STAT4255 Homework 2

Jack Bienvenue

2024-09-24

Hello! In this assignment, we will be working on several exercises from the ISLP book:

Let's begin by importing our modules:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import random
from statsmodels.stats.outliers_influence \
    import variance_inflation_factor as VIF
from statsmodels.stats.anova import anova_lm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
                         summarize, poly)
```

Exercise 1 - 3.3

Suppose we have a data set with five predictors:

- **X1** = GPA
- **X2** = IQ
- **X3** = Level (1 for College and 0 for High School)
- **X4** = Interaction between GPA and IQ
- **X5** = Interaction between GPA and Level

The response is starting salary after graduation (in thousands of dollars).

Suppose we use least squares to fit the model, and get:

- $\beta^0 = 50$

- $\beta^1 = 20$
- $\beta^2 = 0.07$
- $\beta^3 = 35$
- $\beta^4 = 0.01$
- $\beta^5 = -10$

(a) Which answer is correct, and why?

- i. For a fixed value of IQ and GPA, high school graduates earn more, on average, than college graduates.
- ii. For a fixed value of IQ and GPA, college graduates earn more, on average, than high school graduates.
- 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.
- iv. For a fixed value of IQ and GPA, college graduates earn more, on average, than high school graduates provided that the GPA is high enough.

For this question, we will select answer **ii**. All else equal, our model indicates that a college graduate could be expected to earn an additional \$35,000 a year as compared to a high school graduate.

(b) Predict the salary of a college graduate with IQ of 110 and a GPA of 4.0.

Let's apply these conditions to our model.

$$Y = 20 * 4.0 + 110 * 0.07 + 1 * 35 + 0.01 * 100 * 4.0 + -10 * 4.0 * 1$$

This would work out to a prediction of an annual salary of \$86,700.

(c) True or false: Since the coefficient for the GPA/IQ interaction term is very small, there is very little evidence of an interaction effect. Justify your answer.

False. The magnitude of the interaction term does not determine whether an interaction effect is present. To examine this, we would instead have to look towards the p-value and whether it meets our specifications. Regardless, the small magnitude means that the term won't influence predictions much.

Exercise 2 - 3.4

I collect a set of data (n = 100 observations) containing a single predictor and a quantitative response. I then fit a linear regression model to the data, as well as a separate cubic regression, i.e.

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \varepsilon.$$

(a) Suppose that the true relationship between X and Y is linear, i.e. $Y = \beta_0 + \beta_1 X + \epsilon$. Consider the training residual sum of squares (RSS) for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer.

While it is not intuitive, if we have truly linear relationship and fit a cubic model, we may still actually find a reduction in RSS versus the linear model. Why is this? The additional terms in the model may allow for closer fitting of the available data. Because we know that the nature of the relationship is linear, though, this would be *overfitting* behavior and could harm predictions, especially predictions past the extent of the data.

(b) Answer (a) using test rather than training RSS.

To complete this section of the exercise, we will need to make a function to generate data which follows a linear relationship. Let's also generate functions for finding linear and cubic models and their respective RSS's. This will be handy in our response to the **(d)** as well.

```
def generate_linear_data(n=100, slope=1, intercept=0, noise_var = 1):

    X = np.random.uniform(1,10,n)
    # Normally distributed noise
    noise = np.random.normal(0, noise_var, n)

    Y = slope * X + intercept + noise

    # df generation

    data = pd.DataFrame({'X': X, 'Y': Y})

    return data

#Let's now define modeling fxns:
def fit_linear_model(data):
```

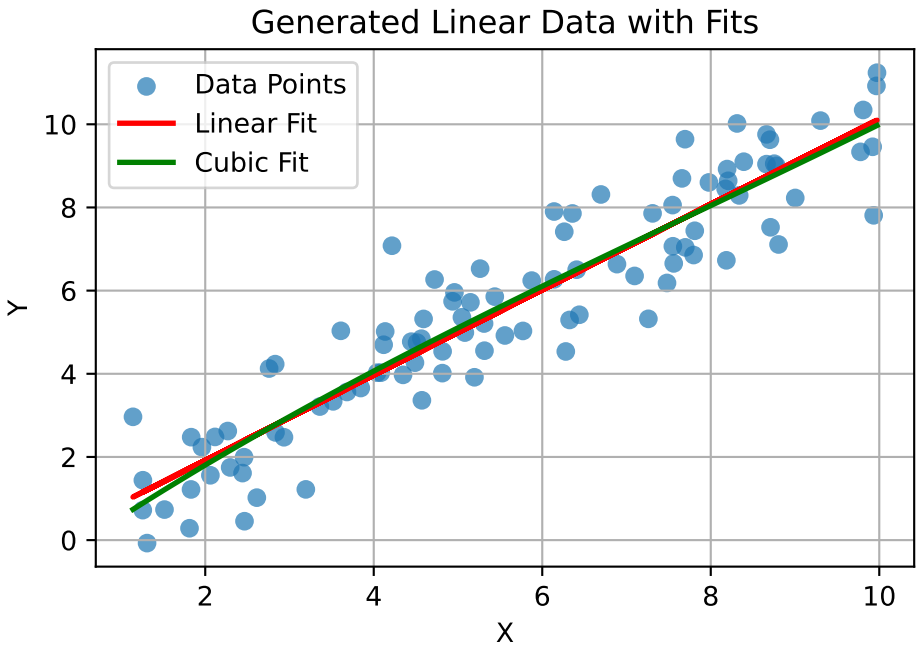
```
    X = sm.add_constant(data['X'])  # Adds intercept term
    model = sm.OLS(data['Y'], X).fit()
    predictions = model.predict(X)
    rss = np.sum((data['Y'] - predictions) ** 2)
    return rss, model

def fit_cubic_model(data):
    data['X_squared'] = data['X'] ** 2
    data['X_cubed'] = data['X'] ** 3
    X = sm.add_constant(data[['X', 'X_squared', 'X_cubed']])
        # Adds intercept term ^

    model = sm.OLS(data['Y'], X).fit()
    predictions = model.predict(X)
    rss = np.sum((data['Y'] - predictions) ** 2)
    return rss, model
```

Time to test:

