

# joint-vs-individual-loading

April 8, 2021

## 1 Joint vs Individual Linear Regression Loadings

First, some notation. Be warned that I am going to take advantage of the typographical similarity between certain Greek and Roman letters (classicists and pedants, avert thy eyes!). Other than that, however, my notation should be very natural to anyone who's taken a college-level linear models course.

Afterward, I'll run some experiments where I prove (or at least state) some theoretical (ground-truth) results about the “Greek” quantities, then demonstrate them using their “Roman” counterparts calculated on toy datasets. To give a trivial example, to demonstrate that  $\sigma(\gamma) \geq 0$ , I could show that  $s(y) = 0.42$  (which is nonnegative) for some specific toy dataset.

### 1.1 The Greeks: Ground-Truth Data-Generating Process

Let  $\gamma, \chi_1, \chi_2, \varepsilon \mid \beta_0, \beta_1, \beta_2$  be a finite-variance (and nonzero-variance) Multivariate Normal “data-generating process” (basically, a vector of real-valued random variables) such that  $\gamma = \beta_0 + \beta_1\chi_1 + \beta_2\chi_2 + \varepsilon$  where  $\varepsilon$  is i.i.d. white noise.

Let  $\sigma(\cdot)$  represent standard deviation,  $\sigma^2(\cdot)$  represent variance,  $\sigma^2(\cdot, \cdot)$  represent covariance, and  $\rho(\cdot, \cdot)$  represent correlation. Let  $\Sigma$  be the variance-covariance matrix between  $\chi_1$  and  $\chi_2$  i.e.

$$\begin{pmatrix} \sigma^2(\chi_1) & \sigma^2(\chi_1, \chi_2) \\ \sigma^2(\chi_1, \chi_2) & \sigma^2(\chi_2) \end{pmatrix},$$

$\Omega$  be the corresponding correlation matrix, and  $\Sigma_\gamma$  be the column vector  $[\sigma^2(\chi_1, \gamma), \sigma^2(\chi_2, \gamma)]$  of covariances between the  $\chi$ 's and  $\gamma$ .

Importantly, assume that  $|\rho(\chi_1, \chi_2)| \neq 1$ .

### 1.2 The Romans: Practical Calculations On Observed Data

Suppose we observe  $N$  different “draws” or “samples” from this process. Arrange the draws of  $\gamma$  into a column vector of real numbers  $y := [y_n]$ , the draws of  $\chi_1$  into a column vector of real numbers  $x_1 := [x_{n,1}]$ , the draws of  $\chi_2$  into a column vector of real numbers  $x_2 := [x_{n,2}]$ , and the draws of  $\varepsilon$  into a column vector of real numbers  $e := [e_n]$ , over  $n \in [1, N]$ . Further, arrange the  $x$ 's into an  $N \times 3$  matrix of real numbers  $X := [1, x_1, x_2]$  whose first column is all ones (AKA “constant” AKA “intercept”).

Let the vector of real numbers  $b := [b_0, b_1, b_2] := (X^\top X)^{-1} X^\top y$  be the coefficients from an OLS linear regression of  $y$  onto  $X$  [1]. More generally, define `ols` such that e.g. `ols(y, [1, x_1, x_2]) := [b_0, b_1, b_2] =: b`.

