Exam in Data Analysis with Programming

EBA350003, 24/11/2023

1. The exam contains five questions. All subexercises such as Exercise 1(a), Exercise 2(b), etc., are weighted equally and partial solutions give partial credit. Don't be scared by the length of the document. Its length is mainly due its tables, examples, and hints.
2. You *need* to cite external sources whenever you use them; this includes AI services like ChatGPT. We encourage you to make prodigious use of external sources, and we will not deduct points when you use them.
3. Some exercises will bring you out of your comfort zone. You are expected to find suitable solutions to programming exercises using, e.g., the Numpy documentation. Several hints are provided.
4. You can e-mail (jonas.moss@bi.no) me if you want me to clarify questions, but I will not help you solve them.
5. Be sure to show your code. If the code is massive and not what the exercise asks for directly, place it in an appendix. If the code is small and tidy, put it directly into the document.
6. Make a document that is easy to read. Presentation is important! Be concise, precise, and clear in your writing.

Exercise 1: Timing of `apply_along_axis`

The Numpy function `apply_along_axis`() lets you compute an arbitrary Numpy function over the axis of your choice. For instance, when `x` is a Numpy array, we have

$$\texttt{x.mean}(\texttt{axis} = 1) = \texttt{np.apply\_along\_axis}(\texttt{np.mean}, \texttt{axis} = 1, \texttt{x}).$$

See the Numpy documentation if you need more information. In this exercise we compare the performance of `apply_along_axis`-reliant functions to their Numpy counterparts.

Exercise 1(a) Implementing functions

Implement the following four functions using `apply_along_axis`(), with axis = 1. You need to show your code in this exercise.

1. The variance with `ddof` = `0` (i.e., the biased variance) and `axis` = `1`. Call it `my_var`.
2. The median with `axis` = `1`. Call it `my_median`.
3. The *cumulative sum* with `axis` = `1`. Call it `my_cumsum`.

Solution

```
def my_variance(x):
  return np.apply_along_axis(np.var, 1, x)

def my_median(x):
  return np.apply_along_axis(np.median, 1, x)

def my_cumsum(x):
  return np.apply_along_axis(np.cumsum, 1, x)
```

Exercise 1(b) Find the relevant functions in the Numpy documentation

Find the Numpy equivalents of the functions you just made and list their names. In addition, find another function similar to these, i.e., it returns a decimal number when applied to an array, and accepts the axis argument. Mention its name, and implement it like you did in the previous exercise.

Test the functions in Exercise 1(a) against their Numpy counterparts on the Numpy array with numbers

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 1 \end{bmatrix}.$$

Solution

```
def numpy_variance(x):
  return x.var(axis = 1, ddof=0)

def numpy_median(x):
  return np.median(x, axis = 1)

def numpy_cumsum(x):
  return x.cumsum(axis = 1)
```

Exercise 1(c) Comparing speed

It would be reasonable to expect the built-in Python functions to outperform our custom `apply_along_axis`-based functions, but by how much? Your estimates will vary widely, since your computers are different, hence one of you should run the study by yourself. Please provide details about your CPU if you know how to find it.

I did a timing experiment using `tidybench` with `x = rng.normal(size = (100, 100))`. I defined functions `my_var`, `numpy_var`, and so forth, which were plugged into the `bench` function. Then I copied the data into Table 1, where the ratio column displays the ratio of the custom implementation to the Numpy implementation. I ran the code on an I7-12700 with 20 cores (but only one is used by Python, unfortunately) and 2100mHz base clock speed.

Make a similar table for your computer, rounded to two decimal values, but with the functions you chose in the previous exercise in the last row. Comment on the findings. Do the timings change if you use a different `size` argument?

	Numpy (millisec)	Custom (millisec)	Ratio
Variance	0.02	0.81	40.5
Median	0.09	1.05	9.62
Cumulative sum	0.02	0.19	11.65

Table 1: Timings for the six different functions with a ratio column for ease of reading.

Solution

You could solve the exercise by doing, e.g.:

```
bench = benchmark(['numpy_variance(x)',
'my_variance(x)', 'numpy_median(x)',
'my_median(x)', 'numpy_cumsum(x)',
'my_cumsum(x)'], ntimes = 100, warmup = 10, g = globals())

result = bench.means
np.array([result["my_variance"] / result["numpy_variance"],
result["my_cumsum"] / result["numpy_cumsum"],
result["my_median"] / result["numpy_median"]]).round(2)

np.array(bench.means.values()).round(2) * 1000
```

The timings change a lot with varying size arguments.

