NK_multicollinearity_toy_ex

September 24, 2024

[30]:
```
import numpy as np
import pandas as pd
from statsmodels.formula.api import ols
from plotnine import ggplot, aes, geom_point, labs
from scipy.stats import linregress
```

1 Multicollinearity Demo

[31]:
```
### multicollinearity

# Uncorrelated predictor variables
X1 = [4, 4, 4, 4, 6, 6, 6, 6]
X2 = [2, 2, 3, 3, 2, 2, 3, 3]
Y = [42, 39, 48, 51, 49, 53, 61, 60]
```

[32]:
```
# Creating DataFrame
df = pd.DataFrame({
    'X1': X1,
    'X2': X2,
    'Y': Y
})

# Plot X1 vs X2
plot = (ggplot(df, aes(x='X1', y='X2')) +
        geom_point() +
        labs(x='X1', y='X2', title='Scatter Plot of X1 vs X2'))
plot
```

[33]:

```python
# Correlation between X1 and X2
correlation = np.corrcoef(X1, X2)[0, 1]
print(f'Correlation between X1 and X2: {correlation}')
```

```
Correlation between X1 and X2: 0.0
```

[34]:

```python
# Linear model Y ~ X1 + X2
model1 = ols('Y ~ X1 + X2', data=df).fit()
model1.summary()
```

```
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/scipy/stats/_axis_nan_policy.py:531: UserWarning: kurtosistest only
valid for n>=20 … continuing anyway, n=8
```

[34]:

Dep. Variable:	Y	**R-squared:**	0.958
Model:	OLS	**Adj. R-squared:**	0.941
Method:	Least Squares	**F-statistic:**	57.06
Date:	Tue, 24 Sep 2024	**Prob (F-statistic):**	0.000361
Time:	11:21:56	**Log-Likelihood:**	-14.511
No. Observations:	8	**AIC:**	35.02
Df Residuals:	5	**BIC:**	35.26
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	0.3750	4.740	0.079	0.940	-11.811	12.561
X1	5.3750	0.664	8.097	0.000	3.669	7.081
X2	9.2500	1.328	6.968	0.001	5.837	12.663

Omnibus:	2.902	**Durbin-Watson:**	2.773
Prob(Omnibus):	0.234	**Jarque-Bera (JB):**	0.878
Skew:	-0.108	**Prob(JB):**	0.645
Kurtosis:	1.391	**Cond. No.**	42.1

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

[35]:
```
# Linear model Y ~ X1
model_X1 = ols('Y ~ X1', data=df).fit()
model_X1.summary()
```

```
/opt/homebrew/anaconda3/lib/python3.12/site-
packages/scipy/stats/_axis_nan_policy.py:531: UserWarning: kurtosistest only
valid for n>=20 … continuing anyway, n=8
```

[35]:

Dep. Variable:	Y	**R-squared:**	0.550
Model:	OLS	**Adj. R-squared:**	0.476
Method:	Least Squares	**F-statistic:**	7.347
Date:	Tue, 24 Sep 2024	**Prob (F-statistic):**	0.0351
Time:	11:21:56	**Log-Likelihood:**	-23.995
No. Observations:	8	**AIC:**	51.99
Df Residuals:	6	**BIC:**	52.15
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	23.5000	10.111	2.324	0.059	-1.242	48.242
X1	5.3750	1.983	2.711	0.035	0.523	10.227

Omnibus:	3.389	**Durbin-Watson:**	1.815
Prob(Omnibus):	0.184	**Jarque-Bera (JB):**	0.935
Skew:	-0.117	**Prob(JB):**	0.626
Kurtosis:	1.341	**Cond. No.**	27.0

Notes:

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

[36]:
```
# Linear model Y ~ X2
model_X2 = ols('Y ~ X2', data=df).fit()
model_X2.summary()
```

/opt/homebrew/anaconda3/lib/python3.12/site-packages/scipy/stats/_axis_nan_policy.py:531: UserWarning: kurtosistest only valid for n>=20 … continuing anyway, n=8

[36]:

Dep. Variable:	Y	**R-squared:**	0.408
Model:	OLS	**Adj. R-squared:**	0.309
Method:	Least Squares	**F-statistic:**	4.128
Date:	Tue, 24 Sep 2024	**Prob (F-statistic):**	0.0885
Time:	11:21:56	**Log-Likelihood:**	-25.100
No. Observations:	8	**AIC:**	54.20
Df Residuals:	6	**BIC:**	54.36
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	27.2500	11.608	2.348	0.057	-1.153	55.653
X2	9.2500	4.553	2.032	0.088	-1.891	20.391

