2024 ■ Vol. 20 ■ no. 2

From Scratch: A Didactic Example of Multivariate Regression in Python

Godwin S. Ashiabi [a]

[a]InMoment Inc.

Abstract ■ Although Python has the statsmodels library that can be used to perform different statistical analyses, including multiple regression, it has not yet implemented multivariate regression (*MVR*) as one of its methods. An investigator interested in conducting a multivariate regression is thus limited to running a series of univariate regression models, which do not take into account the collinearity among variables in the models. Thus, the purpose of this tutorial is to demonstrate how to perform multivariate regression in Python using custom user-defined classes, and linear hypothesis testing using statsmodels.

Keywords ■ linear hypothesis testing, multiple regression, multivariate regression. **Tools** ■ Python, statsmodels.

ashiabig@gmail.com

10.20982/tqmp.20.2.p186

Acting Editor ■ Damiem Rolon-Mérette (Université d'Ottawa)

Reviewers ■ Two anonymous reviewers

Introduction

Although Python (Van Rossum & Drake Jr, 2009) has the statsmodels (Seabold & Perktold, 2010) library that can be used to perform different statistical analyses, including multiple regression, it has not yet implemented multivariate regression (MVR) as one of its methods. An investigator interested in conducting a multivariate regression is thus limited to running a series of univariate regression models, which do not take into account the collinearity among variables in the models. When there is more than one response variable, the response variables may be correlated, thus, it is desirable to incorporate this correlation structure into the regression analysis (Wu & Qui, 2021). Thus, the purpose of this tutorial is to demonstrate how to perform multivariate regression in Python using custom user-defined classes, and linear hypothesis testing using statsmodels.

To start out discussions, we briefly explain the similarities and differences between multiple regression (MR) and MVR because of the confusion between these similar approaches. Multivariate regression refers to regression models that use two or more responses or dependent variables (Hidalgo & Goodman, 2013), irrespective of the number of predictor variables. When multiple predictors are used with a single dependent or outcome variable in a regression analysis, we call that statistical model a multiple regression. We will use matrix notation in our discussion of the similarities and differences because that will make it much easier for the exposition. The notations for multiple and multivariate regression are:

$$\begin{aligned} \text{MR:} Y_{\text{n} \times 1} &= X_{\text{n} \times (\text{p}+1)} B_{(\text{p}+1) \times 1} + E_{\text{n} \times 1} \\ \text{MVR: } Y_{\text{n} \times \text{m}} &= X_{\text{n} \times (\text{p}+1)} B_{(\text{p}+1) \times \text{m}} + E_{\text{n} \times \text{m}} \end{aligned} \tag{1}$$

where

- Y for MR is a column vector of n rows of observations and a single outcome, nx1. However, for MVR, Y is an nxm column vector representing n rows of cases across m columns of responses.
- X represents the model or design matrix. For both MR and MVR, the design matrix has n rows of cases by p column(s) of predictor(s) and a column vector of ones for the regression constant or intercept.
- For MR, B is a matrix of coefficient(s) with p column(s), one for each predictor, plus a column for the intercept coefficient. For MVR, just like MR, there are p coefficient(s), one for each predictor, and a column vector for the intercept term. However, because MVR is multi-response in nature, it has p x m coefficients; that is, it has p coefficient(s) for each m response variable.
- E: There is only a single error term in MR, because there is only one outcome variable. However, because MVR has m outcomes, there are as many random error terms

Listing 1 ■ Reading the data

```
# import the modules
import pandas as pd
import numpy as np

class Figure:
    value = 0
    def __new__(cls, caption):
        cls.value += 1
        print(f"Figure {cls.value}\n\n{caption.title()}")

# read the data
dfile_path = "D:/--data/covid/covid2020-df5.csv"
df = pd.read_csv(dfile_path)
Figure("View of all Columns and First Three Rows of the Data")
print(df.head(3).to_markdown(tablefmt="github", floatfmt=(".2f"),
       stralign="center", numalign="center", index=False))

##Figure 1
##View of all  Columns and First Three Rows of the Data
## |  p_health   |  m_health   |  agegrp   |  gender   |  educ   |  ss_concerns   |  p_concerns   |
## |-------------|-------------|-----------|-----------|---------|----------------|---------------|
## |    2.00     |    3.00     |   2.00    |   0.00    |  0.00   |      6.00      |     6.00      |
## |    1.00     |    2.00     |   2.00    |   1.00    |  1.00   |      6.00      |     9.00      |
## |    2.00     |    2.00     |   3.00    |   0.00    |  0.00   |      4.00      |     5.00      |
```

as there are outcomes. Simply put, each outome, m, has a random error term.

Both analytic approaches (MR and MVR) can have one or more predictors, in which case $p \geq 1$. In the case of MR, when $p = 1$, it is a simple linear regression, whereas for MVR when $p = 1$ we have a simple multivariate linear regression. However, for the purposes of this tutorial, we are going to focus on the case where $p > 1$.

In short, what equation (1) shows is that MR involves the use of one and only one outcome variable across n observations, whereas MVR involves m response variables across n observations. Specifically, MVR is a multivariate model because it has $m > 1$ response variables. As well, because MR has only one response variable, it has only one set of p coefficients plus the intercept. On the other hand, MVR has m response variables, with p predictors, making a p x m matrix of coefficients plus the intercept. The vector of random errors for MVR, $\varepsilon_{i=1}^{m}$, is assumed to follow a multivariate normal distribution ε $N_m(0, \Sigma)$, and the covariance matrix, Σ, measures variances and correlations between the m response variables.

Method

Illustrative Example: Data and Measures

Data on a 20,253 households with children aged 6-to-14 years from the *Crowdsourcing: Impacts of COVID-19 on Canadians' Parenting During the Pandemic Public Use Microdata File* (Statistics Canada, 2020) were used. Refer to Appendix A for more information on the data preprocessing steps.

Response Variables Parents rated (1 = *not at all*) to (4 = *extremely*) their concerns about the general physical and mental health of child(ren).

Parental socio-demographic factors Gender was coded (male: 1, female: 0). Attended University (yes: 1, no: 0). Age, categorized as (1: 15 to 34 years old, 2: 35 to 44 years old, 3: 45 to 54 years old, and 4: 55 and older) was treated like a Likert-type ordinal variable.

Social support and parenting concerns Each measure was a summation of three questions rated (1 = *not at all*) to (4 = *extremely*) that asked, inter alia, about parental concerns staying connected with family/friends, and managing child's behavior and emotions.