```
# Run linear data generating fxn:
data = generate_linear_data()

lin_rss, lin_model = fit_linear_model(data)
cubic_rss, cubic_model = fit_cubic_model(data)

print(f"Linear Model RSS: {lin_rss:.2f}")
print(f"Cubic Model RSS: {cubic_rss:.2f}")

## Plotting:
# Data
plt.scatter(data['X'], data['Y'], alpha=0.7, label='Data Points')

# Linear Model
plt.plot(data['X'], lin_model.predict(sm.add_constant(data['X'])),
    color='red', label='Linear Fit', linewidth=2)

# Cubic Model
X_sorted = np.sort(data['X'])
Y_cubic = cubic_model.predict(sm.add_constant(pd.DataFrame({'X': X_sorted,
    'X_squared': X_sorted**2, 'X_cubed': X_sorted**3})))
plt.plot(X_sorted, Y_cubic, color='green', label='Cubic Fit', linewidth=2)
```

```
plt.title("Generated Linear Data with Fits")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.grid()
plt.show()
```

```
Linear Model RSS: 100.10
Cubic Model RSS: 99.10
```



We find our expected result after experimentation.

(c) Suppose that the true relationship between X and Y is not linear, but we don't know how far it is from linear. Consider the training RSS for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer.

Here we would still expect that the RSS for the cubic model would be lower. The nature of the cubic model allows it to fit itself more closely to a dataset because the additional terms can provide flexibility to move the model line in ways the linear model cannot.

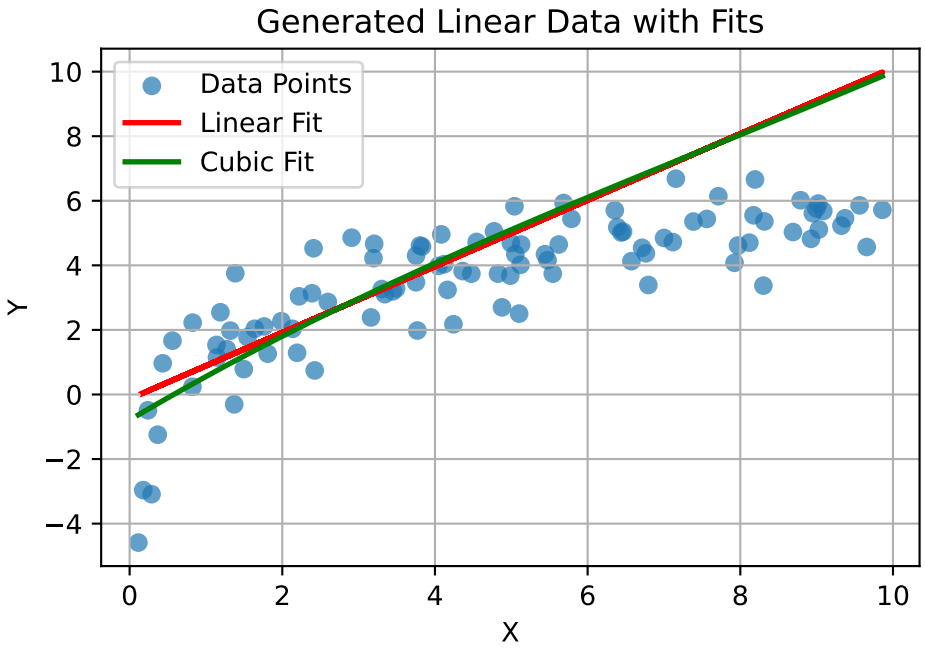
(d) Answer (c) using test rather than training RSS.

As an example here, let's use logarithmic data as a substitute for linear data. Let's make a generating function and test it:

```
def generate_log_data(n=100, a=2, b=1, noise_variance=1):
    # Generate predictor values (start after zero to prevent log(0))
    X = np.random.uniform(0.1, 10, n)

    noise = np.random.normal(0, noise_variance, n)
    Y = a * np.log(X) + b + noise

    # DataFrame synthesis:
    data = pd.DataFrame({'X': X, 'Y': Y})

    return data

log_data = generate_log_data()

lin_rss, lin_model = fit_linear_model(data)
cubic_rss, cubic_model = fit_cubic_model(data)

print(f"Linear Model RSS: {lin_rss:.2f}")
print(f"Cubic Model RSS: {cubic_rss:.2f}")

## Plotting:
# Data
plt.scatter(log_data['X'], log_data['Y'], alpha=0.7, label='Data Points')

# Linear Model
plt.plot(log_data['X'], lin_model.predict(sm.add_constant(log_data['X'])),
    color='red', label='Linear Fit', linewidth=2)

# Cubic Model
X_sorted = np.sort(log_data['X'])
Y_cubic = cubic_model.predict(sm.add_constant(pd.DataFrame({'X': X_sorted,
    'X_squared': X_sorted**2, 'X_cubed': X_sorted**3})))
plt.plot(X_sorted, Y_cubic, color='green', label='Cubic Fit', linewidth=2)

plt.title("Generated Linear Data with Fits")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
```