Exercise 2: Maximum likelihood estimation

We will study the performance of the maximum likelihood estimator for θ in the uniform distribution $[0, \theta]$. In this exercise we let

$$X_i \sim \text{Uniform}(0, \theta), \quad i = 1, \ldots, n$$

be identically and independently distributed.

Exercise 2(a) Derive the maximum likelihood estimator

Show that the maximum likelihood estimator of θ is $\hat{\theta}_{ML} = \max_{1 \leq i \leq n} X_i$. You may refer to a reliable source here.

Solution

The textbook is a reliable source.

Exercise 2(b) Twice the mean

Show that $E\overline{X} = \theta/2$, where $\overline{X}$ is the sample mean of $X_1, X_2, \ldots, X_n$. Then calculate its variance, and use the Chebyshev inequality (or another, perhaps simpler, argument) to argue that the estimator $\hat{\theta}_E = 2\overline{X}$ is consistent for θ. Is $\hat{\theta}_E$ approximately normal when n is large enough?

Solution

Simple calculations; for the variance, use that $\text{Var}(X) = \theta/12$. First observe that $E\overline{X} = \int_0^\theta x dx = \frac{1}{2}\theta$. The variance is $\theta/12$ (see, e.g., wikipedia, or integrate yourself.) Since the variance is finite, the central limit theorem shows that $\sqrt{n}(\hat{\theta}_E - \theta) \to N(0, \theta/12)$ as $n \to \infty$.

Exercise 2(c) Twice the mean

- Make a function `sim_uniform(n, n_nreps, theta, rng)` that simulates random samples of size n a total of `n_reps` times using the random number generator object `rng` (e.g., `rng = np.random.default_seed(seed = 1)`). It should return an array of n columns and `n_reps` rows.
- Then make a function `est_uniform(x)` that calculates both the maximum likelihood estimator and the mean-based estimator for a data frame of n columns and `n_reps` rows. It should return the simulated estimates in two columns, yielding an array of two columns and `n_reps` rows. You may need to *stack* your output here.
- Combine these two functions into one `sim_est_uniform(n, n_nreps, theta, rng)`, with results like:

```
>>> rng = np.random.default_rng(seed=1)
>>> sim_est_uniform(2, 4, 3, rng)
array([[4.38685596, 2.85139109],
       [3.27842718, 2.84594834],
       [2.2054737 , 1.26997935],
       [3.71070519, 2.48310778]])
```

Solution

```
def sim_uniform(n,n_reps,theta, rng):
  return rng.uniform(0,theta, (n_reps, n))

def est_uniform(x):
    return np.stack((2 * x.mean(axis = 1), x.max(axis = 1), x.max(axis = 1) * (x.shape[1] + 1) /

def sim_est_uniform(n, n_reps, theta, rng):
  x = sim_uniform(n, n_reps, theta, rng)
  return est_uniform(x)
```

Exercise 2(d) Use the simulation function

Use the simulation function for $n = 10, 100, 1000$ and $\theta = 3$, with $10,000$ repetitions for each choice of n. Fill out Table 2 with the correlation between the estimators, the means of the estimators and their ratio of mean squared errors; round to two decimal places. It might be beneficial to create a new function for this task, and it might be helpful to use `np.hstack`, and to recall the bias-variance decomposition.

Which estimator do you prefer, and why?

	Corr$(\hat{\theta}_E, \hat{\theta}_{ML})$	Mean $(\hat{\theta}_E)$	Mean $(\hat{\theta}_{ML})$	MSE $(\hat{\theta}_E)$/MSE $(\hat{\theta}_{ML})$
10	0.5	3.00	2.72	2.23
100				
1000				

Table 2: Characteristics of the estimators of θ in the uniform distribution.

Solution

The MSE ratio becomes really large as n increases, and we prefer the ML estimator. The following code can be used for this and the next exercise.

```
def sim_est_uniform_results(n, n_reps, theta, rng):
  x = sim_uniform(n, n_reps, theta, rng)
  results = est_uniform(x)
  np.corrcoef(xx.T)[1, 0]
  means = results.mean(axis = 0)
  variances = results.var(axis = 0)
  corrs = np.array(np.corrcoef(results.T)[1, 0])
  mses = (means - theta)**2 + variances
  mse_ratio = np.array(mses[0]/mses[1])
  mse_ratio2 = np.array(mses[2]/mses[1])
  return np.hstack((corrs, means, n*variances, mse_ratio, mse_ratio2))
```

Exercise 2(e) The modified maximum likelihood estimator

One can show that the expectation of the maximum likelihood estimator $\hat{\theta}_{ML}$ is $\frac{n}{n+1}\theta$. Use this to construct an unbiased estimator of θ based on $\hat{\theta}_{ML}$. Would you prefer this estimator to the standard estimator? You should provide arguments based on the mean squared error, potentially based on the code you wrote for the previous exercise.