Finally, let  $s$ ,  $s^2$ ,  $r$ ,  $S$ ,  $U$ , and  $S_y$  be the usual Bessel-corrected “sample” estimators of their “population” counterparts in Greek above.

```
[1]: from typing import Tuple, Union, Optional
import numpy as np
import pandas as pd
from numpy.linalg import inv
import statsmodels.api as sm

def gen_data(mean: Tuple[float]=(0, 0, 0), std: Tuple[float]=(1, 1, 1),
             corr12: float=0, corr13: float=0, corr23: float=0,
             b: Tuple[float]=(0, 1, 1, 1, 1), x3: bool=False,
             n: int=10_000_000, seed: int=42) -> \
    Tuple[pd.DataFrame, pd.DataFrame, pd.Series, pd.Series]:
    """
    Generate a toy dataset where  $y = b_0 + b_1x_1 + b_2x_2 [+ b_3x_3] +$ 
    ↪  $b_4 \cdot \text{white\_noise}$ .

    input
    -----
    mean: tuple[float] (default 0), ground-truth means of the  $x$ 's.
    std: tuple[float] (default 1), ground-truth standard deviations of the  $x$ 's.
    corr`ij`: float (default 0), ground-truth correlation between the  $x$ 's.
    b: tuple[float], ground-truth beta's.
    x3: bool, whether to use  $x_3$ .
    n: int (default 10 million), number of data points to generate.
    seed: int (default 42), random seed.

    output
    -----
    _X: pd.DataFrame, X: pd.DataFrame, white_noise: pd.Series, y: pd.Series.
    """
    mean = pd.Series({"x1": mean[0], "x2": mean[1], "x3": mean[2],
    ↪ "white_noise": 0})
    std = pd.Series({"x1": std[0], "x2": std[1], "x3": std[2], "white_noise":
    ↪ 1})
    # diagonal matrix with std's on the diagonal
    std_ = pd.DataFrame(np.diag(std), index=std.index, columns=std.index)
    corr = pd.DataFrame({
        "x1": {"x1": 1, "x2": corr12, "x3": corr13, "white_noise": 0},
        "x2": {"x1": corr12, "x2": 1, "x3": corr23, "white_noise": 0},
        "x3": {"x1": corr13, "x2": corr23, "x3": 1, "white_noise": 0},
        "white_noise": {"x1": 0, "x2": 0, "x3": 0, "white_noise": 1}
    })
    cov = std_ @ corr @ std_
```

```

    df = pd.DataFrame(np.random.default_rng(seed=seed).
↳multivariate_normal(mean=mean, cov=cov, size=n),
                        columns=mean.index)
    _X = df[["x1", "x2", "x3"]] if x3 else df[["x1", "x2"]]
    y = b[0] + b[1]*df["x1"] + b[2]*df["x2"] + b[4]*df["white_noise"] +
↳(b[3]*df["x3"] if x3 else 0)
    return _X, sm.add_constant(_X), df["white_noise"], pd.Series(y, name="y")

def cov(X: Union[pd.DataFrame, pd.Series], y: Optional[pd.Series]=None) ->
↳Union[pd.DataFrame, pd.Series]:
    """Return covariance matrix of `X` if `y` is None, else covariance between
↳`X` and `y`."""
    return X.cov() if y is None else pd.concat([X, y], axis="columns").
↳cov()["y"].drop(labels="y")

def ols(X: pd.DataFrame, y: pd.Series, hasconst=True, use_lib=True) -> pd.
↳Series:
    """Get OLS coefficient vector."""
    if not hasconst:
        raise ValueError(hasconst)
    return sm.OLS(exog=X, endog=y, hasconst=hasconst).fit().params if use_lib
↳else \
        pd.Series(inv(X.T @ X) @ (X.T @ y), index=X.columns)

# example
_, X, _, y = gen_data()
pd.DataFrame({"library": ols(X=X, y=y), "us": ols(X=X, y=y, use_lib=False)},
↳columns=["library", "us"])

```

```

[1]:      library      us
const -0.000431 -0.000431
x1      1.000214  1.000214
x2      1.000238  1.000238

```

### 1.3 Result 0.0: Bivariate Loading = Univariate Loading

Suppose  $\rho(\chi_1, \chi_2) = 0$ . Then,  $\beta_i = \sigma^{-2}(\chi_i)\sigma^2(\chi_i, \gamma)$ . This is the familiar formula for a univariate regression slope, otherwise stated as  $\beta_i = \frac{\text{Cov}(\chi_i, \gamma)}{\text{Var}(\chi_i)}$ . Notice how similar this looks to the  $(X^\top X)^{-1}X^\top y$  formula.

Stating the above another way: If the regressors are uncorrelated, then the slope on either regressor in a bivariate regression will be the same as the slope on that regressor in a univariate regression, and it is valid to reduce the bivariate problem to two separate univariate problems.

