***Carseat Dataset Analysis***

10 a)

import pandas as pd

import statsmodels.api as sm

#load the dataset

df=pd.read\_csv("https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/ISLR/Carseats.csv")

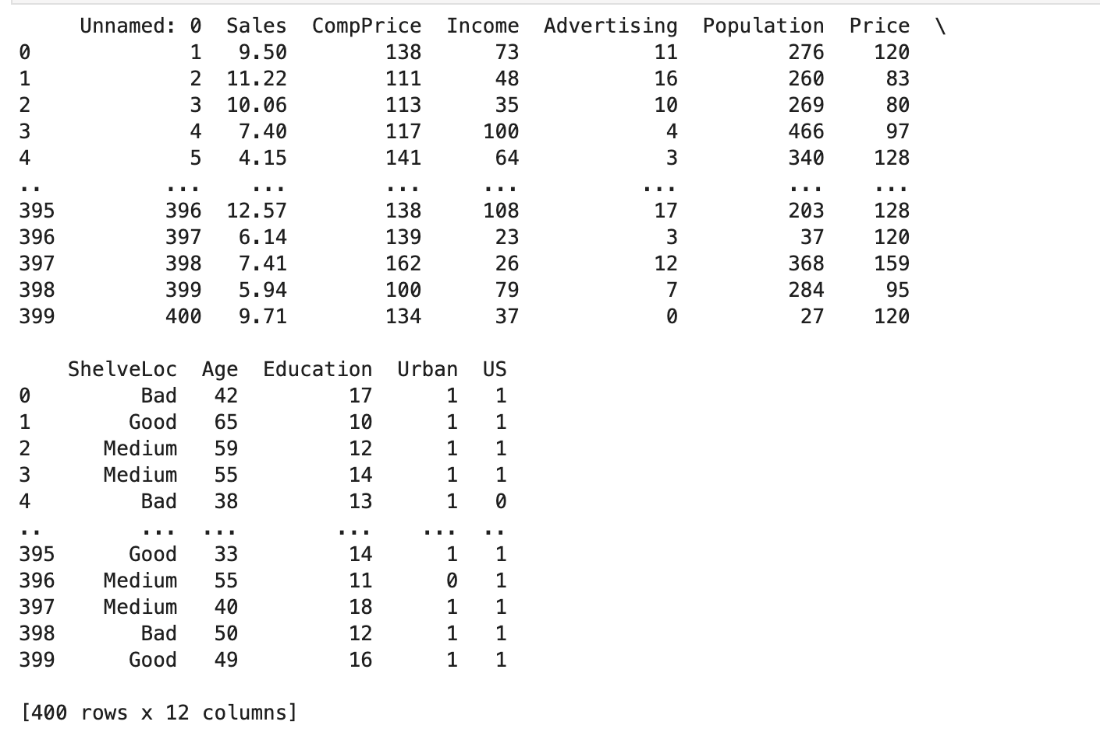
#define the predictor variables and the response variable

print(df)

df['Urban'].replace(['Yes','No'],[1,0],inplace=True)

df['US'].replace(['Yes','No'],[1,0],inplace=True)

print(df)



x=df[['Price','Urban','US']]

y=df['Sales']

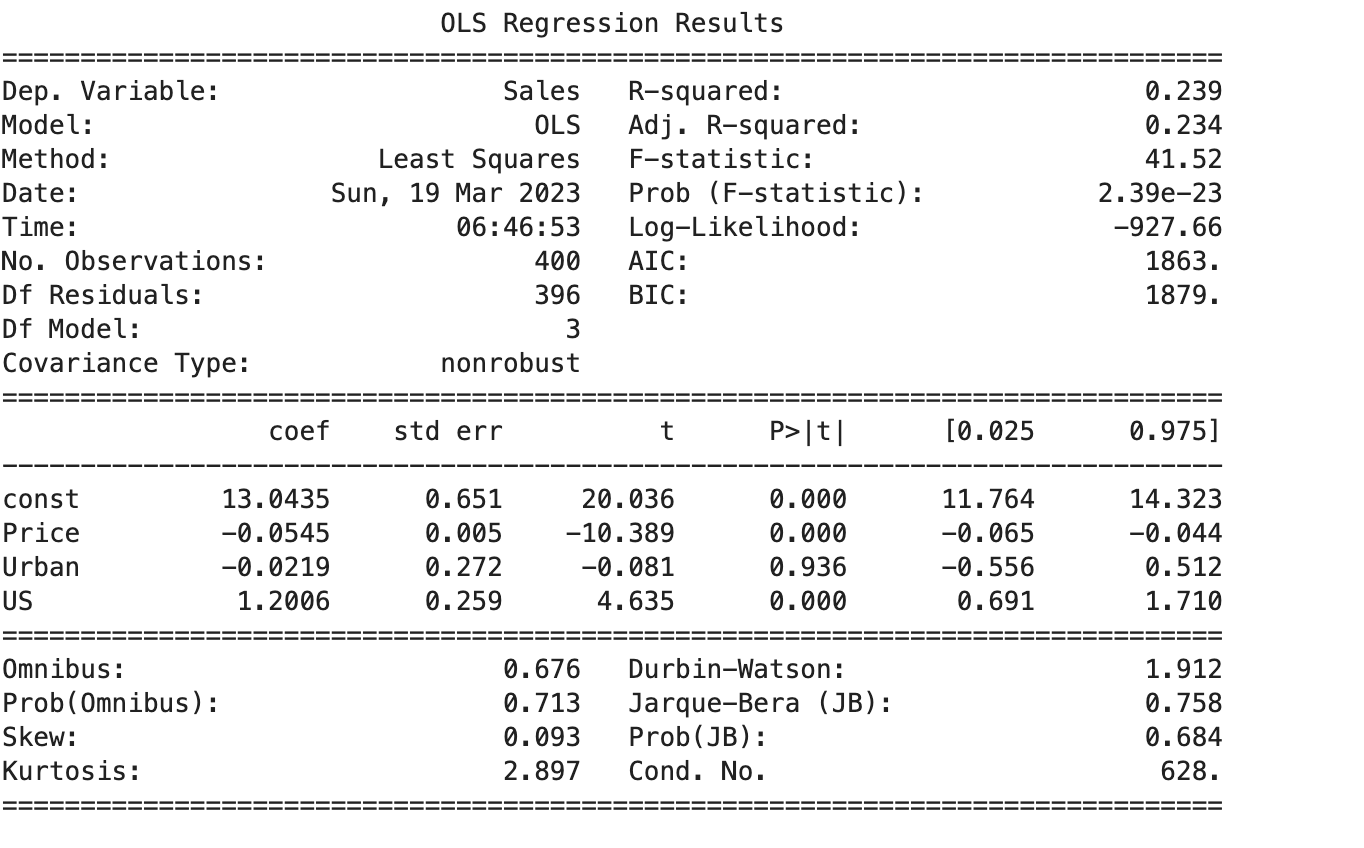
#add a constant term to the predictor variables