Reading the Data

In the first Listing, we import Pandas (McKinney, 2010; The Pandas Development Team, 2023) and Numpy (Harris et al., 2020) Python libraries for reading data, data manipulation and computation. All analyses and writing were completed in RStudio using R Markdown (Posit Team, 2023). The reticulate library (Ushey et al., 2023) in R was used as an interface to Python because it enables calling Python from R Markdown, and the importation of

Listing 2 ■ Checking multivariate normality and homogeneity of variance

```
from pingouin import multivariate_normality
from pingouin import box_m

# perform the Henze-Zirkler Multivariate Normality Test
Figure("Test of multivariate normality")
print(multivariate_normality(df, alpha=.05))

# box m test for homogeneity of variances
# iterate and collect results in a dictionary
boxm = {}
for i in ["agegrp", "gender", "educ"]:
  boxm[i] = box_m(df, dvs=['p_health', 'm_health'], group=i)

# concatenate results into a dataframe and print
out = pd.concat([v for k,v in boxm.items()], axis=0)
out.insert(loc=0, column="vars", value=["agegrp", "gender", "educ"])
Figure("Test of Homogeneity")
print(out.to_markdown(tablefmt="github", floatfmt=(".2f"),
       stralign="center", numalign="center"))

##Figure 2
##Test of Multivariate Normality
## HZResults(hz=826.8271426851385, pval=0.0, normal=False)

##Figure 3
##Test of Homogeneity
## |     |  vars  |  Chi2  |  df  |  pval  |  equal_cov  |
## |-----|--------|--------|------|--------|-------------|
## | box | agegrp | 13.06  | 9.00 |  0.16  |    True     |
## | box | gender |  2.18  | 3.00 |  0.54  |    True     |
## | box |  educ  | 66.38  | 3.00 |  0.00  |    False    |
```

Python modules.

Check multivariate normality and homogeneity of variance

We can check multivariate normality and homogeneity of variance assumptions using the pingouin library (Vallat, 2018), as seen in Listing 2.

The outputs in Figures 2 and 3 show that the data is not multivariate normal and that some of the variables violate the assumption of homogeneity of variance. Under heteroscedasticity, the OLS estimator gives unbiased and consistent coefficient estimates, but the estimator is biased for standard errors. A remedy is to adjust for heterogeneity of variances by using robust standard error esimators to obtain unbiased standard errors (Mansournia et al., 2021). Under homogeneity of variance, the OLS method for estimating standard errors can be expressed as: $(X'X)^{-1}(X'\Omega X)(X'X)^{-1}$ where Ω is an identity matrix of the residual standard error, $\sigma^2 I_n$. However, with heterogeneity of variance, different variants of 'robust standard errors' estimators can be constructed and substituted for the estimate of Ω. For example, the HC3 esimator (MacKinnon & White, 1985) given as $HC3 = \hat{\mu_i}^2/(1-h_{ii})^2$; where: n (number of observations), h (hat values from the hat matrix), and $\hat{\mu_i}^2$ (the squared residuals) can be used for small and large samples.

Results

Importing the Multivariate Regression Modules

We have three Python classes: MVImputer, BaseMLM, and MLM. The classes were written by the author with accessibility in mind. To import these modules as shown below, the module files have to be placed in the working directory of one's project.

```
# import classes for multivariate regression
from mvimputer import MVImputer
from mlm import BaseMLM
from mlm import MLM
```

Listing 3 ■ Running the MVR analyses (first part)

```
# run mlm class, indicating the data, the number of responses
# the starting index position of predictors and error type
mvr = MLM(data=df, endog=2, exog=2, errortype="HC3")  # HC3 robust std error
etable, ftable, loglik = mvr.model_summary            # unpack the tuple output

# print goodness of fit stats
Figure("Goodness of Fit")
print(ftable.to_markdown(tablefmt="github", floatfmt=(".2f"),
      stralign="center", numalign="center"))

# print the regression estimates
for key, vals in etable.items():
  Figure(f"{key} regression output")
  print(etable[key].to_markdown(tablefmt="github", floatfmt=(".2f"),
  stralign="center", numalign="center"))

##Figure 4
##Goodness of Fit
## |          |  K   |    DFE    |  RSE  |  DF1  |    DF2    |    F     |  P-Value   |  RSQ  |  Adj. RSQ  |
## |----------|------|-----------|-------|-------|-----------|----------|------------|-------|------------|
## | p_health | 6.00 | 20247.00 | 0.81  | 5.00  | 20247.00 | 492.89   |    0.00    | 0.11  |    0.11    |
## | m_health | 6.00 | 20247.00 | 0.73  | 5.00  | 20247.00 | 1882.24 |    0.00    | 0.32  |    0.32    |

##Figure 5
##P_Health Regression Output
## |             |  Est.  |  Robust SE   |    t    |  Pr(>|t|)  |  0.025  |  0.975  |  Std Est.  |
## |-------------|--------|--------------|---------|------------|---------|---------|------------|
## |  intercept  |  0.90  |     0.03     | 26.90  |    0.00    |  0.84   |  0.97   |    0.00    |
## |   agegrp    |  0.11  |     0.01     | 10.45  |    0.00    |  0.09   |  0.13   |    0.07    |
## |   gender    |  0.01  |     0.02     |  0.70  |    0.48    |  -0.02  |  0.05   |    0.00    |
## |    educ     | -0.14  |     0.01     | -10.57 |    0.00    |  -0.17  |  -0.12  |   -0.07    |
## | ss_concerns |  0.10  |     0.00     | 28.22  |    0.00    |  0.10   |  0.11   |    0.25    |
## | p_concerns  |  0.04  |     0.00     | 10.87  |    0.00    |  0.03   |  0.04   |    0.09    |

##Figure 6
##M_Health Regression Output
## |             |  Est.  |  Robust SE   |    t    |  Pr(>|t|)  |  0.025  |  0.975  |  Std Est.  |
## |-------------|--------|--------------|---------|------------|---------|---------|------------|
## |  intercept  |  0.63  |     0.03     | 21.86  |    0.00    |  0.58   |  0.69   |    0.00    |
## |   agegrp    |  0.07  |     0.01     |  7.92  |    0.00    |  0.05   |  0.09   |    0.05    |
## |   gender    | -0.04  |     0.02     | -2.53  |    0.01    |  -0.08  |  -0.01  |   -0.01    |
## |    educ     | -0.17  |     0.01     | -13.83 |    0.00    |  -0.19  |  -0.14  |   -0.08    |
## | ss_concerns |  0.12  |     0.00     | 37.25  |    0.00    |  0.11   |  0.13   |    0.28    |
## | p_concerns  |  0.14  |     0.00     | 46.07  |    0.00    |  0.13   |  0.14   |    0.35    |
```

The MVImputer MVImputer, a wrapper around scikit-learn's IterativeImputer (Pedregosa et al., 2011) for missing values imputation, has the fit and transform methods. The fit method runs the imputation algorithm, and the transform method is used to fill in the missing values computed by the fit method. The MVImputer returns a single imputation. To return multiple imputations, the MVImputer can be applied to the same data in a number of loops and the output saved in a dictionary or list of dataframes. See *Appendix A: Data Pre-processing* for how to use the MVImputer for multiple imputation, and *Appendix B* for the class definition.

