Assignment 3

Source of this assignment.

This assignment has two purposes:

- To give you more experience with linear regression
- To introduce simulation as a means of studying the behavior of statistical techniques

Please check the due day on Syllabus and Canvas. Submit your `.ipynb` and PDF files to Canvas.

:::{lnote} Notation In this writeup, I will use capital letters (e.g. X) for random variables, and lowercase variables (e.g. x) for individual samples (or *draws*) from those random variables. :::

You will find the probability notes useful for understanding the derivations in this assignment, and the various tutorial notebooks will be helpful for writing the code.

:::{ltip} Reading **Read the assignment very carefully.** There are quite a few moving parts, and you need to carefully follow the instructions in order to complete it successfully.

For many of the tasks I ask several questions. You need to answer these questions in your assignment writeup. :::

:::{ltip} Repeating Several of the tasks have you *repeat a simulation* so that you can see the distribution (mean, variance, perhaps histogram) of a computational result across many repetitions of an experiment. The general structure of this code looks like this:

```
# initialize data structures for storing repitition outcomes
for i in range(ITER_COUNT):
    # draw the synthetic data
    # fit the model / compute the results
    # extract parameters / statistics from model & store in data structures
```

You can use either Python lists or NumPy arrays for storing your results from repetitions. :::

:::{ltip} Structure There are **many pieces** to this assignment. Pay attention to your narrative text and heading structure to make your notebook readable. :::

Simulation

One common way to understand the behavior of statistical techniques is to use *simulation* (often called *Monte Carlo simulation*). In a simulation, we use a psuedorandom number generator to make up data that follows particular patterns (or lack of patterns). We call this data *synthetic data*.

We then apply a statistical technique, such as correlation coefficient or a linear regression, to the synthetic data, and see how closely its results match the parameters we put in to our simulation. If the analysis reliably estimates the simulation's parameters, we say it *recovers* the parameters. We can do this many times to estimate that reliability — we can run the simulation 1000 times, for example, and examine the *distribution* of the error of its parameter estimates to see if it is unbiased, and how broad the errors are.

This technique is commonly used in statistics research (that is, research about statistics itself, rather than research that uses statistics to study other topics) in order to examine the behavior of statistical methods. By simulating samples of different sizes from a population with known parameters, we can compare the results of analyzing those samples with the actual values the statistical method is supposed to estimate. Further, by mapping its behavior over a *range* of scenarios, we can gain insight into what a statistical technique is likely doing with the particular data we have in front of us.

This is distinct from bootstrapping. In bootstrapping, we are resampling our sample to try to estimate the sampling distribution of a statistic with respect to the population our sample was drawn from; we have actual data, but do not know the actual population parameters. In simulation, we know the population parameters, and do not have any actual data because we make it all up with the random number generator.

One further and important point on simulation: when we are simulating a data generating process, we **know** whether or not it satisfies the assumptions the statistical technique we are studying. We know, for example, whether or not our observations are independent, or whether the residuals are i.i.d. normal. We can also **strategically violate** those assumptions to see how the method responds: what does its output look like when its assumptions do not hold? How substantial is the impact of different assumption violation?

Generating Random Numbers

NumPy's {py:class}`Generator <numpy.random.Generator>` class is the starting point for generating random numbers. It has methods for generating numbers from a range of distributions. For more sophisticated distributions, the various distributions in the {py:mod}`scipy.stats` also support random draws.

Random number generators have a *seed* that is the starting point for picking numbers. Two identical generators with the same seed will produce the same sequence of values.

We can create a generator with {py:func}`np.random.default_rng <numpy.random.default_rng>`:

```
rng = np.random.default_rng(20201014)
```

This will return a {py:class}`numpy.random.Generator`, just like {py:func}`numpy.random.default_rng`.

In my class examples, I have been using the current date as my seed. If you do not specify a seed, it will pick a fresh one every time you start the program; for reproducibility, it is advised to pick a seed for any particular analysis. It's also useful to re-run the analysis with a different seed and double-check that none of the conclusions changed.

We can then use the random number generator to generate random numbers from various distributions. It's important to note that *random* does not mean *uniform* — then uniform distribution is just one kind of random distribution.