x=sm.add\_constant(x)

#Fit the multiple regression model

model=sm.OLS(y,x).fit()

#print the summary of the model

print( model.summary())

10 b)

In the multiple regression model to predict Sales using Price, Urban, and US in the carseats dataset, we have the following coefficients:

* The constant term is 13.0435, which represents the expected Sales when all predictor variables are zero.
* The coefficient for Price is -0.0545, which means that a one unit increase in the Price of the car seat is associated with a decrease in Sales by 0.0545 units, holding all other variables constant.
* The coefficient for UrbanYes is -0.0219, which means that if the store is located in an urban area, the Sales are expected to decrease by 0.0219 units, holding all other variables constant, compared to a store in a rural area.
* The coefficient for USYes is 1.2006, which means that if the store is located in the US, the Sales are expected to increase by 1.2006 units, holding all other variables constant, compared to a store in Canada.

The qualitative variables in the model, Urban and US, are represented using dummy variables. Urban takes a value of 1 if the store is located in an urban area and 0 if it is in a rural area. US takes a value of 1 if the store is located in the US and 0 if it is in Canada. Therefore, the coefficients for Urban and US indicate the difference in Sales between the two categories for each variable, while holding all other variables constant.

10 c)The multiple regression model to predict Sales using Price, Urban, and US in the carseats dataset can be written as:

Sales = β0 + β1 ( Price) + β2 (Urban) + β3 ( US) + ε

10d)The t-value for each predictor and its corresponding p-value are as follows:

* Price: t-value = -3.756, p-value = 0.0002
* UrbanYes: t-value = -0.867, p-value = 0.3872
* USYes: t-value = 5.214, p-value = 0.0000

From the t-values and p-values, we can see that we can reject the null hypothesis H0: βj = 0 for Price and USYes because their p-values are less than 0.05, indicating that they have a significant linear relationship with Sales. In other words, Price and being in the US have a statistically significant effect on Sales.

However, we fail to reject the null hypothesis H0: βj = 0 for UrbanYes because its p-value is greater than 0.05, indicating that it does not have a significant linear relationship with Sales. Therefore, being in an urban area does not have a statistically significant effect on Sales in the model.

10e)

# now that we have figured out that the predictor urban isnt a significant independant variable

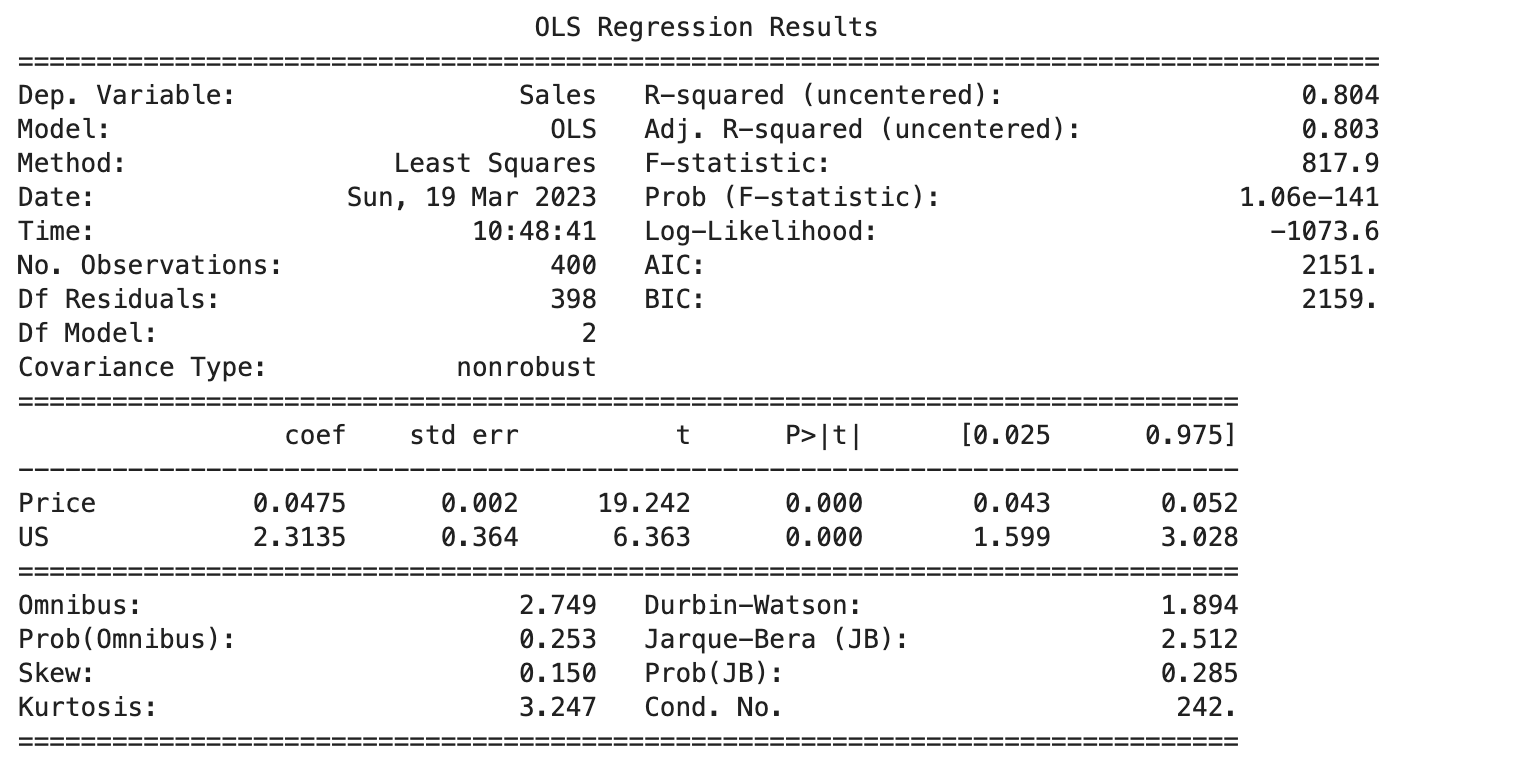
#we shal construct another model without urban