Pf: Trivial. For example, consider the data-generating process  $\gamma, \chi_1, \varepsilon'$  where  $\gamma = \beta_0 + \beta_2\mathbf{E}(\chi_2) +$

$\beta_1\chi_1 + \varepsilon + \beta_2(\chi_2 - \mathbf{E}(\chi_2))$  which we write as  $\beta'_0 + \beta_1\chi_1 + \varepsilon'$  with  $\sigma^2(\varepsilon') = \sigma^2(\varepsilon) + \beta_2^2\sigma^2(\chi_2)$ . This latter form is amenable to univariate OLS, which by construction must be consistent with our original multivariate formulation.

### 1.3.1 Corollary 0.0.C

Let  $[a_0, a_1] := \text{ols}(y, [1, x_1])$ ,  $[b_0, b_2] := \text{ols}(y, [1, x_2])$ ,  $[c_0, c_1, c_2] := \text{ols}(y, [1, x_1, x_2])$ . We will have  $c_0 = a_0 + b_0 - \bar{y}$ ,  $c_1 = a_1$ , and  $c_2 = b_2$  iff  $r(x_1, x_2) = 0$ .

Pf: This follows from Results 0.0 and 0.1

```
[2]: _, X, _, y = gen_data(mean=(4.13, 2.72, 0), b=(1.62, 3.14, 2.72, 1, 1))
df = pd.DataFrame({"bv1": ols(X=X, y=y),
                  "uv11": ols(X=X[["const", "x1"]], y=y),
                  "uv12": ols(X=X[["const", "x2"]], y=y)})
np.round(df, 1)
```

```
[2]:      bv1  uv11  uv12
const  1.6   9.0  14.6
x1     3.1   3.1   NaN
x2     2.7   NaN   2.7
```

```
[3]: # c0 = a0 + b0 - ybar
round(df.loc["const", "bv1"], 1), \
round(df.loc["const", "uv11":"uv12"].sum() - y.mean(), 1)
```

```
[3]: (1.6, 1.6)
```

## 1.4 Result 0.0.1: Univariate-Loading Formula Generalizes Naturally to Higher Dimensions

$[\beta_1, \beta_2] = \Sigma^{-1}\Sigma_\gamma$ . Thence,  $\beta_0$  can be determined by  $\beta_0 = \mathbf{E}(\gamma) - (\beta_1\mathbf{E}(\chi_1) + \beta_2\mathbf{E}(\chi_2))$ .

Pf: Exercise of doing the OLS matrix calculations “pictorially” (using hand-drawn matrices and symbolic algebra software) for the bivariate case where  $\bar{x}_1 = \mathbf{E}(\chi_1) = 0 = \mathbf{E}(\chi_2) = \bar{x}_2$  with  $N$  observations—i.e. on  $N \times 2$  sampled data—left to the reader. (Tip: I had to go pretty far along in the calculations, the intermediate steps had recognizable sub-pieces, but they were all tangled-up with other extraneous stuff that didn’t cleanly square away until the very end.)

Thence, prove the theoretical result by representing the underlying Bivariate Normal distribution as an  $\infty \times 2$  matrix. Fair disclosure: I have no idea if this is valid, or if it is, under what conditions. But let’s pretend it is.

I also haven’t done it out for the case where  $\mathbf{E}(\chi_i) \neq 0$ , but I’ve seen enough empirical evidence that I’m willing to accept on faith that it works.

And although I also drew it for the trivariate— $\chi_1, \chi_2, \chi_3$ —case, I certainly haven’t done it for the general multivariate case. However, I’m going to use the [arcane pattern](#) that in statistics, things either work nowhere (i.e. in only zero dimensions), in only one dimension, only one or two dimensions, only three-or-more dimensions, or everywhere. I’ve shown that it works in both one, two, *and* three dimensions, hence it must work everywhere.