The BaseMLM class It is the Parent class for MVR, and it is where all the computation for the multivariate analysis is done. It has the following methods: (a) *the beta methods*: for computing unstandardized and standardized regression weights; (b) *the sums of squares methods*: for computing sum of squares and cross-products total, regression, and residual; (c) *the variance and statistical methods*: for computing residual variance, residual standard error, coefficient of determination, F-statistic, variance-covariance matrix, robust standard errors, t-statistic, p-value, and confidence intervals; and (d) *loglikelihood methods*: for computing the -loglikelihood, AIC and BIC. See *Appendix C* for the class definition.

The MLM class It is the Child class of the BaseMLM class. It has three methods: (a) *model estimates*: compiles a summary of the multivariate regression – regression coefficients, standard errors, t-statistics, p-value, 95% confidence interval, and standardized regression coefficients; (b) *model stats*: gets a summary of the model statistics – residual standard error and associated degrees of freedom, the F-statistics, its degrees of freedom and p-value, the r-squared and adjusted-rsquared; and (c) *model_summary*: provides a summary of model coefficient estimates, and goodness of fit indices. The model_summary is the method that should be called to run the multivariate regression. It outputs a tuple of three tables – model estimates,

Listing 4 ■ Running the MVR analyses (continued)

```
# print -2loglik, aic and bic
Figure("-2loglik, AIC, BIC")
print(loglik.to_markdown(tablefmt="github", floatfmt=(".2f"),
      straligna="center", numalign="center"))

##Figure 7
##-2Loglik, AIC, BIC
## |          |  -2logLik   |   AIC    |   BIC    |
## |----------|------------|----------|----------|
## | p_health | -24518.16  | 49048.00 | 49096.00 |
## | m_health | -22351.41  | 44715.00 | 44762.00 |
```

model statistics, and model -loglikelihood, AIC and BIC estimates. See *Appendix C* for the class definition.

The MVR Analyses

To run the multivariate analyses (see Listings 3 and 4), we initialize the MLM class and then call the model_summary method. The MLM class has the following arguments: data, endog (the responses), exog (the predictors) and errortype (type of standard error estimate). The endog parameter should indicate the number of response variables, and the exog parameter must be the same value as the endog parameter. All this has to do with how Python does its indexing, starting from zero and counting up to, but excluding the last index value. In the example below, endog = 2 means Python will read the first two columns in the data as including index 0 and 1, but not 2; exog = 2 means start indexing the predictors from column index number 2 to the end, after counting from 0 and 1.

As well, we choose errortype = HC3. There are five errortypes (standard, HC0, HC1, HC2, and HC3), with standard (the default), being a non-robust standard error. For more information about robust standard errors, refer to the following papers (Hayes & Cai, 2007; MacKinnon & White, 1985; Mansournia et al., 2021). The MVR data must include only the variables to be used in the model, with the response variables in the first N columns, and the predictors in the last N columns after the response variables.

The p-values for the F statistic (Figure 4) for both physical ($F_{5,20247} = 492.894$, $p < .0001$) and mental health ($F_{5,20247} = 1882.24$, $p < .0001$) suggests that there is an association between at least one of the predictors and the two response variables. R^2 was 0.11 and 0.32 for physical and mental health, respectively.

The outputs (Figures 5 and 6 with 95% confidence interval) suggest that age group was positively assciated with physical ($\beta = 0.11, [0.09, 0.13]$) and mental health ($\beta = 0.07, [0.05, 0.09]$). Gender was not significantly associated with physical health ($\beta = 0.01, [-0.02, 0.05]$), but was negatively correlated with mental health ($\beta = -0.04, [-0.08, -0.01]$). Education had a significant negative effect on physical ($\beta = -0.14, [-0.17, -0.12]$) and mental health ($\beta = -0.17, [-0.19, -0.14]$). Social support concerns were positively associated with children's physical ($\beta = 0.10, [0.10, 0.11]$) and mental health ($\beta = 0.12, [0.11, 0.13]$). Parenting concerns were positively correlated with children's physical ($\beta = 0.04, [0.03, 0.04]$) and mental health ($\beta = 0.14, [0.13, 0.14]$).

Linear Hypothesis Testing

Hypothesis is conducted using the mv_test() function, which takes three arguments: L, M, and C. The L is the left-hand side contrast matrix. If 2D array, each row is an hypotheses and each column is an independent variable. At least 1 row (1 by the number of independent variables) is required. M is also the left hand side transform matrix. If None or left out, it is set to a k_endog by k_endog identity matrix (i.e. do not transform y matrix). C is the right-hand side constant matrix. if None or left out it is set to a matrix of zeros. It must have the same number of rows as L and the same number of columns as M. See the mv_test() guide for more information.

1. Global test of association We consider whether there is a linear association between the response variables and the predictor variables. Specifically, we consider the null hypothesis that none of the predictors are correlated with the responses, H_0: $\beta_1 = \beta_2, ..., \beta_5 = 0$. We do not wish to include $\beta_0 = 0$ in the hypothesis because that would restrict all the responses to have intercepts of zero.

To run the mv_test(), we first run the _MultivariateOLS.from_formula model statement as shown below. There are two response variables (p_health and m_health) on the left hand side of the tilde (~) sign. The predictor variables (age_group, gender, education, soc_concerns and par_concerns) are on the right hand side of the

Listing 5 ■ Performing global tests of association

```
# Import the modules for the hypothesis testing
from statsmodels.multivariate.manova import MANOVA
from statsmodels.multivariate.multivariate_ols import _MultivariateOLS

# run the manova formula function
mod = _MultivariateOLS.from_formula("p_health + m_health ~ agegrp + gender + "
                                     "educ + ss_concerns + p_concerns", df)

# the hypothesis matrix
L = ["agegrp", "gender", "educ", "ss_concerns, p_concerns"]
M = None
C = None

# fit the model, run the hypothesis test and print the test output
r = mod.fit(method="svd")
r0 = r.mv_test(hypotheses=[("Global test of association", L, M, C)])
Figure("Global test of association")
print(r0.summary(show_contrast_L=False, show_transform_M=False, show_constant_C=False))

##Figure 8
##Global Test of Association
##                       Multivariate linear model
## ======================================================================
##
## ----------------------------------------------------------------------
##  Global test of association Value   Num DF   Den DF    F Value  Pr > F
## ----------------------------------------------------------------------
##              Wilks' lambda 0.6690 10.0000 40492.0000  901.1994 0.0000
##             Pillai's trace 0.3357 10.0000 40494.0000  816.9263 0.0000
##     Hotelling-Lawley trace 0.4875 10.0000 30366.2501  986.9412 0.0000
##        Roy's greatest root 0.4723  5.0000 20247.0000 1912.5904 0.0000
## ======================================================================
```

tilde sign. If there are no arguments in the `mv_test()` function, it defaults to testing the effect of each independent variable on the response variables, assuming a null hypothesis of no linear association. Multivariate test statistics for each of the predictors, including the intercept are then produced.