For example, we can draw 100 samples from the standard normal distribution ($\mu = 0$, $\sigma = 1$) using

{py:meth}`~numpy.random.Generator.standard_normal`:

```
xs = rng.standard_normal(100)
```

This is just a convenient shorthand for the {py:meth}`~numpy.random.Generator.normal` method that allows us to draw samples from a normal distribution with any mean and standard deviation (the method documentation calls these *location* and *scale*):

```
xs = rng.normal(0, 1, 100)
```

:::{admonition} SeedBank :class: note dropdown

If you wish to use SeedBank to manage your RNG seeds, you can do:

```
seedbank.initialize(20221007)
rng = seedbank.numpy_rng()
```

:::

(a4-warmup)=

Warmup: Correlation (10%)

If two variables are independent, their correlation should be zero, right? We can simulate this by drawing two arrays of 100 standard normal variables each, and computing their correlation coefficient:

```
xs = pd.Series(rng.standard_normal(100))
ys = pd.Series(rng.standard_normal(100))
xs.corr(ys)
```

:::{lnote} Code Meaning This code takes *100 draws* (or samples) from the standard normal, twice (once for `xs` and again for `ys`). :::

Mathematically, we write this using $\sim$ as the operator "drawn from":

$$\begin{align*} x & \sim \mathrm{Normal}(0, 1) \ y & \sim \mathrm{Normal}(0, 1) \ \end{align*}$$

{{mtask}} Run 1000 iterations of this simulation to compute 1000 correlation coefficients. What is the mean and variance of these simulated coefficients? Plot their distribution. Are the results what you expect for computing correlations of uncorrelated variables?

:::{tip} What you need to do for this is to run the code example above — that computes the correlation between two 100-item samples — one thousand times. This will draw a total of 200,000 numbers (100 each for `x` and `y`, in each simulation iteration). :::

{{mtask}} Repeat the previous simulation, but using 1000 draws per iteration instead of 100. How does this change the mean and variance of the resulting coefficients?

:::{tip} Now you need to modify the code to draw 1000 normals for `x` and 1000 normals for `y` in each iteration. Remember that the example above is drawing 100 normals for each variable. :::

(a4-covariance)= Remember the **{term}`covariance`** of two variables is defined as:

$$\Cov(X, Y) = \E[(X - \E[X]) (Y - \E[Y])] = \frac{1}{n} \sum_{i} (x_i - \bar{x}) (y_i - \bar{y})$$

And the **{term}`correlation`** is:

$$r = \frac{\Cov(X, Y)}{\sigma_X \sigma_Y} = \frac{\Cov(X, Y)}{\sqrt{\Cov(X, X)} \sqrt{\Cov(Y, Y)}}$$

If we want to generate *correlated variables*, we can do so by combining two random variables to form a third:

$$\begin{align*} x & \sim \mathrm{Normal}(0, 1) \ y & \sim \mathrm{Normal}(0, 1) \ z & = x + y \end{align*}$$

We can draw them with:

```
xs = pd.Series(rng.standard_normal(100))
ys = pd.Series(rng.standard_normal(100))
zs = xs + ys
```

With these variables, we have:

$$\begin{align*} \sigma_X & = 1 \ \E[X] & = 0 \ \sigma_{Y} & = 1 \ \E[Y] & = 0 \ \E[Z] & = \E[X + Y] = \E[X] + \E[Y] = 0 \end{align*}$$

This last identity is from a property called *linearity of expectation*. We can now determine the covariance between X and Z. As a preliminary, since X and Y are independent, their covariance $\Cov(X, Y) = 0$. Further, their independence implies that $\E[X Y] = \E[X]\E[Y]$, which from the equations above is 0.

With that:

$$\begin{align*} \Cov(X, Z) & = \E[(X - \E[X])(Z - \E[Z])] \ & = \E[X (X + Y)] & \text{since $\E[X] = \E[Z] = 0$} \ & = \E[X^2 + XY] \ & = \E[X^2] + \E[X Y] \ & = \E[X^2] \ & = \Var[X] & \text{since $\E[X] = 0$} \ & = \sigma_X^2 = 1 \end{align*}$$

The correlation coefficient depends on $\Cov(X,Z) = 1$, $\Var(X) = 1$, and $\Var(Z)$. We can derive $\Var(Z))$ as follows:

$$\begin{align*} \Var(Z) & = \E[(Z - \E[Z])^2] \ & = \E[Z^2] & \text{since $\E[Z] = 0$} \ & = \E[(X + Y)^2] \ & = \E[X^2 + 2 X Y + Y^2] \ & = \E[X^2] + 2 \E[X Y] + \E[Y^2] & \text{by linearity of expectation} \ & = \Var(X) + \Var(Y) & \text{since both variables' means are 0} \ & = \sigma_X^2 + \sigma_{Y}^2 = 2 \end{align*}$$

Therefore we have $\sigma_X = 1$ (from its distribution), and $\sigma_Z = \sqrt{2}$, and so the correlation

$$r_{XZ} = \frac{\Cov(X,Z)}{\sigma_X \sigma_Z} = \frac{1}{1 \cdot \sqrt{2}} = 0.707$$

:::{admonition} Covariance :class: note

You can compute the covariance with the {py:meth}`pandas.Series.cov` method. It's instructive to also plot that! :::

(a4-correlated)= {{mtask}} Run 1000 iterations simulating these correlated variables to compute 1000 correlation coefficients (`xs.corr(zs)`), with 100 draws of each variable per iteration. Compute the mean and variance of these coefficients, and plot their distributions. Does this match what we expect from the analytic results? What happens when we compute correlations of 1000 draws in each iteration? What about 10000 draws?