I'm sure there's a much more elegant way of proving this by visualizing the problem as a vector-space projection or something, but somebody will have to show it to me. Or I can just cheat and link to these excellent StackOverflow answers [here \(archive\)](#) or [here \(archive\)](#).

```
[4]: _X, X, _, y = gen_data(corr12=0.5)
      pd.DataFrame({"library": ols(X=X, y=y), "us": pd.Series(inv(cov(_X)) @ cov(_X,
      ↪y), index=_X.columns)})
```

```
[4]:      library      us
      const  0.000363    NaN
      x1     0.999411  0.999411
      x2     0.999886  0.999886
```

```
[5]: # works even if E(\chi_i) != 0
      _X, X, _, y = gen_data(mean=(3.14, 2.72, 0), corr12=0.5)
      pd.DataFrame({"library": ols(X=X, y=y), "us": pd.Series(inv(cov(_X)) @ cov(_X,
      ↪y), index=_X.columns)})
```

```
[5]:      library      us
      const  0.002521    NaN
      x1     0.999411  0.999411
      x2     0.999886  0.999886
```

```
[6]: # works even if \beta_0 != 0
      _X, X, _, y = gen_data(corr12=0.5, b=(1.62, 1, 1, 1, 1))
      pd.DataFrame({"library": ols(X=X, y=y), "us": pd.Series(inv(cov(_X)) @ cov(_X,
      ↪y), index=_X.columns)})
```

```
[6]:      library      us
      const  1.620363    NaN
      x1     0.999411  0.999411
      x2     0.999886  0.999886
```

```
[7]: _X, X, _, y = gen_data(mean=(3.14, 2.72, 0), std=(42, 24, 1), corr12=.42, b=(1.
      ↪62, 1.337, 7.331, 1, 1))
      pd.DataFrame({"library": ols(X=X, y=y), "us": pd.Series(inv(cov(_X)) @ cov(_X,
      ↪y), index=_X.columns)})
```

```
[7]:      library      us
      const  1.620025    NaN
      x1     1.337008  1.337008
      x2     7.330978  7.330978
```

```
[8]: # craziest combination i could think of
      _X, X, _, y = gen_data(mean=(3.14, 2.72, 4.13), std=(42, 24, 4.2),
                              corr12=.42, corr13=-.42, corr23=0.24,
                              b=(1.62, 1.337, 7.331, 31.4, 1), x3=True)
```

```
pd.DataFrame({"library": ols(X=X, y=y), "us": pd.Series(inv(cov(_X)) @ cov(_X, _
↪y), index=_X.columns))})
```

```
[8]:      library      us
const  1.619596    NaN
x1      1.336999  1.336999
x2      7.330988  7.330988
x3     31.400003 31.400003
```

## 1.5 Result 0.1: BVL != UVL

Suppose  $\rho(\chi_1, \chi_2) \neq 0$ . Then, the conclusion of Result 0.0 will not hold. Notice this essentially turns Result 0.0 into an “if and only if” statement. (Ignore the uninteresting case where e.g.  $\beta_2 = 0$ .)

Pf: By contradiction. Suppose to the contrary that e.g.  $\beta_1 = \sigma^{-2}(\chi_1)\sigma^2(\chi_1, \gamma)$ . Take for granted the results that  $a_1 := s^{-2}(x_1)s^2(x_1, y)$  is an unbiased estimator of  $\sigma^{-2}(\chi_1)\sigma^2(\chi_1, \gamma)$ <sup>1</sup> (which we have supposed is the same as  $\beta_1$ ), and that  $a_1$  interpreted as a univariate OLS slope estimate for  $\beta_1$  has omitted-variable bias of  $\beta_2\sigma^{-2}(\chi_1)\sigma^2(\chi_1, \chi_2)$ . We assume finite and nonzero variance, and in this scenario are supposing that  $\rho(\chi_1, \chi_2) \neq 0 \implies \sigma^2(\chi_1, \chi_2) \neq 0$ . Therefore, the OVB is nonzero and  $a_1$  is a biased estimator of  $\beta_1$ . We therefore conclude that  $a_1$  is simultaneously both a unbiased and a biased estimator of the same value. Hence by contradiction, QED.