```
plt.grid()
plt.show()
```

```
Linear Model RSS: 100.10
Cubic Model RSS: 99.10
```



Again, we find by testing that we attain our hypothesized result.

Exercise 3 - 3.10 (a-g)

This question should be answered using the Carseats data set.

Let's begin by importing the dataset:

```
Carseats = load_data('Carseats') # Using built-in ISLP functionality
```

Let's take a peek at what the dataset looks like:

```
print(Carseats.head())
```

```
   Sales  CompPrice  Income  Advertising  Population  Price ShelveLoc  Age  \
0   9.50        138      73           11         276    120       Bad   42
1  11.22        111      48           16         260     83      Good   65
2  10.06        113      35           10         269     80    Medium   59
3   7.40        117     100            4         466     97    Medium   55
4   4.15        141      64            3         340    128       Bad   38

   Education Urban   US
0         17   Yes  Yes
1         10   Yes  Yes
2         12   Yes  Yes
3         14   Yes  Yes
4         13   Yes   No
```

(a) Fit a multiple regression model to predict Sales using Price, Urban, and US.

Let's build this out:

```python
## Cleaning
# Categorical Variable handling
Carseats['Urban'] = Carseats['Urban'].map({'Yes': 1, 'No': 0})
Carseats['US'] = Carseats['US'].map({'Yes': 1, 'No': 0})

# Define responses and predictor
X = Carseats[['Price', 'Urban', 'US']]
y = Carseats['Sales']

# Add intercept
X = sm.add_constant(X)

# Fit MLR model
model = sm.OLS(y, X).fit()

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.239
Model:                            OLS   Adj. R-squared:                  0.234
Method:                 Least Squares   F-statistic:                     41.52
Date:                Wed, 25 Sep 2024   Prob (F-statistic):           2.39e-23
```

```
Time:                        20:54:20   Log-Likelihood:                -927.66
No. Observations:                 400   AIC:                             1863.
Df Residuals:                     396   BIC:                             1879.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.0435      0.651     20.036      0.000      11.764      14.323
Price         -0.0545      0.005    -10.389      0.000      -0.065      -0.044
Urban         -0.0219      0.272     -0.081      0.936      -0.556       0.512
US             1.2006      0.259      4.635      0.000       0.691       1.710
==============================================================================
Omnibus:                        0.676   Durbin-Watson:                   1.912
Prob(Omnibus):                  0.713   Jarque-Bera (JB):                0.758
Skew:                           0.093   Prob(JB):                        0.684
Kurtosis:                       2.897   Cond. No.                         628.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

(b) Provide an interpretation of each coefficient in the model. Be careful—some of the variables in the model are qualitative!

- Price
  - Based upon our model, we find that for every dollar increase in price of the carseat, we observe a decline in sales by 0.0545, on average, in this linear model. Our units for sales are thousands of unit sales, so for each increase in price by $1, we would expect a reduction of 54.5 sales.
- Urban
  - *Urban* indicates whether the retailer for the car seats are in an urban or rural location. This variable is encoded in binary fasion. We find that our model estimates that being in an urban location would reduce the expected sales by 21.9 units versus a rural area, all else equal. However, by any reasonable benchmark for p-value, this variable would not be considered significant.
- US
  - *US* indicates whether the retailer for the car seats is in the US. We find that our estimates that being in the US increases sales by 1200.6 units on average in this

linear model versus not being in the US. This could be due to many things, perhaps regulations requiring car seats or perhaps attitudes towards car seats that make it more likely for parents in the United States to buy them.

(c) Write out the model in equation form, being careful to handle the qualitative variables properly.

We can write out our formula for our model in the following way:

$$Y = \beta_0 + \beta_1 * X_1 + \beta_2 * X_2 + \beta_3 * X_3$$

where:

- β_0 represents the intercept
- β_1 represents the coefficient for predictor 'Price'
- β_2 represents the coefficient for predictor 'Urban'
- β_3 represents the coefficient for predictor 'US'
- X_1 represents the price, in US dollars, of the car seat
- X_2 represents the urban/rural status
  - $X_2 = 1$ indicates urban
  - $X_2 = 0$ indicates rural
- X_3 represents the domestic/foreign selling status
  - $X_3 = 1$ indicates selling in US
  - $X_3 = 0$ indicates selling outside US

(d) For which of the predictors can you reject the null hypothesis H0 : j =0?

We can reject the null hypothesis that $H_0 : \beta_j = 0$ for coefficients β_0, β_1, and β_3. The p-values are less than 0.001, meaning that they would be rejected under any reasonable threshold.

(e) On the basis of your response to the previous question, fit a smaller model that only uses the predictors for which there is evidence of association with the outcome.