Solution

A small modification to the program in the previous question can calculate the MSE of the unbiased $\frac{n+1}{n}\hat{\theta}_{ML}$; but it is also possible to calculate the MSE by hand. The MSE of the unbiased estimator is twice the MSE of the biased one.

Exercise 3: Understanding linear regression

Exercise 3(a) The scope of simple linear regression

Consider the following plots. For which, if any, of these is simple linear regression suitable, and why? If simple linear regression is not suitable, suggest alternative methods of analysis. You don't need to provide a lot of details here.

Solution

1. Model 1: Binary regression.
2. Model 2: Linear regression.

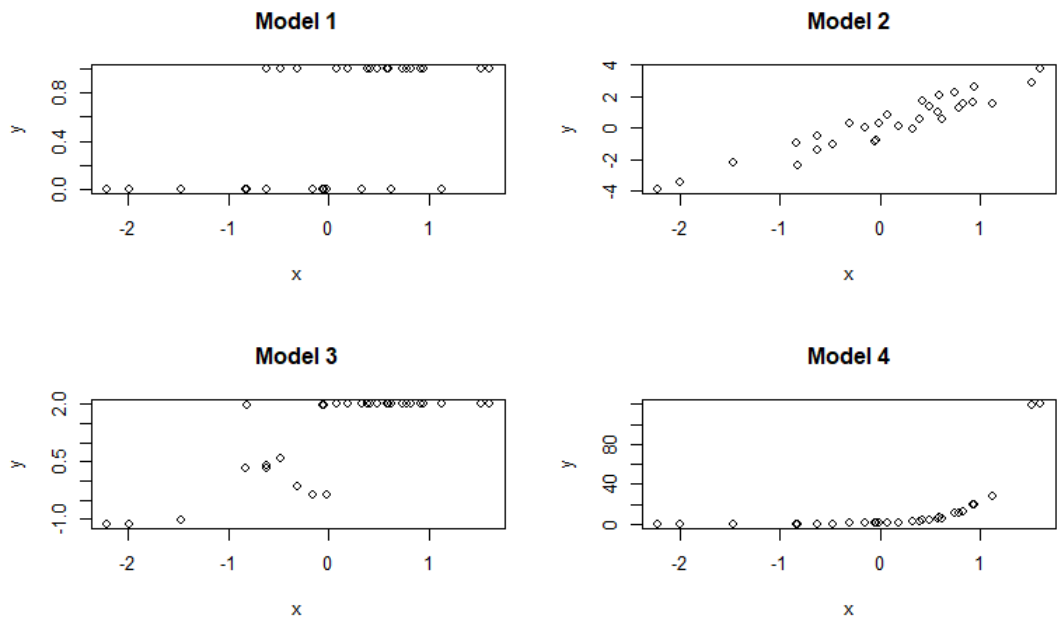

Figure 1: Is linear regression a good idea?

3. Model 3: Split in two at approximately 0, run linear to the left and mean to the right. Other solutions possible.

4. Model 4: This looks exponential, so we need a log-transform.

Exercise 3(b) The coefficients of linear regression

We are dealing with the following truncated regression output, obtained from the bikershare data in the next exercise. The response is the numbers of bikers in a day, the covariates are month, humidity (hum) and normalized temperature (temp). Both hum and temp are bounded between 0 and 1.

```
==============================================================
                       coef      P>|t|       [0.025      0.975]
--------------------------------------------------------------
mnth[April]         92.0652      0.000      60.728     123.403
mnth[Aug]          126.7210      0.000      89.896     163.546
mnth[April]:hum   -193.5908      0.000    -232.736    -154.446
mnth[Aug]:hum     -308.8857      0.000    -354.141    -263.630
temp               359.3156      0.000     333.061     385.570
==============================================================
```

In addition, we have the following data about mean temperatures and humidities in August and April.

```
>>> wage[["mnth", "temp", "hum"]].groupby("mnth").mean()
           temp       hum
```

```
mnth
April  0.471015  0.668220
Aug    0.705554  0.625622
```

Answer the following questions. Provide your reasoning.