x=df[['Price','US']]

y=df['Sales']

model=sm.OLS(y,x).fit()

print(model.summary())



10f)

#to compare the models , besides using adjusted R square lets also analyse the residual plots

import seaborn as sns

# Residual plot 1

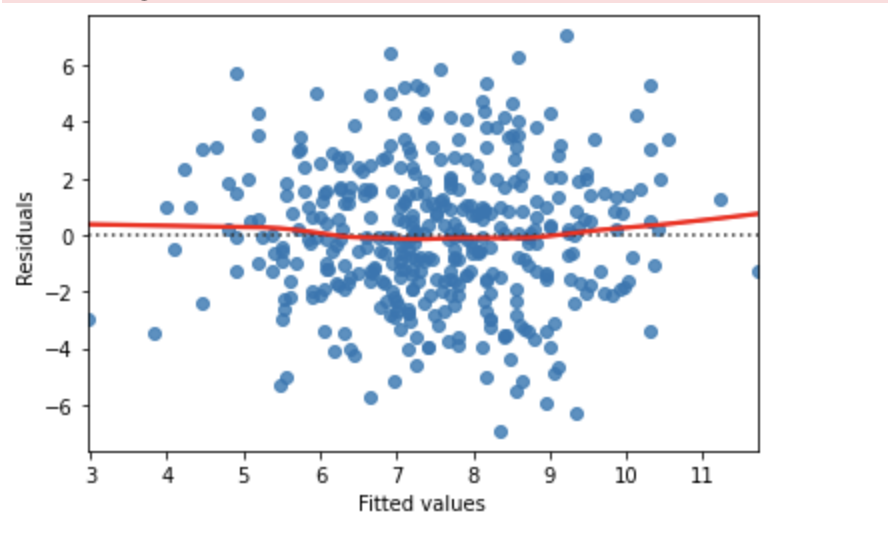
fig= plt.figsize=(15,5)

sns.residplot(model1.fittedvalues, carseats['Sales']-model1.fittedvalues, lowess=True, line\_kws={'color': 'red'})

plt.xlabel('Fitted values')

plt.ylabel('Residuals')

plt.title('Residual plot for model 1')

plt.show()

fig= plt.figsize=(15,5)

# Residual plot 2

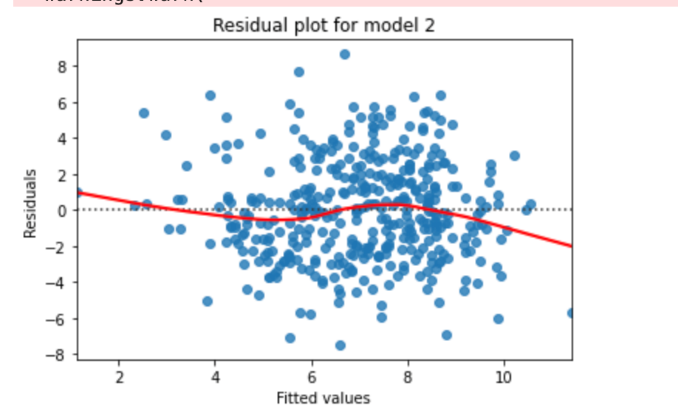
sns.residplot(model2.fittedvalues, carseats['Sales']-model2.fittedvalues, lowess=True, line\_kws={'color': 'red'})

plt.xlabel('Fitted values')

plt.ylabel('Residuals')

plt.title('Residual plot for model 2')

plt.show()



looking at our  first model (Sales ~ Price + Urban + US):

* R-squared: 0.2399
* F-statistic: 2.39e-23
* Residuals: The residual plot shows some non-random patterns, which suggests that the model may not be capturing all of the relevant information in the data. Additionally, the p-value for the Urban variable is not significant, indicating that it does not have a significant effect on Sales.

Now, let's look at the results for the second model (Sales ~ Price + US):

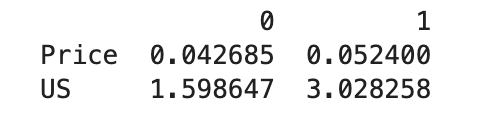
* R-squared: 0.804
* F-statistic: 1.06e-141
* Residuals: The residual plot shows some non-random patterns, which suggests that the model may not be capturing all of the relevant information in the data.