## 1.6 Footnote

```
[9]: # even in this simple case, it fails (although thanks to our setup, it is nice
↪and symmetric)
# notice that as expected, the OVB on the UVL's is positive
_, X, _, y = gen_data(corr12=0.5)
df = pd.DataFrame({"library": ols(X=X, y=y),
                  "uvl1": ols(X=X[["const", "x1"]], y=y),
                  "uvl2": ols(X=X[["const", "x2"]], y=y)})
np.round(df, 2)
```

```
[9]:      library  uvl1  uvl2
const      0.0  -0.0   0.0
x1         1.0   1.5   NaN
x2         1.0   NaN   1.5
```

## 1.7 Result 1: Results About the “SumVL”

In the following section, we explore interesting properties of some misspecified regression models. Let  $[a_0, a_1] := \text{ols}(y, [1, x_1])$ ,  $[b_0, b_2] := \text{ols}(y, [1, x_2])$ ,  $[c_0, c_{1+2}] := \text{ols}(y, [1, x_1 + x_2])$ .

<sup>1</sup>TODO(sparshsah): Citation needed.. where did I get this from? Probably need to stare at Gauss-Markov to prove it myself. The denominator and numerator certainly aren’t independent, and even if they were, that would tell us nothing about the expectation of their ratio. Recall that if instead of a ratio, it were a product, and we could have used something like Basu’s Theorem to prove that the estimators were independent, we *could* have used the fact that each individual estimator is unbiased for its estimand, to conclude that the product of the estimators is also unbiased for the product of the estimands.

Caution: When a regression model is misspecified, it will not in general yield coefficient estimates that are good estimates for the ground-truth  $\beta$ ! But the calculation itself just linear algebra, it can of course be done even if it lacks good motivation or interpretation.

Recall also that although in the below examples you know what the ground-truth  $\beta$  is because it's an input to our data sampler, even if you didn't, you could get it by calculating the ground-truth expectation  $\mathbb{E}((X^\top X)^{-1} X^\top y)$ , since the random variable inside the expectation operator is an unbiased estimator for  $\beta$ .

## 1.8 Result 1.0: SumVL = UVL

That is,  $c_0 = a_0 = b_0$  and  $c_{1+2} = a_1 = b_2$ . Obviously, if the individual UVL's differ from each other, this stricter relationship cannot be true. However, supposing the UVL's *do* match, then this will be true iff  $\bar{x}_1 = 0 = \bar{x}_2$  and  $r(x_1, x_2) = 0$ . (I'm ignoring the case where  $a_1 = 0 \iff r(x_1, y) = 0$ , which is uninteresting: Trivially, we can make both  $x_i$ 's totally unrelated to  $y$  and then we will have slopes of zero and intercepts of  $\bar{y}$ .)

Pf: As will become a recurring theme in this section, we'll work our way backward. First of all (or perhaps.. last of all.. heh), we're assuming the slopes match i.e.

$$s^{-2}(x_1)s^2(x_1, y) =: a_1 = b_2 := s^{-2}(x_2)s^2(x_2, y).$$

Let's collapse this and define  $k := s^2(x_2)/s^2(x_1)$  so that we can write

$$s^{-2}(x_1)s^2(x_1, y) = k^{-1}s^{-2}(x_1)s^2(x_2, y)$$

$$s^2(x_1, y) = k^{-1}s^2(x_2, y)$$

$$ks^2(x_1, y) = s^2(x_2, y).$$

Now we can also write

$$c_{1+2} = \frac{s^2(x_1 + x_2, y)}{s^2(x_1 + x_2)} = \frac{s^2(x_1, y) + s^2(x_2, y)}{s^2(x_1) + s^2(x_2) + 2r(x_1, x_2)s(x_1)s(x_2)}$$

Then substitute

$$= \frac{s^2(x_1, y) + ks^2(x_1, y)}{s^2(x_1) + ks^2(x_1) + 2r(x_1, x_2)s(x_1)s(x_2)}$$

Now the numerator is  $(1+k)s^2(x_1, y)$  so the only way for the entire thing to match  $a_1$  is for the denominator to be  $(1+k)s^2(x_1)$  and the only way for that to be possible is for  $\boxed{r(x_1, x_2) = 0}$  (let's ignore the degenerate case where e.g.  $s(x_1) = 0$ ).