1. What is the predicted number of bikers for a day in April when the humidity is 1/2 and the normalized temperature 1/2?
2. Are there values of humidity and temperature where the predicted number of bikers for a day in April is larger than the expected number of bikers in August?
3. Suppose you don't know the value of humidity and temperature on a particular day. Construct a reasonable prediction of bikers on an April day and an August day. Is the predicted number largest for April or August?

Solution

1. $92 - 0.5 \cdot 193 + 0.5 \cdot 359$.
2. Yes, for instance $hum = 1$.
3. Use the mean values for each month. The expected number is largest for April.

Exercise 4: Regression with bikershare

We use study the bikeshare data from Introduction to Statistical Learning, available on the website. Document ion for the data can be found here.

Our goal is to understand which factors affect the the number of bikers.

Exercise 4(a) Describing the data

1. Load the data. Remove the first column, which contains no information. Also remove the casual and registered columns.
2. How many rows and columns are there in the data?
3. Which variables are qualitative and which are quantitative?
4. Are there any missing values in the data?
5. Describe the relationship between `temp` and `atemp`.

Solution

(2-3) Use the info function. (4) No. (5) They are deterministic functions of each other; described in the documentation.

Exercise 4(b) Inspect the correlation between bikers and the variables

Take a look at the univariate correlation between bikers and all the other variables. Comment on the correlations, especially the high ones. Are any of them surprising?

Solution

Many possible solutions, but at the very least it's strange that "days" have a linear relationship with the number of bikers.

Exercise 4(c) Fitting a regression model

Fit a regression model with bikers as response using all covariates in the data frame.

1. Which coefficients are not significant?
2. Explain what it means that a coefficient isn't significant.

Solution

1. Use the summary or pvalues method.
2. You can reject the null-hypothesis that the regression coefficient of a covariate is 0.

Exercise 4(d) Bikers and weekday

Make scatter plots of `bikers` vs `day`. Comment on what you see. Fit a simple linear regression model `bikers` $\sim$ `day`. Then find a dramatic improvement on this model, using transformations. State the model's formula and explain why it is better. Finally, give a plausible explanation for why the data looks like this.

Solution

The relationship between bikers and day is roughly quadratic, with adjusted R^2s of 0.032 for the linear model and 0.117 for the quadratic model.

Exercise 4(d) Looking at months

Most, but not all, of the months in the whole model were statistically significant. It would be instructive to see how the number of bikers vary with respect to month. Use, e.g., the sns.catplot function to visualize how the number of bikers vary with months.

Now make a new data frame with months replaced with their respective number. E.g., January becomes 1, February 2, and so on, until December becomes 12. You can do this with the Pandas method replace, provided you have a dictionary with months as keys and "month numbers" as values. Call this covariate `mnth_num` and add it to the data frame. Fit a quadratic regression model with `bikers` as response and `mnth_num` as covariate. Compare the fit to the regression model `bikers` $\sim$ `mnth`. Explain what you see, and connect it to the distinction between inference / explanation and prediction discussed in Introduction to Statistical Learning.

Solution Using a quadratic model works remarkably well, with and R^2 of 0.11 compared 0.12 for the categorical variables. Quadratic functions are easy to understand, so that a quadratic model works so well gives you insight into what the data says.

Exercise 4(e) Choosing a model without transformations

Fit four different models, but without using transformations. You may use either `mnth_num` or `mnth`, but not both. If you use `mnth_num`, you may use the quadratic transform on that covariate, but that covariate only. Which one do you choose as your final model? Be explicit about your reasoning here. Make a table containing the adjusted R^2s and AICs for each model. Comment on what you see.

Solution

Skipped.

Exercise 4(f) Choosing a model with transformations

As explored in a previous exercise, the relationship between `bikers` and `day` is not linear. Now try to fit at least five models using transformations and, possibly, interaction terms (e.g., `day` $*$ `temp`). Do you find anything interesting in your investigations?

Solution

This exercise is open-ended: Virtually any set of models will do, except those chosen to be bad on purpose (e.g., only the covariates with the smallest correlation.

Exercise 5: Confidence intervals

Let $X_i \sim f_\theta, i = 1, \ldots, n$ be n iid observations from a statistical model with density f_θ parameterized by θ. For instance, f_θ could be exponential, $f_\theta(x) = \frac{1}{\theta} e^{-\frac{1}{\theta}x}$, or f_θ could be normal, with $\theta = f_{(\mu,\sigma)}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.