Omnibus:	4.296	**Durbin-Watson:**	0.359
Prob(Omnibus):	0.117	**Jarque-Bera (JB):**	1.012
Skew:	-0.008	**Prob(JB):**	0.603
Kurtosis:	1.257	**Cond. No.**	14.9

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

[37]:
```
# ANOVA for both models
from statsmodels.stats.anova import anova_lm

anova_results1 = anova_lm(model1)
anova_results1
```

[37]:

	df	sum_sq	mean_sq	F	PR(>F)
X1	1.0	231.125	231.125	65.567376	0.000466
X2	1.0	171.125	171.125	48.546099	0.000937
Residual	5.0	17.625	3.525	NaN	NaN

[38]:
```
anova_results_X1 = anova_lm(model_X1)
anova_results_X1
```

[38]:

	df	sum_sq	mean_sq	F	PR(>F)
X1	1.0	231.125	231.125000	7.34702	0.035081
Residual	6.0	188.750	31.458333	NaN	NaN

[39]:
```
anova_results_X2 = anova_lm(model_X2)
anova_results_X2
```

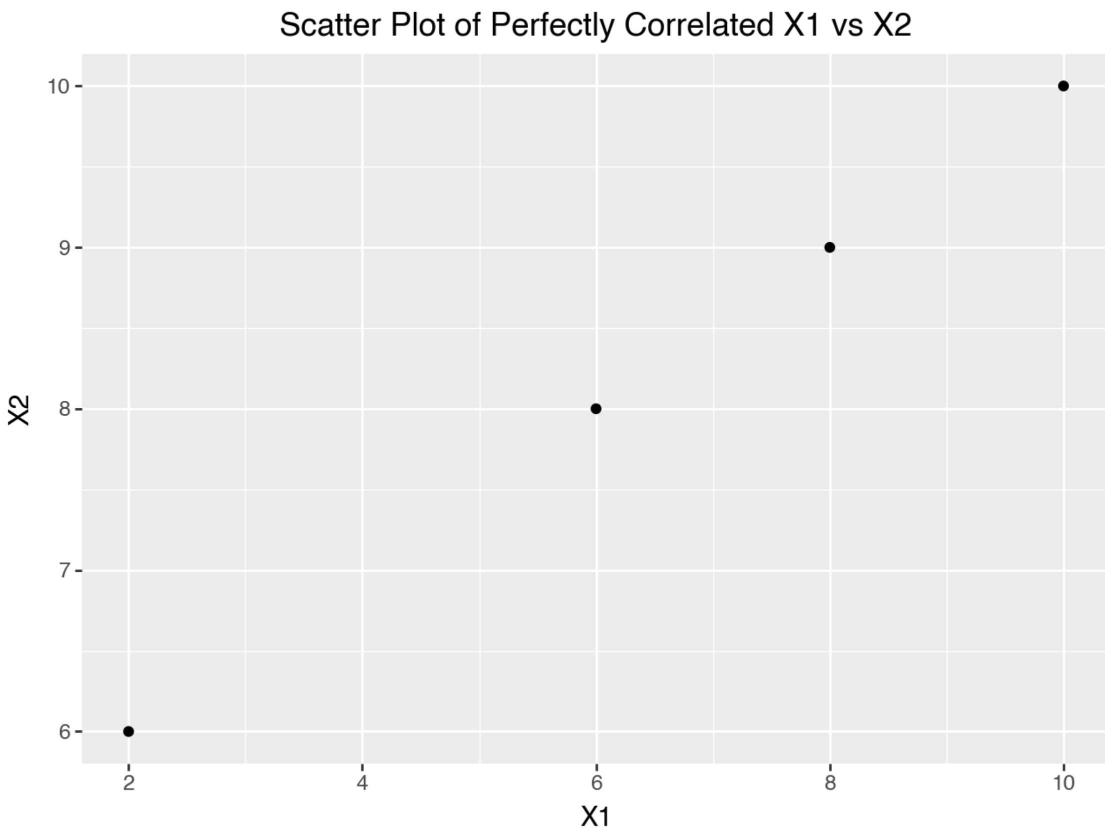
[39]:

	df	sum_sq	mean_sq	F	PR(>F)
X2	1.0	171.125	171.125000	4.127638	0.08846
Residual	6.0	248.750	41.458333	NaN	NaN