On to the intercepts. Let's write

$$\bar{y} - a_1\bar{x}_1 =: a_0 = b_0 := \bar{y} - b_2\bar{x}_2 = \bar{y} - a_1\bar{x}_2 \implies \boxed{\bar{x}_1 = \bar{x}_2}.$$

We want also

$$a_0 = c_0 := \bar{y} - c_{1+2}\bar{x}_1 + \bar{x}_2 = \bar{y} - a_1(\bar{x}_1 + \bar{x}_2) = \bar{y} - a_1\bar{x}_1 - a_1\bar{x}_2 = \bar{y} - a_1\bar{x}_1 - a_1\bar{x}_1 = \bar{y} - 2a_1\bar{x}_1.$$

Uh-oh. This means

$$\bar{y} - a_1\bar{x}_1 = \bar{y} - 2a_1\bar{x}_1..$$

Either  $a_1 = 0$  (the uninteresting case we ignore above) or  $\boxed{\bar{x}_1 = 0}$ . QED.

```
[10]: # simple (almost trivial) case
_, X, _, y = gen_data()
X1 = sm.add_constant(X["x1"])
X2 = sm.add_constant(X["x2"])
X12 = sm.add_constant(pd.Series(X["x1"] + X["x2"], name="x1+x2"))

np.round(pd.DataFrame({"bv1": ols(y=y, X=X),
                        "uv11": ols(y=y, X=X1),
                        "uv12": ols(y=y, X=X2),
                        "uv11+2": ols(y=y, X=X12)},
                        index=["const", "x1", "x2", "x1+x2"])),
1)
```

```
[10]:      bv1  uv11  uv12  uv11+2
const -0.0  -0.0  -0.0    -0.0
x1      1.0   1.0   NaN     NaN
x2      1.0   NaN   1.0     NaN
x1+x2   NaN   NaN   NaN     1.0
```

```
[11]: # more interesting case (variances are unequal, and intercept is nontrivial)
_, X, _, y = gen_data(std=(1, 2, 1), b=(3.14, 2.72, 2.72, 0, 1))
X1 = sm.add_constant(X["x1"])
X2 = sm.add_constant(X["x2"])
X12 = sm.add_constant(pd.Series(X["x1"] + X["x2"], name="x1+x2"))

np.round(pd.DataFrame({"bv1": ols(y=y, X=X),
                        "uv11": ols(y=y, X=X1),
                        "uv12": ols(y=y, X=X2),
                        "uv11+2": ols(y=y, X=X12)},
                        index=["const", "x1", "x2", "x1+x2"])),
1)
```

```
[11]:      bv1  uv11  uv12  uv11+2
const  3.1   3.1   3.1     3.1
x1      2.7   2.7   NaN     NaN
x2      2.7   NaN   2.7     NaN
x1+x2   NaN   NaN   NaN     2.7
```

## 1.9 Result 1.1: SumVL = Average UVL

This is a generalization of Result 1.0, here we want merely  $c_0 = 0.5a_0 + 0.5b_0$  and  $c_{1+2} = 0.5a_1 + 0.5b_2$ . Under what condition will this be true?

```
[10]: # TODO(sparshsah): I got bored of doing these calc's, I'll fill this section in
      ↪ later
```



### 1.10 Result 1.2: SumVL = Sum of UVL's

Under what conditions will we have  $c_0 = a_0 + b_0$  and  $c_{1+2} = a_1 + b_2$ ? Let's start backward:

$$\begin{aligned} a_1 &= \frac{s^2(x_1, y)}{s^2(x_1)} \\ b_2 &= \frac{s^2(x_2, y)}{s^2(x_2)} \\ a_1 + b_2 &= \frac{s^2(x_1, y)}{s^2(x_1)} + \frac{s^2(x_2, y)}{s^2(x_2)} = \frac{s^2(x_2)s^2(x_1, y) + s^2(x_1)s^2(x_2, y)}{s^2(x_1)s^2(x_2)} \\ c_{1+2} &= \frac{s^2(x_1 + x_2, y)}{s^2(x_1 + x_2)} = \frac{s^2(x_1, y) + s^2(x_2, y)}{s^2(x_1) + s^2(x_2) + 2s^2(x_1, x_2)} \end{aligned}$$