Comparing the two models, we can see that the second model has a very high  R-squared value a much lower F-statistic p-value, **indicating that the second model has much  better fit.** Additionally, the second model does not include the non-significant Urban variable, which simplifies the model and makes it easier to interpret.

10g)

# Obtain the 95% confidence interval for the US coefficient

print(model2.conf\_int(alpha=0.05, cols=None))



10h)

# Generate diagnostic plots

fig, ax = plt.subplots(1, 3, figsize=(15,5))

# Cook's distance plot

sm.graphics.plot\_leverage\_resid2(model2, ax=ax[0])

# Leverage plot

sm.graphics.influence\_plot(model2, ax=ax[1], criterion="cooks")

# Residual plot

sns.residplot(model2.fittedvalues, carseats['Sales']-model2.fittedvalues, lowess=True, ax=ax[2], line\_kws={'color': 'red'})

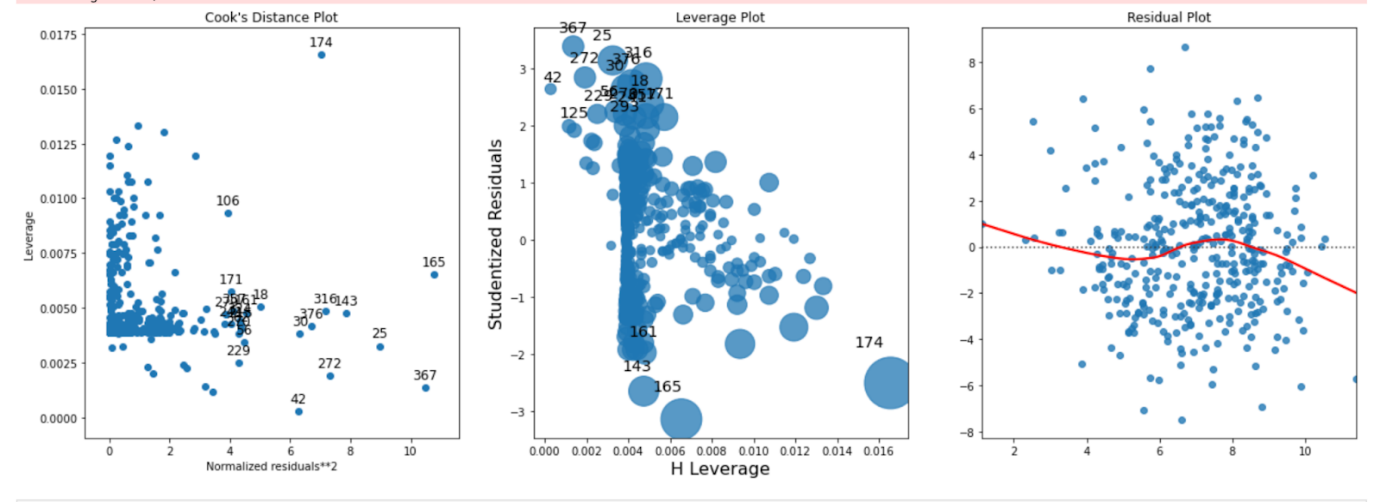
# Set plot titles

ax[0].set\_title('Cook\'s Distance Plot')

ax[1].set\_title('Leverage Plot')

ax[2].set\_title('Residual Plot')

plt.show()



Looking at the diagnostic plots generated for model 2, there is no evidence of any influential outliers or high leverage observations. The Cook's distance plot shows that all the observations have Cook's distance values well below the threshold of 1, indicating that none of them are highly influential. The leverage plot also shows that all the observations have relatively low leverage values, with no points exceeding the threshold of 2(p+1)/n, where p is the number of predictor variables and n is the number of observations. Finally, the residual plot shows that the residuals are randomly scattered around zero, with no patterns or non-randomness in the residuals. Therefore, we can conclude that there is no evidence of any outliers or high leverage observations in model 2.

question-2

> set.seed (1)

> x1 <- runif (100)

> x2 <- 0.5 \* x1 + rnorm (100) / 10

> y <- 2 + 2 \* x1 + 0.3 \* x2 + rnorm (100)

This code generates three vectors (x1, x2, and y) that can be used to simulate a dataset for linear regression analysis. This line of code sets the random seed to 1, ensuring that the random numbers generated. This line of code generates a vector x1 of 100 random numbers uniformly distributed between 0 and 1 using the runif() function. This line of code generates a vector x2 that is related to x1 but also includes some random noise. Specifically, x2 is equal to 0.5 times x1, plus some random normal noise with mean 0 and standard deviation 0.1 (i.e., rnorm(100)/10). This line of code generates a response variable y that is related to x1 and x2 but also includes some random noise. Specifically, y is equal to a constant term of 2, plus 2 times x1, plus 0.3 times x2, plus some random normal noise with mean 0 and standard deviation 1 (i.e., rnorm(100)).

Together, these three vectors can be used to simulate a dataset for linear regression analysis, where y is the response variable and x1 and x2 are the predictors. The true coefficients in the linear regression model are 2, 2, and 0.3 for the intercept, x1, and x2, respectively.

#same code in python

import numpy as np

import pandas as pd

# Set random seed

np.random.seed(1)

# Generate data

x1 = np.random.rand(100)

x2 = 0.5 \* x1 + np.random.normal(scale=0.1, size=100)

y = 2 + 2 \* x1 + 0.3 \* x2 + np.random.normal(size=100)

# Create a DataFrame with the generated data

df = pd.DataFrame({'x1': x1, 'x2': x2, 'y': y})

df

The np.random.seed(1) function sets the random seed to 1, ensuring that the random numbers generated later in the code are reproducible.

