit polynomial models for each predictor
#empty list
polynomial_r2 = []
#loop and apply function
for predictor in linear_r2["predictor"]:
    adj_r2_poly = fit_polynomial(boston_df, predictor)
    polynomial_r2.append(
        {"predictor": predictor, "Adj R2 (Polynomial)": adj_r2_poly})

# Convert to DataFrame
polynomial_r2_df = pd.DataFrame(polynomial_r2)

# Merge  results into a table
comparison_df = linear_r2.merge(polynomial_r2_df, on="predictor")
comparison_df["Difference"] = (
   comparison_df["Adj R2 (Linear)"] - comparison_df["Adj R2 (Polynomial)"]
)

from tabulate import tabulate
print(tabulate(comparison_df, headers="keys", tablefmt="grid"))
```

```
+----+-------------+-------------------+-----------------------+--------------+
|    | predictor   |   Adj R2 (Linear) |   Adj R2 (Polynomial) |   Difference
|
+====+=============+===================+=======================+==============+
|  0 | ZN          |         0.0378783 |             0.0520163 |  -0.014138
|
+----+-------------+-------------------+-----------------------+--------------+
|  1 | INDUS       |         0.161937  |             0.252486  |  -0.0905491
|
```

2	CHAS	0.00107951	-0.000910111	0.00198962
3	NOX	0.172686	0.288145	-0.115459
4	RM	0.0464854	0.0628415	-0.0163561
5	AGE	0.121309	0.167472	-0.0461626
6	DIS	0.14111	0.271521	-0.130411
7	RAD	0.385704	0.391982	-0.00627877
8	TAX	0.334577	0.361	-0.0264232
9	PTRATIO	0.0812689	0.10718	-0.025911
10	B	0.140702	0.13884	0.00186292
11	LSTAT	0.202925	0.209716	-0.00679132
12	MDEV	0.147177	0.412562	-0.265385

source: https://www.geo.fu-berlin.de/en/v/soga-py/Basics-of-statistics/Linear-Regression/Polynomial-Regression/Polynomial-Regression—An-example/index.html

table source: https://www.datacamp.com/tutorial/python-tabulate

For all non-indicator predictors, the adjusted R-squared is higher in the polynomial vs the simple regression, except for B. This means that the polynomial regression explains more of the variation in the model, thus fits the data better. It also means that the additional coomplexity in the model did generally add value.