Hence we want

$$\frac{s^2(x_2)s^2(x_1, y) + s^2(x_1)s^2(x_2, y)}{s^2(x_1)s^2(x_2)} = \frac{s^2(x_1, y) + s^2(x_2, y)}{s^2(x_1) + s^2(x_2) + 2s^2(x_1, x_2)}$$

Clearly this is underidentified so let's assert that  $s(x_1) = s = s(x_2)$ , whereby we get

$$\frac{s^2(x_1, y) + s^2(x_2, y)}{s^2} = \frac{s^2(x_1, y) + s^2(x_2, y)}{2s^2 + 2s^2(x_1, x_2)}$$

Now the numerators match, so we just need to make the denominators match

$$s^2 = 2s^2 + 2s^2(x_1, x_2) = 2s^2 + 2r(x_1, x_2)s^2 = 2(1 + r(x_1, x_2))s^2$$

$$1 = 2(1 + r(x_1, x_2))$$

$$\boxed{-0.5 = r(x_1, x_2)}.$$

So if their standard deviations are the same, we need  $x_1$  and  $x_2$  to be  $-0.5$  correlated if we want to slopes to add up.

Now what about the intercept? Well, we know

$$a_0 = \bar{y} - a_1\bar{x}_1$$

and

$$b_0 = \bar{y} - b_2\bar{x}_2$$

so that

$$a_0 + b_0 = 2\bar{y} - a_1\bar{x}_1 - b_2\bar{x}_2,$$

and

$$\begin{aligned} c_0 &= \bar{y} - c_{1+2}\bar{x}_1 + \bar{x}_2 \\ &= \bar{y} - (a_1 + b_2)\bar{x}_1 - (a_1 + b_2)\bar{x}_2 \\ &= \bar{y} - a_1\bar{x}_1 - b_2\bar{x}_1 - a_1\bar{x}_2 - b_2\bar{x}_2 \end{aligned}$$

hence we want also

$$2\bar{y} - a_1\bar{x}_1 - b_2\bar{x}_2 = \bar{y} - a_1\bar{x}_1 - b_2\bar{x}_1 - a_1\bar{x}_2 - b_2\bar{x}_2$$

$$\boxed{\bar{y} = -(a_1\bar{x}_2 + b_2\bar{x}_1)}.$$

```
[11]: x1bar, x2bar = 3.14, 4.13
a1, b2 = 1.62, 2.61
intercept = -22.820025177145162 # crystal ball told me this is what is needed

_, X, _, y = gen_data(mean=(x1bar, x2bar, 0), corr12=-0.5, b=(intercept, a1,
↪b2, 1, 1))
X1 = sm.add_constant(X["x1"])
X2 = sm.add_constant(X["x2"])
X12 = sm.add_constant(pd.Series(X["x1"] + X["x2"], name="x1+x2"))

df = pd.DataFrame({"bv1": ols(y=y, X=X),
                    "uv11": ols(y=y, X=X1),
                    "uv12": ols(y=y, X=X2),
                    "uv11+2": ols(y=y, X=X12)},
                    index=["const", "x1", "x2", "x1+x2"])
df
```

```
[11]:          bv1      uv11      uv12      uv11+2
const -22.821736 -7.942412 -14.392452 -22.332507
x1      1.619886  0.314720         NaN         NaN
x2      2.610589         NaN  1.801025         NaN
x1+x2         NaN         NaN         NaN  2.115421
```

```
[12]: # ybar = -(a1 x2bar + b2 x1bar) .. just to prove that my crystal ball was right
round(y.mean(), 2), \
round(-(df.loc["x1", "uv11"] * X["x2"].mean() + df.loc["x2", "uv12"] * X["x1"].
↪mean()), 2)
```

```
[12]: (-6.95, -6.95)
```

```
[13]: # c_{1+2} = a1 + b2
round(df.loc["x1+x2", "uv11+2"], 2), \
round(df.loc["x1", "uv11"] + df.loc["x2", "uv12"], 2)
```

```
[13]: (2.12, 2.12)
```

```
[14]: # c0 = a0 + b0
round(df.loc["const", "uv11+2"], 2), \
round(df.loc["const", "uv11": "uv12"].sum(), 2)
```

```
[14]: (-22.33, -22.33)
```