[40]:
```
# Perfectly correlated case
X1 = [2, 8, 6, 10]
X2 = [6, 9, 8, 10]
Y = [23, 83, 63, 103]

df_perfect = pd.DataFrame({
    'X1': X1,
    'X2': X2,
    'Y': Y
})

# Plot X1 vs X2
plot_perfect = (ggplot(df_perfect, aes(x='X1', y='X2')) +
                geom_point() +
                labs(x='X1', y='X2', title='Scatter Plot of Perfectly␣
 ↪Correlated X1 vs X2'))
plot_perfect
```



[41]:
```
# Correlation between X1 and X2
correlation_perfect = np.corrcoef(X1, X2)[0, 1]
print(f'Correlation between X1 and X2: {correlation_perfect}')
```

```
Correlation between X1 and X2: 1.0
```

[42]:
```
X = np.column_stack([np.ones(len(X1)), X1, X2])
X
```

[42]:
```
array([[ 1.,  2.,  6.],
       [ 1.,  8.,  9.],
       [ 1.,  6.,  8.],
       [ 1., 10., 10.]])
```

[43]:
```
# Matrix operations

XtX = np.dot(X.T, X)
print(f'XtX Matrix:\n{XtX}')
```

```
XtX Matrix:
[[  4.  26.  33.]
 [ 26. 204. 232.]
```

```
 [ 33. 232. 281.]]
```

[44]:
```
# Eigenvalues
eigenvalues = np.linalg.eigvals(XtX)
print(f'Eigenvalues:\n{eigenvalues}') #2nd eigen val very close to zero !!!!
#numerically near singular
#this is going to be an issue for us:
```

```
Eigenvalues:
[4.81365468e+02 7.42814262e-17 7.63453185e+00]
```

[45]:
```
import scipy.linalg as spla

# Attempt to invert the matrix
XtX_inv = spla.solve(XtX, np.eye(3))
print(f'Inverse of XtX:\n{XtX_inv}')

#it blows up and gives you a warning on the "Condition Number" of the matrix
```

```
Inverse of XtX:
[[-4.56937087e+15 -4.56937087e+14  9.13874175e+14]
 [-4.56937087e+14 -4.56937087e+13  9.13874175e+13]
 [ 9.13874175e+14  9.13874175e+13 -1.82774835e+14]]
```

```
/var/folders/86/c2gz31wn29b2r_53d_q3g3hc0000gn/T/ipykernel_1914/3933071815.py:4:
LinAlgWarning: Ill-conditioned matrix (rcond=3.08324e-19): result may not be
accurate.
```

[46]:
```
#the warning is basically telling you the result is nonsense. Sanity check!
```

[47]:
```
XtX_inv @ XtX # this is not the identity.... cry
```

[47]:
```
array([[ 2.75    , -2.      , 16.75    ],
       [ 0.21875,  6.75    ,  4.46875],
       [-0.25    , -2.      , -4.25    ]])
```

[48]:
```
#If you use np.linalg.inv ... it doesn't do the float calcs exactly and will
 ↪actually let you invert a singular matrix... watch out!
np.linalg.inv(XtX)

# Interesting discussion of the issue: https://github.com/numpy/numpy/issues/
 ↪2074
```

[48]:
```
array([[-4.56937087e+15, -4.56937087e+14,  9.13874175e+14],
       [-4.56937087e+14, -4.56937087e+13,  9.13874175e+13],
       [ 9.13874175e+14,  9.13874175e+13, -1.82774835e+14]])
```

1.1 Variance Inflation Factors - VIFs

[49]:
```
# Variance Inflation - generating new data
np.random.seed(0)
X1 = np.random.uniform(25, 50, 100)
X2 = np.random.normal(10, 3, 100)
eps = np.random.normal(0, 4, 100)

df_vif = pd.DataFrame({
    'X1': X1,
    'X2': X2,
    'Y': 7 + 3 * X1 + 5 * X2 + eps
})

# Correlation between X1 and X2
correlation_X1_X2 = np.corrcoef(X1, X2)[0, 1]
print(f'Correlation between X1 and X2: {correlation_X1_X2}')
```

```
Correlation between X1 and X2: -0.01824782721587361
```