Linear Regression (35%)

If we want to simulate a single-variable linear regression:

$$y = \alpha + \beta x + \epsilon$$

there are four things we need to control:

- the distribution of x
- the intercept α
- the slope β
- the variance of errors σ_ϵ^2

Remember that the linear regression model assumes errors are i.i.d. normal, and the OLS model will result in a mean error of 0; thus we have $\epsilon \sim \mathrm{Normal}(0, \sigma_\epsilon)$. Sampling data for this model involves the following steps:

1. Sample x
2. Sample ϵ
3. Compute $y = \alpha + \beta x + \epsilon$

Let's start with a very simple example: x is drawn from a standard normal, $\alpha=0$, $\beta=1$, and $\sigma_\epsilon^2 = 1$.

```
xs = rng.standard_normal(1000)
errs = rng.standard_normal(1000)
ys = 0 + 1 * xs + errs
data = pd.DataFrame({
    'X': xs,
    'Y': ys
})
```

{{mtask}} Fit a linear model to this data, predicting Y with X. What are the intercept and slope? What is R^2? Are these values what you expect? Plot residuals vs. fitted and a Q-Q plot of residuals to check the model assumptions - do they hold?

{{mtask}} Repeat the simulation 1000 times, fitting a linear model each time. Show the mean, variance, and a distribution plot of the intercept, slope, and R^2 from these simulations.

:::{ltip} Extracting Parameters The {py:class}`RegressionResults <statsmodels.regression.linear_model.RegressionResults>` class returned by {py:meth}`OLS.fit() <statsmodels.regression.linear_model.OLS.fit>` contains the model parameters. The `.params` field has the coefficients (including intercept), and `.rsquared` has the R^2 value:

```
fit.params['X']
```

:::

{{mtask}} Fit a model to data with $\alpha=1$ and $\beta=4$. Are the resulting model parameters what you expect? How did R^2 change, and why? Do the linear model assumptions still hold? What are the distributions of the slope, intercept, and R^2 if you do this 1000 times?

Now, let's look at increasing the amount of noise σ_ϵ. We'll do this with:

$$\begin{align*} x & \sim \mathrm{Normal}(0, 1) \ \epsilon & \sim \mathrm{Normal}(0, 5) \ y & = 5 - 2 x + \epsilon \end{align*}$$

{{mtask}} Fit a model to data generated with these parameters. Are the resulting model parameters what you expect? What about the R^2? Do the linear model assumptions still hold?

:::{tip} The residual vs. fitted plot may look circular. Is this a violation? To better understand it, consider running with $x \sim \mathrm{Uniform}(-3, 3)$ — does this change affect any of our linear model assumptions? :::

Nonlinear Data (15%)

{{mtask}} Generate 1000 data points with the following distributions and formula:

$$\begin{align*} x & \sim \mathrm{Normal}(0, 1) \ \epsilon & \sim \mathrm{Normal}(0, 5) \ y & = 10 + 5 e^x + \epsilon \end{align*}$$

{{mtask}} Fit a linear model predicting y with x. How well does the model fit? Do the assumptions seem to hold?

{{mtask}} Draw a scatter plot of x and y.

:::{ltip} Drawing Normals You can draw from $\mathrm{Normal}(0, 5)$ either by using tbe {py:meth} `normal() <numpy.random.Generator.normal>` method of `Generator`, or by drawing an array of standard normals and multiplying it by 5. This is because the normal distribution is in the *scale-location family* of distributions.
:::

:::{ltip} Exponentiation The NumPy function {py:func} `numpy.exp` computes e^x. :::

{{mtask}} Repeat with $y = -2 + 3 x^3 + \epsilon$

Non-Normal Covariates (15%)

{{mtask}} Generate 1000 data points with the model:

$$\begin{align*} y & = -10 + 5 x + \epsilon \ \epsilon & \sim \mathrm{Normal}(0, 30) \ x & \sim \mathrm{Uniform}(0, 100) \end{align*}$$

- Plot the distributions of X and Y
- Fit a linear model predicting y with x
- How well does this model fit? How much of the variance does it explain? Do the assumptions seem to hold? Does the linear regression seem appropriate to the data?