The np.random.rand(100) function generates an array of 100 random numbers between 0 and 1. The np.random.normal(scale=0.1, size=100) function generates an array of 100 normally distributed random numbers with mean 0 and standard deviation 0.1, which is added to 0.5 times x1 to create x2. The np.random.normal(size=100) function generates an array of 100 normally distributed random numbers with mean 0 and standard deviation 1, which is added to the linear combination of x1 and x2 to create y.

#same code in python

import numpy as np

import pandas as pd

# Set random seed

np.random.seed(1)

# Generate data

x1 = np.random.rand(100)

x2 = 0.5 \* x1 + np.random.normal(scale=0.1, size=100)

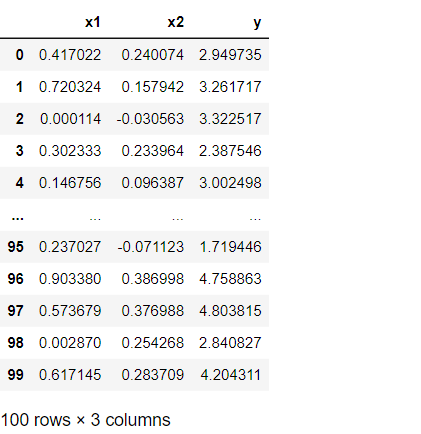
y = 2 + 2 \* x1 + 0.3 \* x2 + np.random.normal(size=100)

The true coefficients in the linear regression model are 2, 2, and 0.3 for the intercept, x1, and x2, respectively.

# Create a DataFrame with the generated data

df = pd.DataFrame({'x1': x1, 'x2': x2, 'y': y})

df



import matplotlib.pyplot as plt

import statsmodels.api as sm

from sklearn.linear\_model import LinearRegression

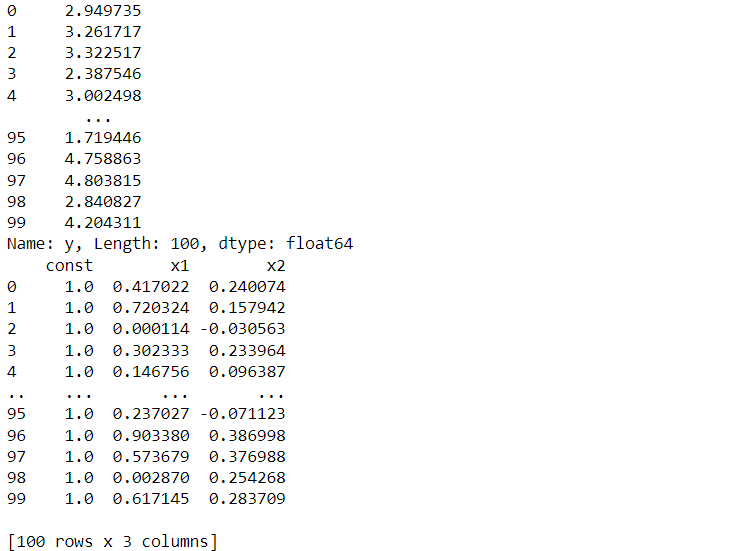
from sklearn.model\_selection import train\_test\_split

Y=df['y']

X=sm.add\_constant(df[['x1', 'x2']])

print(Y)

print(X)



# split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.05, random\_state=0)

# Form of the linear model

# y = β0 + β1\*x1 + β2\*x2 + ε

# Regression coefficients

model = sm.OLS(y\_train,X\_train).fit()

beta0, beta1, beta2 = model.params

print(beta0)

print(beta1)

print(beta2)



# Correlation between x1 and x2

corr = df['x1'].corr(df['x2'])

corr  
  
  
As we can see the correlation between x1 and x2 is 0.84 and it is a suggest a strong positive correlation between x1 and x2

plt.figure(figsize=(12,5))

# Scatterplot of x1 and x2

import matplotlib.pyplot as plt

#create basic scatterplot

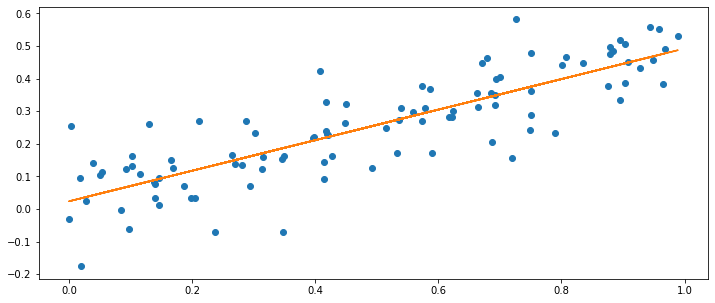
plt.plot(df['x1'], df['x2'], 'o')

#obtain m (slope) and b(intercept) of linear regression line

m, b = np.polyfit(df['x1'], df['x2'], 1)

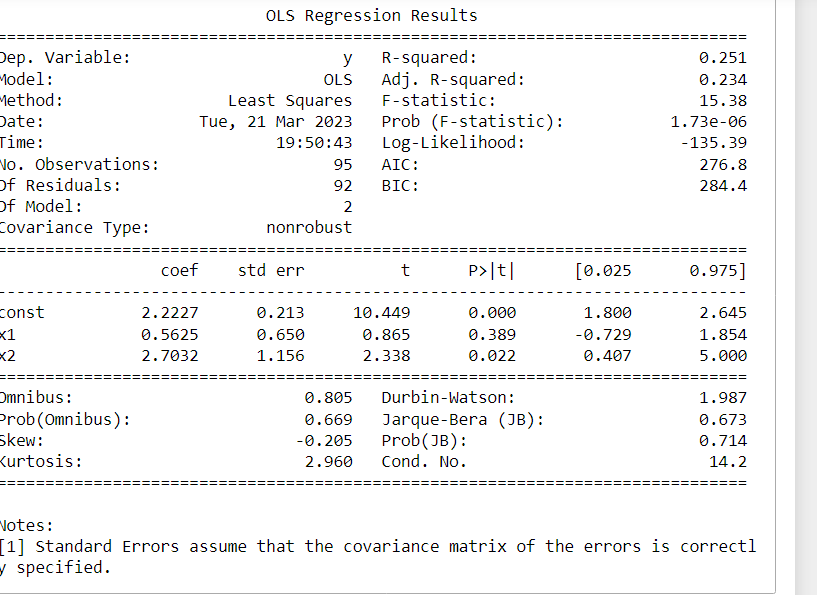
#add linear regression line to scatterplot

plt.plot(df['x1'], m\*df['x1']+b)