The results, Figure 8, suggests rejecting the null hypothesis of no association because all the p-values associated with the F statistic are significant. MANOVA test statistics are affected by the violation of homogeneity of covariance and normality assumptions. For example, Wilks' lambda and Hotelling's trace are sensitive to violations of homogeneity of covariances in smaller samples and Roy's largest root increases the likelihood of *Type I* errors. Thus, Olson (1976) recommends Pillai's trace for general use and Ateş et al. (2019) also suggested that Pillai's trace gives more robust results in the case of homogeneous and heterogeneous variances.

2. Identical slopes for physical and mental health We test if the intercepts and slopes are identical for the response variables. We are testing the null hypothesis that H_0: $\beta_{01} = \beta_{02}, \beta_{11} = \beta_{12}, ..., \beta_{15} = \beta_{25}$. The L matrix for this test is an an identity matrix of dimension 6 (I_6). The `L` matrix can be specified using the names of the predictors as shown below. The `M` matrix is a two row-matrix with an element in each row. The element in the first row is associated with the first response variable, and the element in the second row is for mental health. Just like the `L` matrix, the names of the response variables can be used.

The output in Figure 9 above indicates that at least one of the regression slopes for physical and mental health are not identical given that $p < .0001$ for all the four multivariate tests.

3. Equality of predictor coefficients for responses We are interested in assessing if the effects of each predictor on each of the responses, controlling for other predictors are equal; that is, if $\beta_{ij} = \beta_{ik}$, where ij represents the ef-

Listing 6 ■ Running tests of slopes

```
# the hypothesis matrix
L = ["Intercept", "agegrp", "gender", "educ", "ss_concerns, p_concerns"]
M = ["p_health - m_health"]
C = None

# run the hypothesis and print the output
r1 = r.mv_test(hypotheses=[("Identical slopes for responses", L, M, C)])
Figure("Identical slopes for responses")
print(r1.summary(show_contrast_L=False, show_transform_M=False, show_constant_C=False))

##Figure 9
##Identical Slopes for Responses
##                           Multivariate linear model
## ========================================================================
##
## ------------------------------------------------------------------------
##  Identical slopes for responses Value  Num DF   Den DF    F Value  Pr > F
## ------------------------------------------------------------------------
##                 Wilks' lambda 0.6408 6.0000 20247.0000 1891.5177 0.0000
##                Pillai's trace 0.3592 6.0000 20247.0000 1891.5177 0.0000
##        Hotelling-Lawley trace 0.5605 6.0000 20247.0000 1891.5177 0.0000
##           Roy's greatest root 0.5605 6.0000 20247.0000 1891.5177 0.0000
## ========================================================================
```

fects of predictor i on response j, and ik represents the effects of the same predictor i on response k. For illustrative purposes, we will look at the coefficients for gender and education.

Figure 10 suggests that the coefficients associated with gender for physical $(\beta = 0.01)$ and mental health $(\beta = -0.04)$ are significantly different from each other. Next, we examine the equality of education effects on the responses.

Figure 11 shows that the coefficients associated with education for physical $(\beta = -0.14)$ and mental health $(\beta = -0.17)$ are not significantly different from each other.

Discussion

The goals of this tutorial were two-fold: to show how to perform a multivariate regression using custom-defined classes in Python and to demonstrate how to conduct linear hypothesis testing using the statsmodels library. To conduct the multivariate regression in Python, the author defined three classes: `MVImputer`, `BaseMLM`, and `MLM`. The `MVImputer` class could be used for multivariate missing values imputation; the `BaseMLM` class does all the computation for the multivariate regression; and the `MLM` class summarizes all the outputs from the `BaseMLM` class. A limitation of the `BaseMLM` class is that an interaction term cannot be specified directly in the model. A work-around would be to include the interaction term as a predictor variable in the data one intends to use. Additionally, using the statsmodels library, this tutorial was able to demonstrate how to test multivariate hypothesis about regression coefficients.

In conclusion, via this tutorial, we hope to have provided the reader with an accessible way to conduct multivariate regression in Python and test hypothesis about coefficients. As a corollary, we also hope that this tutorial encourages the reader in their to journey to learn and write code with Python.

References

Ateş, C., Kaymaz, Ö., Kale, H. E., & Tekindal, M. A. (2019). Comparison of test statistics of nonnormal and unbalanced samples for multivariate analysis of variance in terms of Type-I error rates. *Computational and Mathematical Methods in Medicine*, *2019*, 1–8. doi: 10.1155/2019/2173638.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Rio, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. doi: 10.1038/s41586-020-2649-2.

Hayes, A. F., & Cai, L. (2007). Using heteroskedasticity-consistent standard error estimators in ols regression:

Listing 7 ■ Testing equality of predictor coefficients (first part)

```
# the hypothesis matrix
L = ["gender"]
M = ["p_health - m_health"]
C = None

# run the hypothesis and print the output
r2 = r.mv_test(hypotheses=[("Equal gender coefs for responses", L, M, C)])
Figure("Equal gender coefs for responses")
print(r2.summary(show_contrast_L=False, show_transform_M=False, show_constant_C=False))

##Figure 10
##Equal Gender Coefs for Responses
##                           Multivariate linear model
## =========================================================================
##
## -------------------------------------------------------------------------
##  Equal gender coefs for responses Value  Num DF   Den DF   F Value Pr > F
## -------------------------------------------------------------------------
##                     Wilks' lambda 0.9996 1.0000 20247.0000  7.3540 0.0067
##                    Pillai's trace 0.0004 1.0000 20247.0000  7.3540 0.0067
##            Hotelling-Lawley trace 0.0004 1.0000 20247.0000  7.3540 0.0067
##               Roy's greatest root 0.0004 1.0000 20247.0000  7.3540 0.0067
## =========================================================================
```

An introduction and software implementation. *Behavior Research Methods*, *39*(4), 709–722. doi: 10.3758/bf03192961.

Hidalgo, B., & Goodman, M. (2013). Multivariate or multivariable regression? *American Journal of Public Health*, *103*(1), 39–40. doi: 10.2105/AJPH.2012.300897.

MacKinnon, J. G., & White, H. (1985). Some heteroskedasticity-consistent covariance matrix estimators with improved finite sample properties. *Journal of Econometrics*, *29*(3), 305–325. doi: 10.1016/0304-4076(85)90158-7.

Mansournia, M. A., Nazemipour, M., Naimi, A. I., Collins, G. S., & Campbell, M. J. (2021). Reflection on modern methods: Demystifying robust standard errors for epidemiologists. *International Journal of Epidemiology*, *50*(1), 346–351. doi: 10.1093/ije/dyaa260.

McKinney, W. (2010). Data structures for statistical computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). doi: 10.25080/Majora-92bf1922-00a.