{{mtask}} Generate 1000 data points with the model:

$$\begin{align*} y & = 10 + 2x + \epsilon \ \epsilon & \sim \mathrm{Normal}(0, 1) \ x & \sim \mathrm{Exponential}(5) \end{align*}$$

- Plot the distributions of X and Y
- Fit a linear model predicting y with x
- How well does this model fit? How much of the variance does it explain? Do the assumptions seem to hold? Does the linear regression seem appropriate to the data?

:::{ltip} Exponential Distributions

You can draw 1000 samples from the $\mathrm{Exponential}(5)$ distribution with {py:meth} `~numpy.random.Generator.exponential` method:

```
rng.exponential(5, 1000)
```

:::

Multiple Regression (10%)

Now we're going to look at regression with two or more independent variables.

We will use the following data generating process:

$$\begin{align*} x_1 & \sim \mathrm{Normal}(10, 2) \ x_2 & \sim \mathrm{Normal}(-2, 5) \ \epsilon & \sim \mathrm{Normal}(0, 1) \ y & = 1 + 0.5 x_1 + 3 x_2 + \epsilon \end{align*}$$

:::{ltip} Scale-Location Distribution

To draw from $\mathrm{Normal}(\mu, \sigma)$, you can draw `xs` from a standard normal and compute `xs * σ + μ`. :::

{{mtask}} Fit a linear model `y ~ x1 + x2` on 1000 data points drawn from this model. What are the intercept and coefficients from the model? Are they what you expect? Check the model assumptions — do they hold?

:::{lnote} Multivariate Normals You can draw both x_1 and x_2 simultaneously with:

```
xs = rng.multivariate_normal([10, -2], [[2, 0], [0, 5]], 1000)
# turn into a data frame
xdf = pd.DataFrame(xs, columns=['X1', 'X2'])
```

The multivariate normal distribution is parameterized by a list (or array) of means, and a positive symmetric covariance matrix defined as follows:

$$\begin{bmatrix} \Var(X_1) & \Cov(X_1, X_2) \ \Cov(X_2, X_1) & \Var(X_2) \end{bmatrix}$$

That is, the diagonals of the matrix are the variances of the individual variables, and the other cells are the covariances between pairs of variables. The example code sets up the following matrix:

$$\begin{bmatrix} 2 & 0 \ 0 & 5 \end{bmatrix}$$ :::

Correlated Predictors (10%)

Now we're going to see what happens when we have correlated predictor variables. Remember I said those were a problem?

We're going to use the *multivariate normal* from the hint in the previous part to draw *correlated* variables X_1 and X_2 to use as predictors. We will use the following procedure:

1. Draw 1000 samples of variables X_1 and X_2 from a multivariate normal with means $\langle 1, 3 \rangle$, variances of 1, and a covariance $\Cov(X_1, X_2) = 0.85$:

   ```
   xs = rng.multivariate_normal([1, 3], [[1, 0.85], [0.85, 1]], 1000)
   ```

2. Draw $\epsilon \sim \mathrm{Normal}(0, 2)$

3. Compute $y = 3 + 2 x_1 + 3 x_2 + \epsilon$

{{mtask}} Show a pairplot of our variables X_1, X_2, and Y. What do we see about their distributions and relationships?

{{mtask}} Fit a linear regression for `y ~ x1 + x2`. How well does it fit? Do its assumptions hold?

{{mtask}} Run this simulation (drawing 1000 variables and fitting a linear model) 100 times. Show the mean, variance, and appropriate distribution plots of the estimated intercepts and coefficients (for `x1` and `x2`).

{{mtask}} Repeat the repeated simulation for a variety of different covariances from 0 to 1 (including at least 0, 0.9, 0.99, and 0.999). Create line plots (or a single line plot with multiple colors) that show how the *variance* of the estimated regression parameters (intercept and x_1 and x_2 coefficients) change as you increase the correlation (covariance) between X_1 and X_2.

:::{ltip} Repeated Repeats One iteration of the simulation is the following steps:

1. draw variables
2. compute result (linear model intercept & coefficients)

Repeating that is performing this simulation 100 times, so we can compute the variance of our linear model's estimate.

To repeat the repeated simulation, you need to do *that* multiple times:

```
for real_cov in COV_LIST:
    for rep in range(100):
        # draw numbers
        # fit parameters
```

:::

:::{note} I didn't ask you to include 1 in your selected covariances - what happens if you do? How does that plot differ from a plot that only goes up to 0.999 or 0.9999?
:::

Reflection (5%)

Write a couple of paragraphs about what you learned from this assignment.

Expected Time

I'm providing here some estimates of how long I expect each part might take you.

1. Warmup: 1 hour
2. Linear Regression: 2.5 hours
3. Nonlinear Data: 1 hour
4. Non-normal Covariates: 1 hour
5. Multiple Regression: 1 hour
6. Correlated Predictors: 2 hours
7. Cleanup and Reflection: 1 hour