[50]:
```
# Linear model with X1 and X2
mod1 = ols('Y ~ X1 + X2', data=df_vif).fit()
mod1.summary()
```

[50]:

Dep. Variable:	Y	**R-squared:**	0.978
Model:	OLS	**Adj. R-squared:**	0.978
Method:	Least Squares	**F-statistic:**	2193.
Date:	Tue, 24 Sep 2024	**Prob (F-statistic):**	1.82e-81
Time:	11:21:56	**Log-Likelihood:**	-277.67
No. Observations:	100	**AIC:**	561.3
Df Residuals:	97	**BIC:**	569.2
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	5.9068	2.505	2.358	0.020	0.934	10.880
X1	2.9676	0.055	54.178	0.000	2.859	3.076
X2	5.1591	0.132	39.062	0.000	4.897	5.421

Omnibus:	0.196	**Durbin-Watson:**	2.109
Prob(Omnibus):	0.907	**Jarque-Bera (JB):**	0.026
Skew:	0.032	**Prob(JB):**	0.987
Kurtosis:	3.046	**Cond. No.**	247.

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

[51]:
```
vif_model_X1o = ols('X1 ~ X2', data=df_vif).fit()
r2j_X1o = vif_model_X1o.rsquared
vif_j_x1 = 1/(1-r2j_X1o)
```

```
print(f'VIF for X1: {vif_j_x1}')
```

```
VIF for X1: 1.000333094112843
```

1.1.1 VIF Close to 1 indicates not very much multicollinearity between X1 and other predictors.

1.1.2 On the other hand...

[52]:
```
# Creating a nearly perfectly collinear variable X3
df_vif['X3'] = 2 * df_vif['X1'] + 3 * df_vif['X2'] + np.random.normal(0, 0.01,␣
↪100)

correlation_X1_X3 = np.corrcoef(df_vif['X1'], df_vif['X3'])[0, 1]
correlation_X2_X3 = np.corrcoef(df_vif['X2'], df_vif['X3'])[0, 1]
correlation_comb = np.corrcoef(2 * df_vif['X1'] + 3 * df_vif['X2'],␣
↪df_vif['X3'])[0, 1]

print(f'Correlation between X1 and X3: {correlation_X1_X3}')
print(f'Correlation between X2 and X3: {correlation_X2_X3}')
print(f'Correlation between (2*X1 + 3*X2) and X3: {correlation_comb}')
```

```
Correlation between X1 and X3: 0.8464409676819135
Correlation between X2 and X3: 0.5169479332790725
Correlation between (2*X1 + 3*X2) and X3: 0.9999998570072173
```

[53]:
```
# Linear model with X1, X2, and X3
mod2 = ols('Y ~ X1 + X2 + X3', data=df_vif).fit()
mod2.summary()
```

[53]:

Dep. Variable:	Y	**R-squared:**	0.979
Model:	OLS	**Adj. R-squared:**	0.978
Method:	Least Squares	**F-statistic:**	1469.
Date:	Tue, 24 Sep 2024	**Prob (F-statistic):**	4.67e-80
Time:	11:21:56	**Log-Likelihood:**	-276.92
No. Observations:	100	**AIC:**	561.8
Df Residuals:	96	**BIC:**	572.3
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	5.8079	2.501	2.322	0.022	0.844	10.772
X1	-102.9961	87.588	-1.176	0.243	-276.858	70.866
X2	-153.7766	131.375	-1.171	0.245	-414.553	107.000
X3	52.9822	43.795	1.210	0.229	-33.949	139.914

Omnibus:	0.280	**Durbin-Watson:**	2.107
Prob(Omnibus):	0.870	**Jarque-Bera (JB):**	0.044
Skew:	0.020	**Prob(JB):**	0.978
Kurtosis:	3.094	**Cond. No.**	4.73e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.73e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

[54]:
```
# VIF calculation for X1
vif_model_X1 = ols('X1 ~ X2 + X3', data=df_vif).fit()
r2j_X1 = vif_model_X1.rsquared
vif_j_X1 = 1 / (1 - r2j_X1)
print(f'VIF for X1: {vif_j_X1}')
```

```
VIF for X1: 2570128.137094504
```

[55]:
```
# VIF calculation for X3
vif_model_X2 = ols('X2 ~ X1 + X3', data=df_vif).fit()
r2j_X2 = vif_model_X2.rsquared
vif_j_X2 = 1 / (1 - r2j_X2)
print(f'VIF for X2: {vif_j_X2}')
```

```
VIF for X2: 994491.2150962795
```