Let's fit the new model, this time excluding the "Urban" term:

```python
# Here, we can use our linear model generating function again,
# but we'll clip out the unnecessary column

newdata = Carseats.drop(columns=['Urban'])

X = newdata[['Price', 'US']]
X = sm.add_constant(X)
y = newdata['Sales']

model_drop_urban = sm.OLS(y, X).fit()

print(model_drop_urban.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.239
Model:                            OLS   Adj. R-squared:                  0.235
Method:                 Least Squares   F-statistic:                     62.43
Date:                Wed, 25 Sep 2024   Prob (F-statistic):           2.66e-24
Time:                        20:54:20   Log-Likelihood:                -927.66
No. Observations:                 400   AIC:                             1861.
Df Residuals:                     397   BIC:                             1873.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.0308      0.631     20.652      0.000      11.790      14.271
Price         -0.0545      0.005    -10.416      0.000      -0.065      -0.044
US             1.1996      0.258      4.641      0.000       0.692       1.708
==============================================================================
Omnibus:                        0.666   Durbin-Watson:                   1.912
Prob(Omnibus):                  0.717   Jarque-Bera (JB):                0.749
Skew:                           0.092   Prob(JB):                        0.688
Kurtosis:                       2.895   Cond. No.                         607.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

(f) How well do the models in (a) and (e) fit the data?

It appears that both models fit the data in a way that is very similar to one another. The coefficients are not changed very much from one model to another and R^2 is essentially the same for each model. Upon inspection of the models, it seems that they both have useful components, but that the included variables do not provide a holistic review of the factors that influence sales, since the whole model's R^2 value is around 24%.

(g) Using the model from (e), obtain 95% confidence intervals for the coefficient(s).

Conveniently, the 95% confidence intervals for the values of the coefficients are included in the printout. We can see for:

- β_0
  - The 95% confidence interval ranges from 11.790 to 14.271
- β_1
  - The 95% confidence interval ranges from -0.065 to -0.044
- β_3
  - The 95% confidence interval ranges from 0.692 to 1.708

Exercise 4 - 3.13

13. In this exercise you will create some simulated data and will fit simple linear regression models to it. Make sure to use the default random number generator with seed set to 1 prior to starting part (a) to ensure consistent results.

```
np.random.seed(1)
```

(a) Using the normal() method of your random number generator, create a vector, x, containing 100 observations drawn from a N(0,1) distribution. This represents a feature, X.

Let's find a way to do this:

```
x = np.random.normal(0, 1, 100) # N(0,1), 100 observations
```

(b) Using the normal() method, create a vector, eps, containing 100 observations drawn from a N(0,0.25) distribution—a normal distribution with mean zero and variance 0.25.

```
eps = np.random.normal(0, 0.25, 100)
```

(c) Using x and eps, generate a vector y according to the model Y =−1+0.5X+ . What is the length of the vector y? What are the values of 0 and 1 in this linear model?

```
y = np.zeros(100) # placeholders

ex4df = pd.DataFrame({'x': x, 'eps': eps, 'y': y})

# Now we have to calculate out y:
ex4df['y'] = -1 + 0.5 * ex4df['x'] + ex4df['eps']
# Check if this worked:
print(ex4df.head())

print(f'Length of y is: {len(ex4df["y"])}')
```

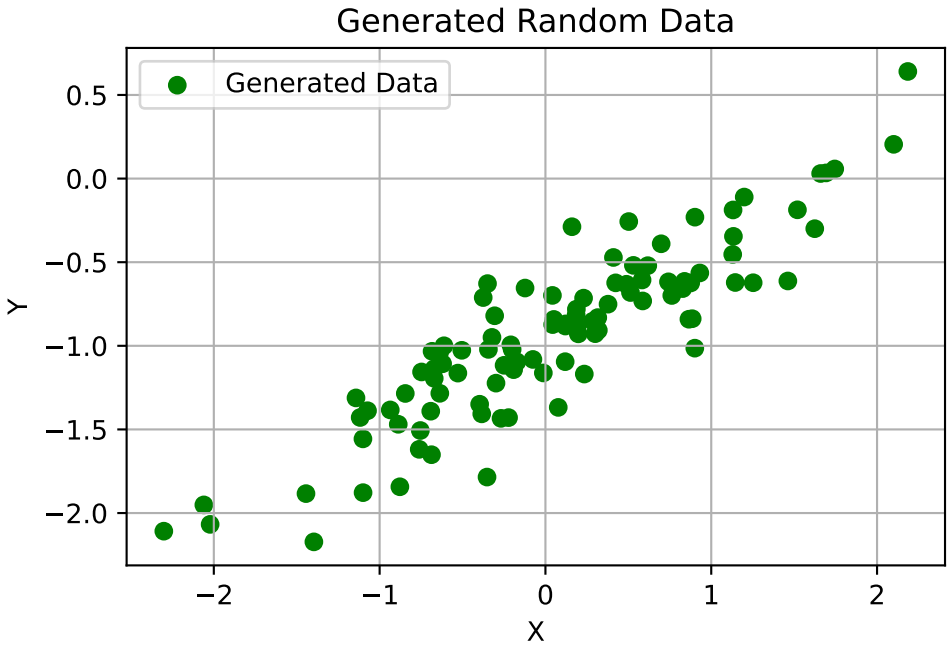
```
          x       eps         y
0  1.624345 -0.111782 -0.299609
1 -0.611756  0.306127 -0.999751
2 -0.528172  0.100873 -1.163213
3 -1.072969  0.148395 -1.388090
4  0.865408 -0.273728 -0.841024
Length of y is: 100
```

Based upon this linear model,

- $\beta_0 = -1$
- $\beta_1 = 0.5$

(d) Create a scatterplot displaying the relationship between x and y. Comment on what you observe.

```
plt.scatter(ex4df['x'], ex4df['y'], color='green', label='Generated Data')

plt.title("Generated Random Data")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.grid()
plt.show()
```



It appears as if a linear relationship is built into the way we designed our program. This makes sense, as y is dependent on randomly generated x in linear fashion in our model.

(e) Fit a least squares linear model to predict y using x. Comment on the model obtained. How do ˆ0 and ˆ1 compare to 0 and 1?