Olson, C. L. (1976). On choosing a test statistic in multivariate analysis of variance. *Psychological Bulletin*, (4), 579–586. doi: 10.1037/0033-2909.83.4.579.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830. doi: 10.5555/1953048.2078195.

Posit Team. (2023). *RStudio: Integrated Development Environment for R*. Version Version 2023.6.0.421, Mountain Hydrangea. Posit Software, PBC. Boston, MA. http://www.posit.co/

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 92–96). doi: 10.25080/Majora-92bf1922-011.

Statistics Canada. (2020, August). Crowdsourcing: Impacts of COVID-19 on Canadians' Parenting During the Pandemic Public Use Microdata File. Public use microdata: 45-25-0006. doi: 10.25318/45250006-eng.

The Pandas Development Team. (2023, May). pandas-dev/pandas: Pandas. doi: 10.5281/zenodo.3509134.

Ushey, K., Allaire, J., & Tang, Y. (2023). *reticulate: Interface to 'Python'*. Version Version 1.30. https://CRAN.R-project.org/package=reticulate

Vallat, R. (2018). Pingouin: Statistics in Python. *The Journal of Open Source Software*, *3*(31), 1026. doi: 10.21105/joss.01026.

Van Rossum, G., & Drake Jr, F. L. (2009). *Python 3 reference manual*. CreateSpace.

Wu, L., & Qui, J. (2021). *Applied multivariate statistical analysis and related topics with R*. EDP Sciences. doi: doi:10.1051/978-2-7598-2602-5.

Listing 8 ■ Testing equality of predictor coefficients (continued)

```
# the hypothesis matrix
L = ["educ"]
M = ["p_health - m_health"]
C = None

# run the hypothesis and print the output
r3 = r.mv_test(hypotheses=[("Equal educ coefs for responses", L, M, C)])
Figure("Equal education coefs for responses")
print(r3.summary(show_contrast_L=False, show_transform_M=False, show_constant_C=False))

##Figure 11
##Equal Education Coefs for Responses
##                         Multivariate linear model
## ========================================================================
##
## ------------------------------------------------------------------------
##  Equal educ coefs for responses Value  Num DF   Den DF   F Value Pr > F
## ------------------------------------------------------------------------
##                   Wilks' lambda 0.9999 1.0000 20247.0000  2.6817 0.1015
##                  Pillai's trace 0.0001 1.0000 20247.0000  2.6817 0.1015
##          Hotelling-Lawley trace 0.0001 1.0000 20247.0000  2.6817 0.1015
##             Roy's greatest root 0.0001 1.0000 20247.0000  2.6817 0.1015
## ========================================================================
```

Appendix A: Data Pre-processing

```
# ------------------------------------------------------------------ #
# Data Pre-Processing                                                #
# ------------------------------------------------------------------ #

# import data
#============
# import the modules
import pandas as pd
import numpy as np

# class to automate table numbers
class Table:
    value = 0
    def __new__(cls, caption):
        cls.value += 1
        print(f"Table {cls.value}: {caption}")

# cols to use in analysis
subset_cols = ["HSI_10A", "HSI_10B", "DEM_10", "PED_05", "PFC_G2", "PAGE",
               "HSI_15A", "HSI_15B", "HSI_15E", "HSI_15C", "HSI_15D", "HSI_15F"]

dfile_path = "D:/--data/covid/iccpdp2020rccpdp_p.csv"
df = pd.read_csv(dfile_path)[subset_cols]
Table("View of selected data columns")
print(df.iloc[:, 0:9].head(3).to_markdown(tablefmt="github", floatfmt=(".2f"),
       stralign="center", numalign="center", index=False))

# get the descriptives for the data
# ============================================================================
# variable labels
var_labels = ["chd: general physical health", "chd: general mental health",
              "gender benchmarked to sex", "highest educ completed",
              "liv'n household: 6 to 14 yrs old", "age group",
              "staying connected fam/friends", "getting along/supporting each other",
              "feelng lonely in home", "balancing chdcare/school/work",
              "managing chd beh/emotions", "having less patience with chd"]

# get descriptives, drop some columns and add variable labels
desc = df.describe(include="all").T
desc.drop(columns=["25%", "50%", "75%"], inplace=True)
desc.insert(loc=0, column="vlabels", value=var_labels)
Table("Descriptives")
print(desc.to_markdown(tablefmt="github", floatfmt=(".2f"),
       stralign="center", numalign="center"))


# get unique values for all columns
# ===============================
Table("Unique data values")
df.apply(lambda x: np.sort(x.unique()))

# create a mask and remove all values greater than or equal to 5
# ============================================================
msk = df.ge(5).any(axis=1)
df2 = df[~msk]

# select responses for 6 to 14 yr old households
# ============================================
df3 = df2[df2["PFC_G2"] == 2]

# get the descriptives again after dropping observations
# ====================================================
desc2 = df3.describe(include="all").T
desc2.drop(columns=["25%", "50%", "75%"], inplace=True)
desc2.insert(loc=0, column="vlabels", value=var_labels)
Table("Descriptives - cleaned data")
print(desc2.to_markdown(tablefmt="github", floatfmt=(".2f"),
      stralign="center", numalign="center"))


# dummify columns
# ==============
def dummify(d):
  # dummy-code two variables using dictionary comprehension
  lst1 = ["DEM_10"]
  lst2 = ["PED_05"]
  # recode 2 into 0, 1 = 1
  d2 = {c: d[c].apply(lambda x: 0 if x == 2 else 1) for c in lst1}
  # recode 1 into 0, 2 into 1
  d3 = {c: d[c].apply(lambda x: 0 if x == 1 else 1) for c in lst2}
  d2b = pd.concat([v for v in d2.values()], axis=1)
  d3b = pd.concat([v for v in d3.values()], axis=1)
  dfs = [d2b, d3b]
  # add suffix to new variable names before merging them
  dfs = [dfs[i].add_suffix("_") for i, c in enumerate(dfs)]
  return d.join(dfs[0:])

# call dummify function
df4 = dummify(df3)
Table("View of dummy-coded variables")
df4.apply(lambda x: np.sort(x.unique())).tail(2)

# a. import the imputation class
# ==============================
from mvimputer import MVImputer

# b. check columns for missing values
print(df4.isnull().sum())
```

```
# c1. single data imputation
mvi = MVImputer(random_state=None)
df4 = mvi.fit_transform(df4)

# c2. 5 multiple data imputations saved in dict
mvi = MVImputer(random_state=None)
dframes = {d: mvi.fit_transform(df4) for d in range(5)}

# d1. verify single data imputation
print(df4.isnull().sum())

# d2. verify multiple data imputation
for k, v in dframes.items():
    print(k, v.isnull().sum())

# summation index for social support and parenting concerns
# ========================================================
df4["ss_concerns"] = np.sum(df4[["HSI_15A", "HSI_15B", "HSI_15E"]], axis=1)
df4["p_concerns"] = np.sum(df4[["HSI_15C", "HSI_15D", "HSI_15F"]], axis=1)

# rename columns using dictionary comprehension
col_names = {"HSI_10A": "p_health", "HSI_10B": "m_health",
             "PAGE": "agegrp", "DEM_10_": "gender",
             "PED_05_": "educ"}
df4.columns = [col_names.get(x, x) for x in df4.columns]

# columns to drop: dummified columns, subset for children, summation index cols
cols_to_drop = ["DEM_10", "PED_05", "PFC_G2", "HSI_15A", "HSI_15B",
                "HSI_15E","HSI_15C", "HSI_15D", "HSI_15F"]
df5 = df4.drop(columns=cols_to_drop)

var_labels = ["chd: general physical health", "chd: general mental health",
              "age group", "gender benchmarked to sex", "highest educ completed",
              "social support concerns", "parenting concerns"]

desc3 = df5.describe(include="all").T
desc3.drop(columns=["25%", "50%", "75%"], inplace=True)
desc3.insert(loc=0, column="vlabels", value=var_labels)
Table("Descriptives of subsetted data")
print(desc3.to_markdown(tablefmt="github", floatfmt=(".2f"),
      stralign="center", numalign="center"))

# compute correlations
# ====================
corr = df.corr(method="pearson")
Table("Correlations among variables")
print(corr.to_markdown(tablefmt="github", floatfmt=(".2f"),
       stralign="center", numalign="center"))
```