[56]:
```
# VIF calculation for X3
vif_model_X3 = ols('X3 ~ X1 + X2', data=df_vif).fit()
r2j_X3 = vif_model_X3.rsquared
vif_j_X3 = 1 / (1 - r2j_X3)
print(f'VIF for X3: {vif_j_X3}')
```

```
VIF for X3: 3506271.326686703
```

1.1.3 VIFs»>10 indicate severe multicollinearity among predictors.

[ ]:
```

```

1.2 Exercise

For the dataset "multicollinearity.txt", perform the following:

1. Check the pairwise correlations of the predictors.
2. Compare the coefficient for each predictor in two cases: a. when regressing Y on that predictor only, and b. when regressing Y on all predictors. Do you see a major change in the coefficients? Their standard errors? What does this tell you about the presence of multicollinearity?
3. Calculate the VIFs for each predictor. Which VIFs are concerning? Comment on what you find.
4. Consider regressing `X4` onto the other predictors `X1` through `X3`. Save the residuals from this model and call them `res4`. Then regress Y onto `res4` and look at the coefficient. Compare this coefficient to the coefficients for `X4` you found in part 2. What do you notice?
5. Explain geometrically what we are doing in #4. Are we surprised by the results?

[57]:
```
data_mc = pd.read_csv('../data/multicollinearity.txt', delimiter='\t')
data_mc.head(10)
```

[57]:
```
        y     x1     x2     x3     x4
0   1.860  0.374  0.846  1.951  1.325
1   9.747  2.184  3.754  5.099  5.372
2  12.707  2.164  3.635  2.505  2.582
3  12.822  5.595  4.805  3.653  3.399
4   5.160  5.330  2.611  1.873  1.474
5  16.463  5.180  7.259  5.632  3.105
6  15.956  7.487  7.506  5.838  2.890
7  23.978  8.738  8.506  6.709  4.773
8  28.819  9.576  8.705  8.606  8.326
9  17.527  9.695  9.335  6.971  5.401
```

[62]:
```python
# Exercise 1

predictor_corr = data_mc[['x1', 'x2', 'x3', 'x4']].corr()
print('Predictor Correlations:')
print(predictor_corr)
```

```
Predictor Correlations:
          x1        x2        x3        x4
x1  1.000000  0.999811  0.999495  0.998843
x2  0.999811  1.000000  0.999671  0.999013
x3  0.999495  0.999671  1.000000  0.999306
x4  0.998843  0.999013  0.999306  1.000000
```

1. Check the pairwise correlations of the predictors.

The correlations are all very high and close to 1. This indicates high multicollinearity.

[65]:
```python
# Exercise 2

def regression_model(predictors, data):
    formula = 'y ~ ' + ' + '.join(predictors)
    model = ols(formula, data=data).fit()
    coefs = model.params
    std_errs = model.bse
    return coefs, std_errs

coefs_x1, std_errs_x1 = regression_model(['x1'], data_mc)
coefs_x2, std_errs_x2 = regression_model(['x2'], data_mc)
coefs_x3, std_errs_x3 = regression_model(['x3'], data_mc)
coefs_x4, std_errs_x4 = regression_model(['x4'], data_mc)

coefs_all, std_errs_all = regression_model(['x1', 'x2', 'x3', 'x4'], data_mc)

print(f"Coefficient for x1 (individual): {coefs_x1['x1']:.4f}, Standard Error:␣
↪{std_errs_x1['x1']:.4f}")
print(f"Coefficient for x2 (individual): {coefs_x2['x2']:.4f}, Standard Error:␣
↪{std_errs_x2['x2']:.4f}")
```

```python
print(f"Coefficient for x3 (individual): {coefs_x3['x3']:.4f}, Standard Error:␣
↪{std_errs_x3['x3']:.4f}")
print(f"Coefficient for x4 (individual): {coefs_x4['x4']:.4f}, Standard Error:␣
↪{std_errs_x4['x4']:.4f}")

print(f"Coefficient for x1 (multiple): {coefs_all['x1']:.4f}, Standard Error:␣
↪{std_errs_all['x1']:.4f}")
print(f"Coefficient for x2 (multiple): {coefs_all['x2']:.4f}, Standard Error:␣
↪{std_errs_all['x2']:.4f}")
print(f"Coefficient for x3 (multiple): {coefs_all['x3']:.4f}, Standard Error:␣
↪{std_errs_all['x3']:.4f}")
print(f"Coefficient for x4 (multiple): {coefs_all['x4']:.4f}, Standard Error:␣
↪{std_errs_all['x4']:.4f}")
```

```
Coefficient for x1 (individual): 2.1250, Standard Error: 0.0057
Coefficient for x2 (individual): 2.3613, Standard Error: 0.0053
Coefficient for x3 (individual): 2.9492, Standard Error: 0.0063
Coefficient for x4 (individual): 4.2002, Standard Error: 0.0116
Coefficient for x1 (multiple): -0.1675, Standard Error: 0.2041
Coefficient for x2 (multiple): 1.1364, Standard Error: 0.2806
Coefficient for x3 (multiple): 0.9788, Standard Error: 0.2556
Coefficient for x4 (multiple): 1.1175, Standard Error: 0.2104
```