```
X = ex4df[['x']]
X = sm.add_constant(X)
y = ex4df['y']

ex4model1 = sm.OLS(y, X).fit()
```

```
print(ex4model1.summary())
```

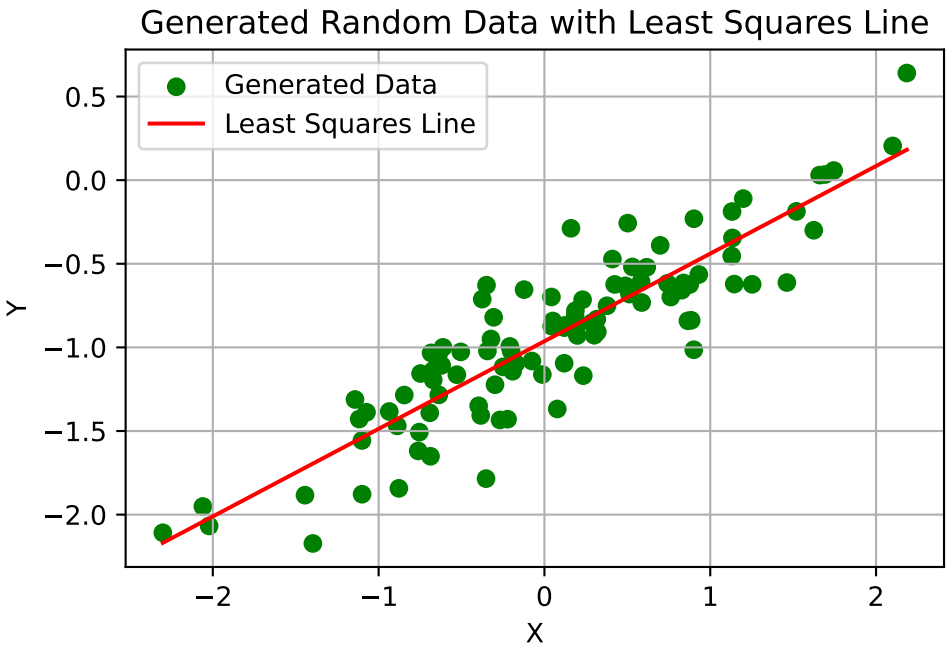
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.800
Model:                            OLS   Adj. R-squared:                  0.798
Method:                 Least Squares   F-statistic:                     391.4
Date:                Wed, 25 Sep 2024   Prob (F-statistic):           5.39e-36
Time:                        20:54:21   Log-Likelihood:                 4.1908
No. Observations:                 100   AIC:                            -4.382
Df Residuals:                      98   BIC:                            0.8288
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.9632      0.023    -40.999      0.000      -1.010      -0.917
x              0.5239      0.026     19.783      0.000       0.471       0.576
==============================================================================
Omnibus:                        0.898   Durbin-Watson:                   2.157
Prob(Omnibus):                  0.638   Jarque-Bera (JB):                0.561
Skew:                          -0.172   Prob(JB):                        0.755
Kurtosis:                       3.127   Cond. No.                         1.15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The values we get are:

- $\hat{\beta}_0 = -0.9632$
  - versus $\beta_0 = -1$
- $\hat{\beta}_1 = 0.5239$
  - versus $\beta_1 = 0.5$

(f) Display the least squares line on the scatterplot obtained in (d). Draw the population regression line on the plot, in a different color. Use the legend() method of the axes to create an appropriate legend.

```
plt.scatter(ex4df['x'], ex4df['y'], color='green', label='Generated Data')

# least squares line
x = ex4df['x']
y = ex4df['y']
m, b = np.polyfit(x, y, 1)  # m = slope, b = intercept

# Points for the least squares line to follow
x_line = np.linspace(x.min(), x.max(), 100)
y_line = m * x_line + b

# Plot the least squares line
plt.plot(x_line, y_line, color='red', label='Least Squares Line')

# Add titles and labels
plt.title("Generated Random Data with Least Squares Line")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.grid()
plt.show()
```



(g) Now fit a polynomial regression model that predicts y using x and x2. Is there evidence that the quadratic term improves the model fit? Explain your answer.

```
# Let's just write a function to return to later:
def fit_quadratic_model(data):
    data['X_squared'] = data['X'] ** 2
    X = sm.add_constant(data[['X', 'X_squared']])  # Adds intercept term
    model = sm.OLS(data['Y'], X).fit()
    predictions = model.predict(X)
    rss = np.sum((data['Y'] - predictions) ** 2)
    return rss, model

#To allow fxn to handle our data:
ex4df.rename(columns={'x': 'X'}, inplace=True)
ex4df.rename(columns={'y': 'Y'}, inplace=True)

# Model construction
quad_rss, quad_model = fit_quadratic_model(ex4df) ###

print(quad_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.800
Model:                            OLS   Adj. R-squared:                  0.796
Method:                 Least Squares   F-statistic:                     193.8
Date:                Wed, 25 Sep 2024   Prob (F-statistic):           1.32e-34
Time:                        20:54:21   Log-Likelihood:                 4.2077
No. Observations:                 100   AIC:                            -2.415
Df Residuals:                      97   BIC:                             5.400
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.9663      0.029    -33.486      0.000      -1.024      -0.909
X              0.5234      0.027     19.582      0.000       0.470       0.576
X_squared      0.0039      0.021      0.181      0.856      -0.038       0.046
==============================================================================
Omnibus:                        0.893   Durbin-Watson:                   2.152
Prob(Omnibus):                  0.640   Jarque-Bera (JB):                0.552
Skew:                          -0.170   Prob(JB):                        0.759
Kurtosis:                       3.132   Cond. No.                         2.10
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The p-value for the squared term is extremely high. By no criteria would we accept this term into the model. Because of this, we can say that the inclusion of X^2 into the model is not helpful, regardless of whether it improves fit. Including unnecessary higher order polynomial terms like this can lead to overfitting.