Appendix B: MVImputer class

```
# ------------------------------------------------------------------ #
# Class for multivariate imputation, that imputes missing values as a #
# function of all other variables in the model                        #
# ------------------------------------------------------------------ #

# mising values imputation
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.base import BaseEstimator, TransformerMixin

# class for missing data imputation
# ==================================================
class MVImputer(BaseEstimator, TransformerMixin):
    """
    A multivariate imputation approach
    :returns: A DataFrame of imputed values.
    """
    def __init__(self, random_state=None, fill='NA'):
        self.random_state = random_state
        self.fill = fill

    def fit(self, X, y=None):
        categorical_dtypes = ['object', 'category', 'bool']
        numerical_dtypes = ['float', 'int']
        for col in X.columns:
            if X[col].dtype.name in categorical_dtypes:
                self.fill = X.mode().iloc[0]
            elif X[col].dtype.name in numerical_dtypes:
                min_val = X[col].min(axis=0)
                max_val = X[col].max(axis=0)
                imputer = (IterativeImputer(max_iter=10,
                                            random_state=self.random_state,
                                            min_value=min_val,
                                            max_value=max_val))
                self.fill = imputer.fit(X)
        return self

    def transform(self, X, y=None):
        return X.fillna(self.fill)
```

Appendix C: BaseMLM and MLM classes

```
# ------------------------------------------------------------------- #
# Parent BaseMLM Class and Child MLM Class that Runs the Multivariate #
# Regression in Python                                                #
# ------------------------------------------------------------------- #

# import the modules
import pandas as pd
import numpy as np
from numpy.linalg import inv, qr, solve, lstsq

import scipy.stats as stats
from scipy.linalg import solve_triangular

# multivariate anova test
from statsmodels.multivariate.manova import MANOVA
from statsmodels.multivariate.multivariate_ols import _MultivariateOLS


# BaseMLM class for multivariate linear model
# ======================================================================
class BaseMLM:
    """
    A Multivariate Linear Model (MLM) class that implements methods for
    MLM estimation.
    :return: Methods for MLM.
    """

    def __init__(self, data, endog=2, exog=2, errortype="standard"):
        self.data = data
        self.endog = endog
        self.exog = exog
        self.errortype = errortype

        self.y = self.data.iloc[:, :self.endog]
        self.X = self.data.iloc[:, self.exog:]
        self.X.insert(0, 'intercept', np.ones((len(self.X),), dtype=int))
        self.nobs = self.y.shape[0]
        self.nvars = self.X.shape[1]
        self.dfe = self.nobs - self.nvars
        self.dfr = self.nvars - 1
        self.H = self.X.dot(inv(self.X.T.dot(self.X)).dot(self.X.T))

        self.beta = self._beta()
        self.std_beta = self._std_beta()

        self.yhat = np.dot(self.X, self.beta)
        self.ybar = pd.DataFrame([self.y.mean()] * self.y.shape[0])

        self.sscp_tot = self._sscp_tot()
        self.sscp_reg = self._sscp_reg()
        self.resid = self.y - self.yhat
        self.sscp_resid = self._sscp_resid()
        self.ev = self.sscp_resid / self.dfe
        self.resid_var = self._resid_var()
        self.rse = self._rse()

        self.rsquared = self._rsquared()
        self.fstats = self._fstats()
        self.all_stats = self._all_stats()

        self.vcov = self._vcov()
        self.se = self._se()
        self.tstats = self._tstats()
        self.pvalue = self._pvalue()
        self.confints = self._confints()
        self.loglik = self._loglik_aic_bic()

    def _beta(self):
        if len(self.y.shape) == 1 or self.y.shape[1] == 1:
            raise ValueError("There must be more than one response variable"
                             "to fit a multivariate linear model")
        Q, R = qr(self.X)
        beta = pd.DataFrame(solve_triangular(R, Q.T.dot(self.y)))
        beta.columns = list(self.y.columns)
        beta.index = list(self.X.columns)
        return beta

    def _std_beta(self):
        sx = np.std(self.X, axis=0)
        sy = np.std(self.y, axis=0)
        stdB = self.beta.mul(sx, axis=0) / sy
        return stdB

    def _sscp_tot(self):
        total = self.y - self.ybar
        sscp_tot = pd.DataFrame(np.dot(total.T, total))
        sscp_tot.columns = list(self.y.columns)
        sscp_tot.index = list(self.y.columns)
        return sscp_tot

    def _sscp_reg(self):
        reg = self.yhat - self.ybar
        sscp_reg = pd.DataFrame(np.dot(reg.T, reg))
        sscp_reg.columns = list(self.y.columns)
        sscp_reg.index = list(self.y.columns)
        return sscp_reg

    def _sscp_resid(self):
        sscp_resid = pd.DataFrame(np.dot(self.resid.T, self.resid))
        sscp_resid.columns = list(self.y.columns)
        sscp_resid.index = list(self.y.columns)
        return sscp_resid

    def _resid_var(self):
        evar = pd.DataFrame(np.diag(self.ev), columns=["evar"], index=[list(self.y.columns)])
        evar[["k", "dfe"]] = [self.nvars, self.dfe]
        cols = ["k", "dfe", "evar"]
        evar = evar[cols]
        evar.index = evar.index.get_level_values(0)
        return evar

    def _rse(self):
        rse = pd.Series(np.sqrt(np.diag(self.ev)), name="rse")
        rse.index = list(self.y.columns)
        return rse

    def _rsquared(self):
        R2 = pd.Series(1 - self.resid.var() / self.y.var(), name='r2')
        R2.index = list(self.y.columns)
        adjR2 = pd.Series(1 - (1 - R2) * ((self.nobs - 1) / (self.nobs - self.nvars - 1)), name='adjr2')
        adjR2.index = list(self.y.columns)
        rsquare = round(pd.concat([R2, adjR2], axis=1), 3)
        return rsquare

    def _fstats(self):
        # F = (R2 / dfr) / ((1 - R2) / dfe)
        F = np.diag( (self.sscp_reg / self.dfr) / (self.sscp_resid / self.dfe) )
        # find p-value of F test statistic
        p = np.round(1 - stats.f.cdf(F, dfn=self.dfr, dfd=self.dfe), 5)
        fstat = round(pd.DataFrame([F, p]).T, 3)
        fstat[["num", "denom"]] = [self.dfr, self.dfe]
        fstat.index = list(self.y.columns)
        fstat.columns = ["f-stat", "p-value", "num", "denom"]
        cols = ["num", "denom", "f-stat", "p-value"]
        fstat = fstat[cols]
        return fstat

    def _all_stats(self):