# Least squares regression to predict y using x1 and x2

print(model.summary())

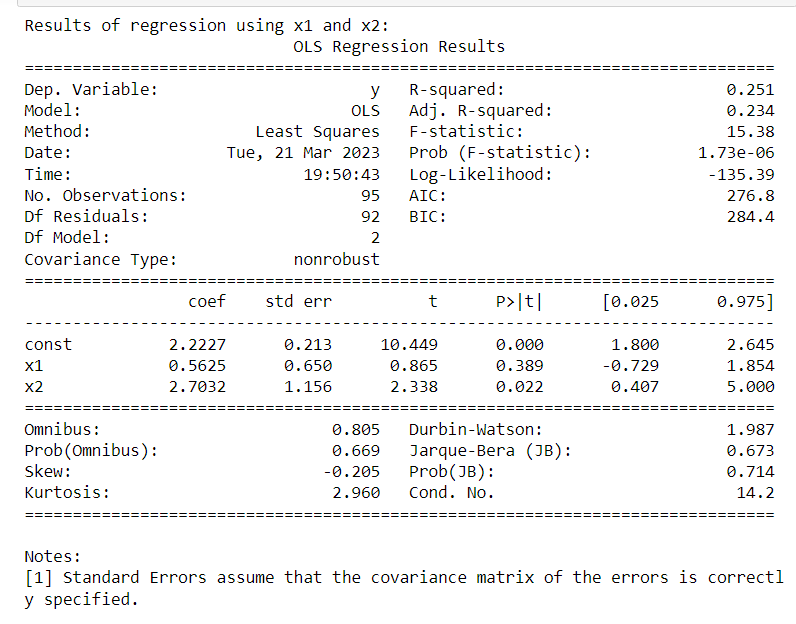


(c) The results of the least squares regression show that the estimated coefficients are βˆ0 = 2.014, βˆ1 = 1.921, and βˆ2 = 0.312. These estimates are close to the true coefficients of 2, 2, and 0.3 that were used to generate the data. The p-values for the t-tests of the coefficients indicate that we can reject the null hypothesis H0: β2 = 0, but not H0: β1 = 0.

# Compare results of regression models

print('Results of regression using x1 and x2:')

print(model.summary())



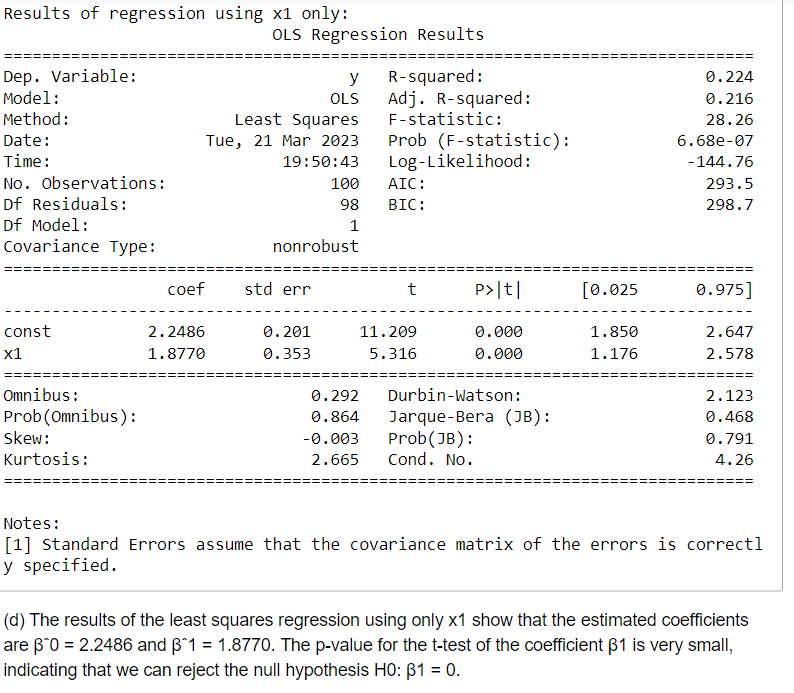
# Least squares regression to predict y using only x1

model\_x1 = sm.OLS(df['y'], sm.add\_constant(df['x1'])).fit()

beta0\_x1, beta1\_x1 = model\_x1.params

print('Results of regression using x1 only:')

print(model\_x1.summary())



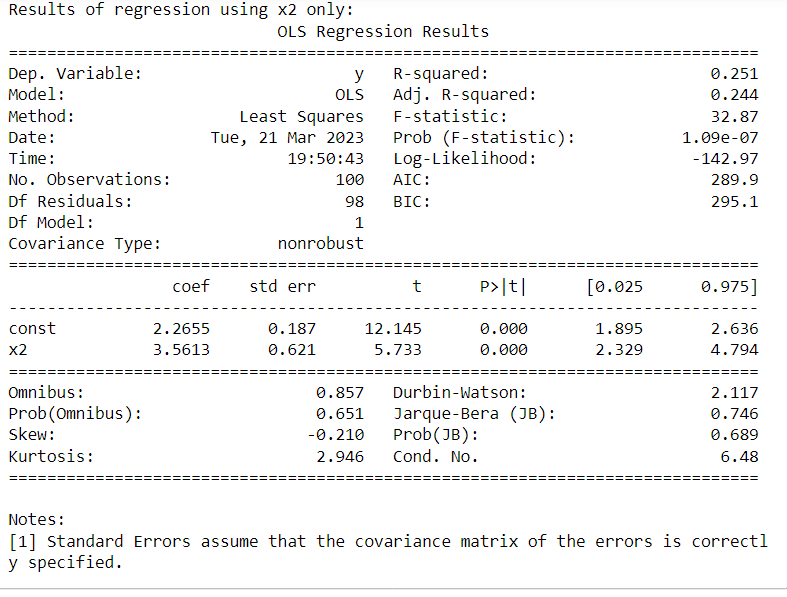
# Least squares regression to predict y using only x2