(h) Repeat (a)–(f) after modifying the data generation process in such a way that there is less noise in the data. The model (3.39) should remain the same. You can do this by decreasing the variance of the normal distribution used to generate the error term in (b). Describe your results.

I will elect not to rewrite all the code for brevity and instead just report on the changes.

For this problem, I changed the variance to 0.05.

After executing this, I observed the following:

```
low_variance_results = {
    "Variable": ["const", "x"],
    "Coefficient": [-0.9926, 0.5048],
    "Std. Error": [0.005, 0.005],
    "t": [-211.252, 95.309],
    "P > |t|": [0.000, 0.000],
    "[0.025": [-1.002, 0.494],
    "0.975]": [-0.983, 0.515]
}

low_var_results_df = pd.DataFrame(low_variance_results)

print(low_var_results_df)
```

```
  Variable  Coefficient  Std. Error        t  P > |t|  [0.025  0.975]
0    const      -0.9926       0.005 -211.252      0.0  -1.002  -0.983
1        x       0.5048       0.005   95.309      0.0   0.494   0.515
```

As compared to the original model, we observe slightly lower expected values for the coefficients, a lower standard error for both coefficients, and a contraction of the confidence intervals.

(i) Repeat (a)–(f) after modifying the data generation process in such a way that there is more noise in the data. The model (3.39) should remain the same. You can do this by increasing the variance of the normal distribution used to generate the error term in (b). Describe your results.

For this problem, I changed the variance to 0.5.

After executing this, I observed the following:

```
high_variance_results = {
    "Variable": ["const", "x"],
    "Coefficient": [-0.9265, 0.5477],
    "Std. Error": [0.047, 0.053],
    "t": [-19.717, 10.342],
    "P > |t|": [0.000, 0.000],
    "[0.025": [-1.020, 0.443],
    "0.975]": [-0.833, 0.653]
}

high_var_results_df = pd.DataFrame(high_variance_results)
```

```
print(high_var_results_df)
```

```
  Variable  Coefficient  Std. Error       t  P > |t|  [0.025  0.975]
0    const      -0.9265       0.047 -19.717      0.0  -1.020  -0.833
1        x       0.5477       0.053  10.342      0.0   0.443   0.653
```

As compared to the original model, we observe slightly higher expected values for the coefficients, a larger standard error for both coefficients, and an expansion of the confidence intervals.

(j) What are the confidence intervals for 0 and 1 based on the original data set, the noisier data set, and the less noisy data set? Comment on your results.

Confidence intervals for: - β_0: - Original data set: - -1.010 to -0.917 - Noiser data set: - -1.020 to -0.833 - Less noisy data set: - -1.002 to -0.983 - β_1 - Original data set: - 0.471 to 0.576 - Noiser data set: - 0.443 to 0.653 - Less noisy data set: - 0.494 to 0.515

As observed in the previous question, lower variance leads to a contraction of the interval, and high variance leads to expansion of it.

Exercise 5 - 3.15

This problem involves the Boston data set, which we saw in the lab for this chapter. We will now try to predict per capita crime rate using the other variables in this data set. In other words, per capita crime rate is the response, and the other variables are the predictors.

Import dataset:

```
Boston_df = load_data('Boston')
print(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

   lstat  medv
0   4.98  24.0
```

```
1   9.14  21.6
2   4.03  34.7
3   2.94  33.4
4   5.33  36.2
```

(a) For each predictor, fit a simple linear regression model to predict the response. Describe your results. In which of the models is there a statistically significant association between the predictor and the response? Create some plots to back up your assertions.

Creating 12 separate SLR models will get tedious so let's handle this more efficiently:

```
response_variable = 'crim'
predictors = Boston_df.columns.drop(response_variable)

results = {}

for predictor in predictors:
    X = Boston_df[[predictor]]  # Selecting the predictor
    y = Boston_df[response_variable]  # Selecting the response variable

    # Adding a constant term for the intercept
    X = sm.add_constant(X)

    # Fitting the model
    model = sm.OLS(y, X).fit()

    # Storing the results
    results[predictor] = {
        'coef': model.params,
        'pvalues': model.pvalues,
        'summary': model.summary()
    }
    #print(f"Summary for predictor '{predictor}':")
    #print(model.summary())
    #print("\n" + "-" * 80 + "\n")
```

I omitted printing the outputs for brevity. In summary, we find significance for all predictors except for 'chas'.

- 'zn'

  - Description: Proportion of residential land zoned for lots over 25,000 sq.ft.
  - p-value < 0.000
  - Coefficient on linear term: -0.0739

- 'indus'
  - Description: Proportion of non-retail business acres per town.
  - p-value < 0.000
  - Coefficient on linear term: 0.5098

- 'chas'
  - Description: Charles River dummy variable (=1 if tract bounds river; 0 otherwise)
  - p-value $= 0.209$
  - Coefficient on linear term: -1.8928

- 'nox'
  - Description: Nitrogen oxides concentration (parts per 10 million).
  - p-value < 0.000
  - Coefficient on linear term: 31.2485

- 'rm'
  - Description: Average number of rooms per dwelling.
  - p-value < 0.000
  - Coefficient on linear term: -2.6841

- 'age'
  - Description: Proportion of owner-occupied units built prior to 1940.
  - p-value < 0.000
  - Coefficient on linear term: 0.1078

- 'dis'
  - Description: Weighted mean of distances to five Boston employment centres.
  - p-value < 0.000
  - Coefficient on linear term: -1.5509

- 'rad'
  - Description: Index of accessibility to radial highways.
  - p-value < 0.000
  - Coefficient on linear term: 0.6179

- 'tax'
  - Description: Full-value property-tax rate per $10,000.
  - p-value < 0.000

  - Coefficient on linear term: 0.0297

- 'ptratio'
  - Description: Pupil-teacher ratio by town.
  - p-value < 0.000
  - Coefficient on linear term: 1.1520

- 'lstat'
  - Description: Lower status of the population (percent).
  - p-value < 0.000
  - Coefficient on linear term: 0.5488

- 'medv'
  - Description: Median value of owner-occupied homes in $1000s.
  - p-value < 0.000
  - Coefficient on linear term: -0.3632

(b) Fit a multiple regression model to predict the response using all of the predictors. Describe your results. For which predictors can we reject the null hypothesis H0 : j = 0?