2. Do you see a major change in the coefficients? Their standard errors? What does this tell you about the presence of multicollinearity?

Individual regressions have much largers coefficients and smaller SE's than the multiple. This suggests high multicollinarity causing instabiltiy and unreliablity when all predictors are included together.

[67]:
```python
# Excercise 3
def calculate_vif(target_predictor, data):
    predictors = ['x1', 'x2', 'x3', 'x4']
    other_predictors = [p for p in predictors if p != target_predictor]
    formula = f"{target_predictor} ~ " + ' + '.join(other_predictors)
    model = ols(formula, data=data).fit()
    r_squared = model.rsquared
    vif = 1 / (1 - r_squared)
    return vif

vif_x1 = calculate_vif('x1', data_mc)
vif_x2 = calculate_vif('x2', data_mc)
vif_x3 = calculate_vif('x3', data_mc)
vif_x4 = calculate_vif('x4', data_mc)

print(f"VIF for x1: {vif_x1:.4f}")
print(f"VIF for x2: {vif_x2:.4f}")
print(f"VIF for x3: {vif_x3:.4f}")
```

```
print(f"VIF for x4: {vif_x4:.4f}")
```

```
VIF for x1: 2653.8686
VIF for x2: 4066.1980
VIF for x3: 2162.4584
VIF for x4: 722.2889
```

3. Calculate the VIFs for each predictor. Which VIFs are concerning? Comment on what you find.

The VIF values are very high across the board. Indicating high multicollinearity for all predictors.

[70]:
```
# Excercise 4
def regression_residuals(dependent_var, independent_vars, data):
    formula = f"{dependent_var} ~ " + ' + '.join(independent_vars)
    model = ols(formula, data=data).fit()
    residuals = model.resid
    return residuals

res4 = regression_residuals('x4', ['x1', 'x2', 'x3'], data_mc)

model_res4 = ols('y ~ res4', data=data_mc.assign(res4=res4)).fit()

coef_res4 = model_res4.params['res4']
std_err_res4 = model_res4.bse['res4']

print(f"Coefficient for res4: {coef_res4:.4f}, Standard Error: {std_err_res4:.
 ↪4f}")

# From Excercise 2
print(f"Coefficient for x4 (individual): {coefs_x4['x4']:.4f}, Standard Error:␣
 ↪{std_errs_x4['x4']:.4f}")
print(f"Coefficient for x4 (multiple): {coefs_all['x4']:.4f}, Standard Error:␣
 ↪{std_errs_all['x4']:.4f}")
```

```
Coefficient for res4: 1.1175, Standard Error: 8.0280
Coefficient for x4 (individual): 4.2002, Standard Error: 0.0116
Coefficient for x4 (multiple): 1.1175, Standard Error: 0.2104
```

4. Compare this coefficient to the coefficients for `X4` you found in part 2. What do you notice?

When we regress y on the residuals of x4 (after removing the effects of x1, x2, and x3), the coefficient is 1.1175, which matches the coefficient of x4 in the multiple regression from part 2. This shows that the unique contribution of x4 to y, after accounting for its correlation with the other predictors, is consistent in both analyses.

5. Explain geometrically what we are doing in #4. Are we surprised by the results?

Geometrically, in step 4, we are projecting x4 onto the space spanned by x1, x2, and x3, which represent the component of x4 orthogonal to that space. By regressing y on these residuals, we isolate the effect of x4 that is independent of x1, x2, and x3. The results aren't surprising since we effectively remove the multicollinearity by doing this.