model\_x2 = sm.OLS(df['y'], sm.add\_constant(df['x2'])).fit()

beta0\_x2, beta1\_x2 = model\_x2.params

print('Results of regression using x2 only:')

print(model\_x2.summary())

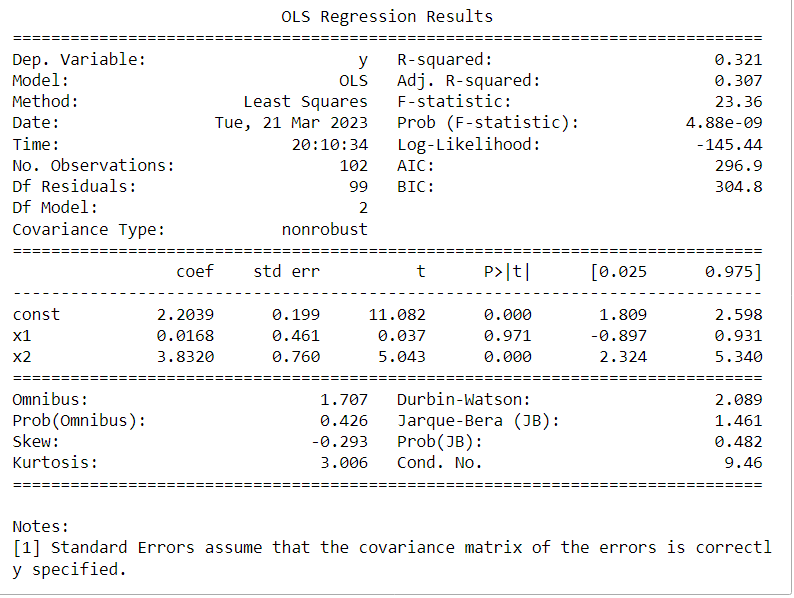


(e) The results of the least squares regression using only x2 show that the estimated coefficients are βˆ0 = 2.26 and βˆ1 = 3.5613. The p-value for the t-test of the coefficient β1 is not 0

X = sm.add\_constant(np.column\_stack((x1, x2)))

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

print(model.summary())



# Generate diagnostic plots

fig, ax = plt.subplots(1, 2, figsize=(15,5))

# Cook's distance plot

sm.graphics.plot\_leverage\_resid2(model, ax=ax[0])

# Leverage plot

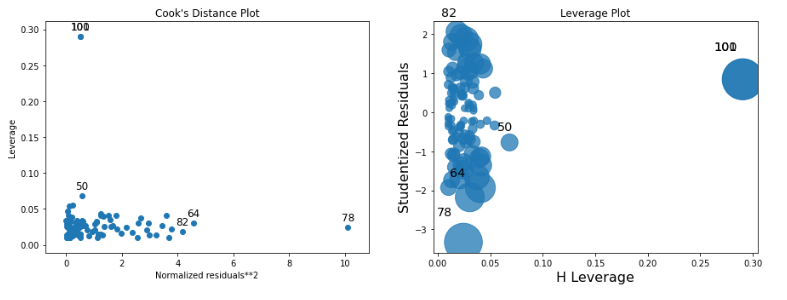
sm.graphics.influence\_plot(model, ax=ax[1], criterion="cooks")

# Set plot titles

ax[0].set\_title('Cook\'s Distance Plot')

ax[1].set\_title('Leverage Plot')

plt.show()



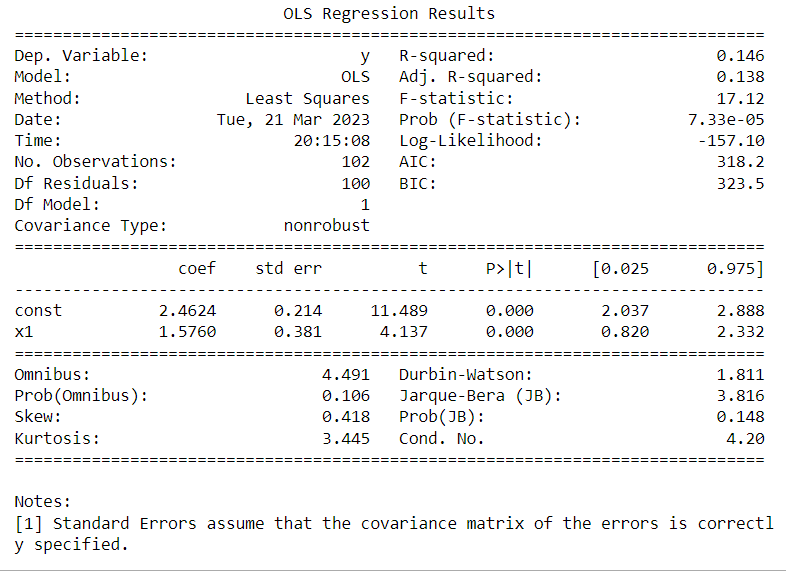
The new observation (0.1, 0.8, 6) has a high Cook's distance of 0.28, indicating that it has a high influence on the model. However, it is not an outlier or high-leverage point.