We'll fit this model in the following way:

```
# Let's start by designing the predictors and responses:
X = Boston_df[['zn','indus','chas','nox','rm',
    'age','dis','rad','tax','ptratio','lstat','medv']]
y = Boston_df['crim']

# Putting together data for significant predictors only
sig_boston_df = pd.concat([X, y], axis=1)

X = sm.add_constant(X)
ex5b_model = sm.OLS(y, X).fit()

results = {
    'coef': ex5b_model.params,
    'pvalues': ex5b_model.pvalues,
    'summary': ex5b_model.summary()
}

print(ex5b_model.summary())
```

OLS Regression Results

Dep. Variable:	crim	R-squared:	0.449
Model:	OLS	Adj. R-squared:	0.436
Method:	Least Squares	F-statistic:	33.52
Date:	Wed, 25 Sep 2024	Prob (F-statistic):	2.03e-56
Time:	20:54:21	Log-Likelihood:	-1655.4
No. Observations:	506	AIC:	3337.
Df Residuals:	493	BIC:	3392.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	13.7784	7.082	1.946	0.052	-0.136	27.693
zn	0.0457	0.019	2.433	0.015	0.009	0.083
indus	-0.0584	0.084	-0.698	0.486	-0.223	0.106
chas	-0.8254	1.183	-0.697	0.486	-3.150	1.500
nox	-9.9576	5.290	-1.882	0.060	-20.351	0.436
rm	0.6289	0.607	1.036	0.301	-0.564	1.822
age	-0.0008	0.018	-0.047	0.962	-0.036	0.034
dis	-1.0122	0.282	-3.584	0.000	-1.567	-0.457
rad	0.6125	0.088	6.997	0.000	0.440	0.784
tax	-0.0038	0.005	-0.730	0.466	-0.014	0.006
ptratio	-0.3041	0.186	-1.632	0.103	-0.670	0.062
lstat	0.1388	0.076	1.833	0.067	-0.010	0.288
medv	-0.2201	0.060	-3.678	0.000	-0.338	-0.103

Omnibus:	663.436	Durbin-Watson:	1.516
Prob(Omnibus):	0.000	Jarque-Bera (JB):	80856.852
Skew:	6.579	Prob(JB):	0.00
Kurtosis:	63.514	Cond. No.	1.24e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.24e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Assuming that the question seeks an $\alpha = 0.05$, we can reject the null hypothesis which stated $\beta_j = 0$ for: - zn - Proportion of residential land zoned for lots over 25,000 sq.ft. - dis - Weighted mean of distances to five Boston employment centres. - rad - Index of accessibility to radial highways. - medv - Median value of owner-occupied homes in $1000s.

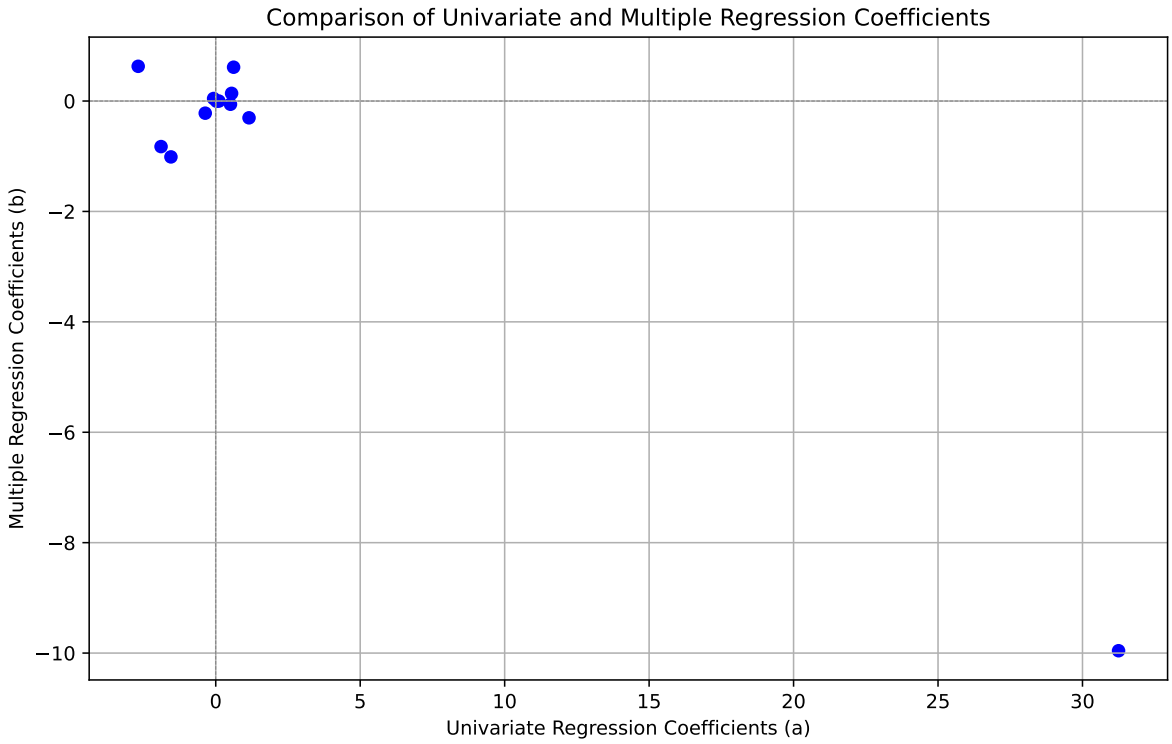
(c) How do your results from (a) compare to your results from (b)? Create a plot displaying the univariate regression coefficients from (a) on the x-axis, and the multiple regression coefficients from (b) on the y-axis. That is, each predictor is displayed as a single point in the plot. Its coefficient in a simple linear regression model is shown on the x-axis, and its coefficient estimate in the multiple linear regression model is shown on the y-axis.