        resid_var_rse = self.resid_var.merge(self.rse, left_index=True, right_index=True)
        fstat_rsq = pd.concat([self.fstats, self.rsquared], axis = 1)
        resid_var_fstat_rsq = resid_var_rse.merge(fstat_rsq, left_index=True, right_index=True)
        return resid_var_fstat_rsq

    def _vcov(self):
        rnames = list(self.beta.index)
        if self.errortype == "standard":
            evd = np.diag(self.ev)
            vcv = {}
            for i in evd:
                vcv[i] = pd.DataFrame([i] * inv(self.X.T @ self.X))
            for v in vcv.values():
                v.columns = rnames
                v.index = rnames
            return vcv
        elif self.errortype == "HC0":
            hc0 = self.resid**2
            vcv = {}
            for col in hc0:
                vcv[col] = (inv(self.X.T @ self.X) @ self.X.T * hc0[col] @ self.X
                            @ inv(self.X.T @ self.X))
            for v in vcv.values():
                v.columns = rnames
                v.index = rnames
            return vcv
        elif self.errortype == "HC1":
            hc1 = self.resid**2
            vcv = {}
            for col in hc1:
                vcv[col] = ((self.nobs / (self.nobs - self.nvars - 1)) * (inv(self.X.T @ self.X) @
                             self.X.T * hc1[col] @ self.X @ inv(self.X.T @ self.X)))
            for v in vcv.values():
                v.columns = rnames
                v.index = rnames
            return vcv
        elif self.errortype == "HC2":
            hc2 = np.transpose(np.transpose(self.resid**2) / (1 - np.diag(self.H)))
            vcv = {}
            for col in hc2:
                vcv[col] = (inv(self.X.T @ self.X) @ (self.X.T * (hc2[col]) @ self.X)
                            @ inv(self.X.T @ self.X))
            for v in vcv.values():
                v.columns = rnames
                v.index = rnames
            return vcv
        elif self.errortype == "HC3":
            hc3 = np.transpose(np.transpose(self.resid**2) / (1 - np.diag(self.H))**2)
            vcv = {}
            for col in hc3:
                vcv[col] = (inv(self.X.T @ self.X) @ (self.X.T * (hc3[col]) @ self.X)
                           @ inv(self.X.T @ self.X))
            for v in vcv.values():
                v.columns = rnames
                v.index = rnames
            return vcv

    def _se(self):
        if self.errortype == "standard":
            vcv = self.vcov
            SE = {}
            for idx, vals in vcv.items():
                SE[idx] = pd.DataFrame(np.sqrt(np.diag(vals)))
                SE[idx].index = list(self.beta.index)
            SE = pd.concat([v for k,v in SE.items()], axis=1)
            SE.columns = self.beta.columns
            return SE
        elif self.errortype == "HC0":
            vcv = self.vcov
            SE = {}
            for idx, vals in vcv.items():
                SE[idx] = pd.DataFrame(np.sqrt(np.diag(vals)))
                SE[idx].index = list(self.beta.index)
            SE = pd.concat([v for k,v in SE.items()], axis=1)
            SE.columns = self.beta.columns
            return SE
        elif self.errortype == "HC1":
            vcv = self.vcov
            SE = {}
            for idx, vals in vcv.items():
                SE[idx] = pd.DataFrame(np.sqrt(np.diag(vals)))
                SE[idx].index = list(self.beta.index)
            SE = pd.concat([v for k,v in SE.items()], axis=1)
            SE.columns = self.beta.columns
            return SE
        elif self.errortype == "HC2":
            vcv = self.vcov
            SE = {}
            for idx, vals in vcv.items():
                SE[idx] = pd.DataFrame(np.sqrt(np.diag(vals)))
                SE[idx].index = list(self.beta.index)
            SE = pd.concat([v for k,v in SE.items()], axis=1)
            SE.columns = self.beta.columns
            return SE
        elif self.errortype == "HC3":
```

```
        vcv = self.vcov
        SE = {}
        for idx, vals in vcv.items():
            SE[idx] = pd.DataFrame(np.sqrt(np.diag(vals)))
            SE[idx].index = list(self.beta.index)
        SE = pd.concat([v for k,v in SE.items()], axis=1)
        SE.columns = self.beta.columns
        return SE

def _tstats(self):
    if self.errortype == "standard":
        tval = self.beta / self.se
        return tval
    elif self.errortype == "HC0":
        tval = self.beta / self.se
        return tval
    elif self.errortype == "HC1":
        tval = self.beta / self.se
        return tval
    elif self.errortype == "HC2":
        tval = self.beta / self.se
        return tval
    elif self.errortype == "HC3":
        tval = self.beta / self.se
        return tval

def _pvalue(self):
    if self.errortype == "standard":
        pval = pd.DataFrame(np.round(stats.t.sf(abs(self.tstats), self.nobs - self.nvars), 4) * 2)
        pval.columns = self.beta.columns
        pval.index = self.beta.index
        return pval
    elif self.errortype == "HC0":
        pval = pd.DataFrame(np.round(stats.t.sf(abs(self.tstats), self.nobs - self.nvars), 4) * 2)
        pval.columns = self.beta.columns
        pval.index = self.beta.index
        return pval
    elif self.errortype == "HC1":
        pval = pd.DataFrame(np.round(stats.t.sf(abs(self.tstats), self.nobs - self.nvars), 4) * 2)
        pval.columns = self.beta.columns
        pval.index = self.beta.index
        return pval
    elif self.errortype == "HC2":
        pval = pd.DataFrame(np.round(stats.t.sf(abs(self.tstats), self.nobs - self.nvars), 4) * 2)
        pval.columns = self.beta.columns
        pval.index = self.beta.index
        return pval
    elif self.errortype == "HC3":
        pval = pd.DataFrame(np.round(stats.t.sf(abs(self.tstats), self.nobs - self.nvars), 4) * 2)
        pval.columns = self.beta.columns
        pval.index = self.beta.index
        return pval

def _confints(self):
    if self.errortype == "standard":
        tcrit = stats.t.ppf(q=1-.05/2, df=self.nobs - self.nvars)
        lwr = (self.beta - tcrit * self.se)
        upr = (self.beta + tcrit * self.se)
        return lwr, upr
    elif self.errortype == "HC0":
        tcrit = stats.t.ppf(q=1-.05/2, df=self.nobs - self.nvars)
        lwr = (self.beta - tcrit * self.se)
        upr = (self.beta + tcrit * self.se)
        return lwr, upr
    elif self.errortype == "HC1":
        tcrit = stats.t.ppf(q=1-.05/2, df=self.nobs - self.nvars)
```

```
            lwr = (self.beta - tcrit * self.se)
            upr = (self.beta + tcrit * self.se)
            return lwr, upr
        elif self.errortype == "HC2":
            tcrit = stats.t.ppf(q=1-.05/2, df=self.nobs - self.nvars)