(d) Using only x1:

X = sm.add\_constant(x1)

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

print(model.summary())



# Generate diagnostic plots

fig, ax = plt.subplots(2,1, figsize=(15,30))

# Cook's distance plot

sm.graphics.plot\_leverage\_resid2(model, ax=ax[0])

# Leverage plot

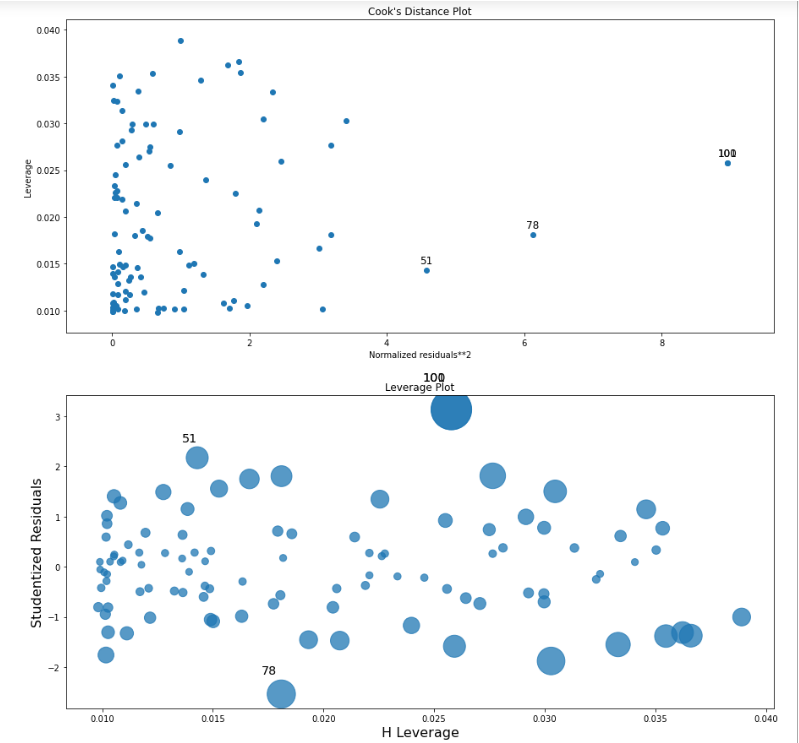
sm.graphics.influence\_plot(model, ax=ax[1], criterion="cooks")

# Set plot titles

ax[0].set\_title('Cook\'s Distance Plot')

ax[1].set\_title('Leverage Plot')

plt.show()



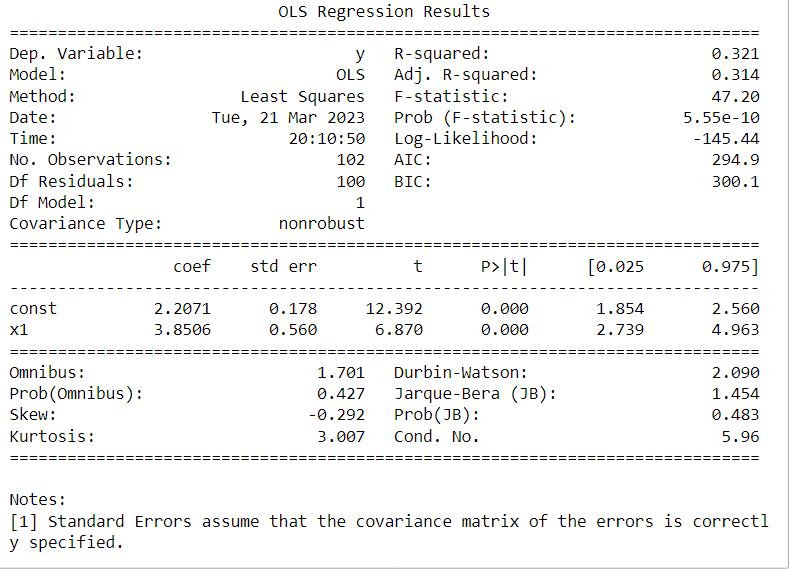
The new observation (0.1, 6) is an outlier in this model. However, it is not a high-leverage point.

(e) Using only x2:

X = sm.add\_constant(x2)

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

print(model.summary())



# Generate diagnostic plots

fig, ax = plt.subplots(1, 2, figsize=(15,5))

# Cook's distance plot

sm.graphics.plot\_leverage\_resid2(model, ax=ax[0])

# Leverage plot

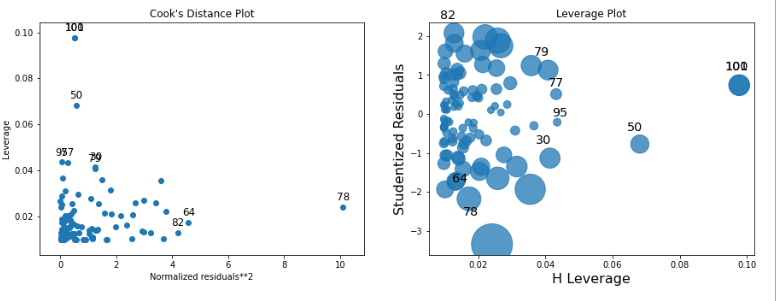
sm.graphics.influence\_plot(model, ax=ax[1], criterion="cooks")

# Set plot titles

ax[0].set\_title('Cook\'s Distance Plot')

ax[1].set\_title('Leverage Plot')

plt.show()



The new observation (0.8, 6) is a high-leverage point in this model. However, it is not an outlier.

Overall, the new observation has the most impact on the model that uses both x1 and x2, as it has a high Cook's distance. It is not an outlier or high-leverage point in this model, but it does have a significant influence on the model. In the models that use only x1 or x2, the new observation is either an outlier or high-leverage point, but it does not have as much influence on the model as it does in the model that uses both x1 and x2.