We find that we select far fewer variables based upon our MLR versus the analysis of distinct SLRs.

Let's plot this out:

```
a_coefs = pd.Series([
    -0.0739,  # zn
    0.5098,   # indus
    -1.8928,  # chas
    31.2485,  # nox
    -2.6841,  # rm
    0.1078,   # age
    -1.5509,  # dis
    0.6179,   # rad
    0.0297,   # tax
    1.1520,   # ptratio
    0.5488,   # lstat
    -0.3632   # medv
])

b_coefs = pd.Series([
    0.0457,   # zn
    -0.0584,  # indus
    -0.8254,  # chas
    -9.9576,  # nox
    0.6289,   # rm
    -0.0008,  # age
    -1.0122,  # dis
    0.6125,   # rad
    -0.0038,  # tax
    -0.3041,  # ptratio
    0.1388,   # lstat
    -0.2201   # medv
])

# Create the plot
```

```
plt.figure(figsize=(10, 6))
plt.scatter(a_coefs, b_coefs, color='blue')

# Add labels and title
plt.xlabel('Univariate Regression Coefficients (a)')
plt.ylabel('Multiple Regression Coefficients (b)')
plt.title('Comparison of Univariate and Multiple Regression Coefficients')
plt.axhline(0, color='grey', linestyle='--', lw=0.5)
plt.axvline(0, color='grey', linestyle='--', lw=0.5)

plt.grid()
plt.show()
```



(d) Is there evidence of non-linear association between any of the predictors and the response? To answer this question, for each predictor X, fit a model of the form Y = 0 + 1X+ 2X2 + 3X3 + .

Luckily for this problem, we defined a function earlier in this homework assignment to generate cubic MLR functions. Let's put that to use again. We'll combine this with the iterative

generation method we used for part (a).

```
# Assuming Boston_df is already defined
response_variable = 'crim'
predictors = Boston_df.columns.drop(response_variable)

results = {}

for predictor in predictors:
    X = Boston_df[[predictor]].copy()  # Make copy, no SettingWithCopyWarning
    y = Boston_df[response_variable]  # Selecting the response variable

    # Adding polynomial terms using .assign() for clarity
    X = X.assign(
        X_squared=X[predictor] ** 2,
        X_cubed=X[predictor] ** 3
    )
    X = sm.add_constant(X)  # Adds intercept term

    # Fitting the model
    model = sm.OLS(y, X).fit()

    # Storing the results
    results[predictor] = {
        'coef': model.params,
        'pvalues': model.pvalues,
        'summary': model.summary()
    }

    # Uncomment to print the summary for each predictor
    #print(f"Summary for predictor '{predictor}':")
    #print(model.summary())
    #print("\n" + "-" * 80 + "\n")
```

Again, I will cut print statements out of the document for brevity, but in summary, the following variables demonstrate p-values that could lead to rejection of null hypotheses with plausible thresholds. Note that for certain predictors, there was high collinearity observed between the powers of the data as predictors.

- 'zn' *Not significant for higher order terms*
- 'indus'
  - Description: Proportion of non-retail business acres per town.

  - (X^2)
    * p-value of coefficient < 0.000
  - (X^3)
    * p-value of coefficient < 0.000

- 'chas' *Not significant for higher order terms*
- 'nox'
  - Description: Nitrogen oxides concentration (parts per 10 million).
  - (X^2)
    * p-value of coefficient < 0.000
  - (X^3)
    * p-value of coefficient < 0.000

- 'rm' *Not significant for higher order terms*
- 'age'
  - Description: Proportion of owner-occupied units built prior to 1940.
  - (X^2)
    * p-value of coefficient $= 0.047$
  - (X^3)
    * p-value of coefficient $= 0.007$

- 'dis'
  - Description: Weighted mean of distances to five Boston employment centres.
  - (X^2)
    * p-value of coefficient < 0.000
  - (X^3)
    * p-value of coefficient < 0.000

- 'rad' *Not significant for higher order terms*
- 'tax' *Not significant for higher order terms*
- 'ptratio'
  - Description: Pupil-teacher ratio by town.
  - (X^2)
    * p-value of coefficient $= 0.004$
  - (X^3)
    * p-value of coefficient $= 0.006$

- 'lstat'
  - Description: Lower status of the population (percent).

  - (X^2)
    - p-value of coefficient = 0.065
  - (X^3)
    - p-value of coefficient = 0.130

- 'medv'
  - Description: Median value of owner-occupied homes in $1000s.
  - (X^2)
    - p-value of coefficient < 0.000
  - (X^3)
    - p-value of coefficient < 0.000