            lwr = (self.beta - tcrit * self.se)
            upr = (self.beta + tcrit * self.se)
            return lwr, upr
        elif self.errortype == "HC3":
            tcrit = stats.t.ppf(q=1-.05/2, df=self.nobs - self.nvars)
            lwr = (self.beta - tcrit * self.se)
            upr = (self.beta + tcrit * self.se)
            return lwr, upr

    def _loglik_aic_bic(self):
        if len(self.y.shape) == 1 or self.y.shape[1] == 1:
            raise ValueError("There must be more than one response variable"
                             "to fit a multivariate linear model")
        n = self.nobs
        n2 = n/2
        k = self.nvars
        mod = lstsq(self.X, self.y, rcond=None)
        rss = mod[1]
        ll = -n2 * np.log(2 * np.pi) - n2 * np.log(rss/n) - n2
        aic = np.round(-2 * ll + 2 * k, 0)
        bic = np.round(-2 * ll + np.log(n) * k, 0)
        return pd.DataFrame({"-2logLik": ll, "AIC": aic, "BIC":bic},
                             index=list(self.y.columns))

# MLM child class
# ======================================================================
class MLM(BaseMLM):
    """
    An MLM subclass for BaseMLM. Aggregates methods specific to the MLM estimation.
    : returns summary methods for MLM
    """

    def __init__(self, data, endog=2, exog=2, errortype="standard"):
        super().__init__(data, endog=endog, exog=exog, errortype=errortype)
        self.model_estimates = self._model_estimates()
        self.model_stats = self._model_stats()
        self.model_summary = self.model_summary()

    def _model_estimates(self):
        if self.errortype == "standard":
            dflist = [self.beta,
                      self.se,
                      self.tstats,
                      self.pvalue,
                      self.confints[0],
                      self.confints[1],
                      self.std_beta]
            keys = list(self.y.columns)
            df2 = {keys: value for keys, value in enumerate(dflist)}
            results = {}
            for i in keys:
                results[i] = pd.DataFrame({x: df2[x][i] for x in df2})
            cols = ['Est.', 'Std Error', 't', 'Pr(>|t|)', '0.025', '0.975', 'Std Est.']
            for v in results.values():
                v.columns = cols
            return results
        elif self.errortype == "HC0":
            dflist = [self.beta,
                      self.se,
                      self.tstats,
                      self.pvalue,
                         self.confints[0],
                         self.confints[1],
                         self.std_beta]
            keys = list(self.y.columns)
            df2 = {keys: value for keys, value in enumerate(dflist)}
            results = {}
            for i in keys:
                results[i] = pd.DataFrame({x: df2[x][i] for x in df2})
            cols = ['Est.', 'Robust SE', 't', 'Pr(>|t|)', '0.025', '0.975', 'Std Est.']
            for v in results.values():
                v.columns = cols
            return results
        elif self.errortype == "HC1":
            dflist = [self.beta,
                         self.se,
                         self.tstats,
                         self.pvalue,
                         self.confints[0],
                         self.confints[1],
                         self.std_beta]
            keys = list(self.y.columns)
            df2 = {keys: value for keys, value in enumerate(dflist)}
            results = {}
            for i in keys:
                results[i] = pd.DataFrame({x: df2[x][i] for x in df2})
            cols = ['Est.', 'Robust SE', 't', 'Pr(>|t|)', '0.025', '0.975', 'Std Est.']
            for v in results.values():
                v.columns = cols
            return results
        elif self.errortype == "HC2":
            dflist = [self.beta,
                         self.se,
                         self.tstats,
                         self.pvalue,
                         self.confints[0],
                         self.confints[1],
                         self.std_beta]
            keys = list(self.y.columns)
            df2 = {keys: value for keys, value in enumerate(dflist)}
            results = {}
            for i in keys:
                results[i] = pd.DataFrame({x: df2[x][i] for x in df2})
            cols = ['Est.', 'Robust SE', 't', 'Pr(>|t|)', '0.025', '0.975', 'Std Est.']
            for v in results.values():
                v.columns = cols
            return results
        elif self.errortype == "HC3":
            dflist = [self.beta,
                         self.se,
                         self.tstats,
                         self.pvalue,
                         self.confints[0],
                         self.confints[1],
                         self.std_beta]
            keys = list(self.y.columns)
            df2 = {keys: value for keys, value in enumerate(dflist)}
            results = {}
            for i in keys:
                results[i] = pd.DataFrame({x: df2[x][i] for x in df2})
            cols = ['Est.', 'Robust SE', 't', 'Pr(>|t|)', '0.025', '0.975', 'Std Est.']
            for v in results.values():
                v.columns = cols
            return results

    def _model_stats(self):
        """
        The summary of the model statistics: RSE, F, R , Adj. R
        :return: A DataFrame of model statistics
        """

        self.all_stats.drop(columns=["evar"], inplace=True)
        self.all_stats.rename({"k": "K",
                               "dfe": "DFE",
                               "rse": "RSE",
                               "num": "DF1",
                               "denom": "DF2",
                               "f-stat": "F",
                               "p-value": "P-Value",
                               "r2": "RSQ",
                               "adjr2": "Adj. RSQ"},
                               axis="columns", inplace=True)
        return self.all_stats

    def model_summary(self):
        """
        A summary of model coefficient estimates and goodness-of-fit indices.
        :return: A tuple of DataFrame model esimates and statistics.
        """

        table = self.model_estimates
        ftable = self.model_stats
        loglik = self.loglik

        return table, ftable, loglik
```

Open practices

The *Open Material* badge was earned because supplementary material(s) are available on the journal's web site.

Citation

Ashiabi, G. S. (2024). From scratch: A didactic example of multivariate regression in Python. *The Quantitative Methods for Psychology*, *20*(2), 186–204. doi: 10.20982/tqmp.20.2.p186.

Received: 25/05/2023 ∼ Accepted: 09/07/2024