The goal of this exercise is to investigate how well the t-interval peforms for different choices of models, samples sizes, and parameters. To do this, we need the notion of *coverage*. Suppose you have a fixed density f_θ with known population mean μ, and denote the t-interval for the population mean μ with confidence level 95% by $[L, U]$. The *coverage* of the confidence interval at f_θ is

$$P_{f_\theta}(L \leq \mu \leq U),$$

where P_{f_θ} means the probability is calculated with respect to f_θ. The *nominal* coverage of the interval is 95%, but the actual coverage will not be equal to 95% for all models.

We will study confidence intervals for the normal, exponential, logistic, and log-normal models. You can find random number generators for all of these in the Numpy documentation.

Exercise 5(a) Normality

What is the nominal coverage of the 95% t-interval when the data is normal $N(\mu, \sigma)$? Why?

Solution

We know from the book that the interval is exact under normality.

Density	Parameter 1	Parameter 2	Mean
Normal	loc	scale	loc
Exponential	scale		scale
Logistic	loc	scale	
Log-normal	mean	sigma	

Table 3: Table of densities, parameters, and means.

Exercise 5(b) Interpreting confidence intervals

Your friend Peter observes a 95% t-interval $[-0.5, 0.2]$ and proclaims that the probability that the true mean is between -0.5 and 0.2 is 95%. Is Peter correct? Explain.

Solution

This claim is false. He is 95% confident.

Exercise 5(c) Find the means of the models

To calculate the coverage of our models we need their population means. Using the parameterization found in the Numpy documentation, fill out the following table of means. You may use any source you like, including wikipedia, or calculate the values yourself.

Solution

For the log normal, the mean is $e^{\mu+\frac{1}{2}\sigma^2}$. Logistic has mean loc.

Exercise 5(d) Calculate t-interval

Let `dist` be a function of one argument that returns random samples from a distribution, e.g., `dist` = `lambdasize : rng.normal(1, 2, size)`. Make a function `sim_ci(n, n_reps, dist)` that simulates an array of dimension (`n_nreps`, 2), where each row contains the t-interval calculated from n independent samples from `dist`.

On my machine, I got the following, but there *may* be differences due different default number generators in varying Python versions. For reference, I use Python 3.10.7 on Windows.

```
sim_ci(10, 3, dist)
array([[-1.47579403,  1.26424767],
       [ 0.05607252,  2.57385862],
       [ 0.76130578,  2.02706914]])
```

Solution

```
def sim_ci(n, n_reps, dist):
  ci = st.ttest_1samp(dist((n, n_reps)), 0).confidence_interval(0.95)
  return np.array(ci).T
```

Exercise 5(e) Calculating coverage

Make a function `sim_cov(n, n_reps, dist, mean)` that returns the simulated coverage of the t-interval from n_reps repetitions of n independent samples from dist. This is done in two steps.

Distribution	n	Cov	Distribution	n	Cov
Normal(1,3)	10	0.951	Exponental(3)	10	
	100			100	
Normal(0,0.1)	10		Exponental(0.3)	10	
	100			100	
Logistic(3,3)	10		Log-normal(0,1)	10	
	100			100	
Logistic(0,0.1)	10		Log-normal(2,2)	10	
	100			100	

Table 4: Table of coverages.

First, map each row in $\texttt{x} = \texttt{sim\_ci}(\texttt{n}, \texttt{n\_reps}, \texttt{dist})$ to 1 if the interval contains `mean` and 0 otherwise. Then take mean of these values.

Testing the function, I got the following.

```
>>> dist = lambda size: np.random.default_rng(seed=1).normal(1, 2, size)
>>> sim_cov(10, 10000, dist, 1)
0.951
```

Solution

```
def sim_cov(n, n_reps, dist, mean):
  ci = sim_ci(n, n_reps, dist)
  return ((ci[:, 0] <= mean) & (mean <= ci[:, 1])).mean()
```

Exercise 5(f) Simulating multiple confidence intervals at once

Fill out the following table with coverages using $\texttt{n\_reps} = 10,000$ and comment on the results. You should use the the function defined in the previous exercise and the means you found in exercise 5.2. If you weren't able to find the formulas for the exact means, you may simulate them instead. (*Hint*: You probably want to use lists here. One list of distributions and one list of means; the zip function may also be helpful.)

Solution

One can use the following code and fill in the values.

```
dists = [
    lambda size: np.random.default_rng(seed=1).normal(1, 3, size),
    lambda size: np.random.default_rng(seed=1).normal(0, 0.01, size),
    lambda size: np.random.default_rng(seed=1).logistic(3, 3, size),
    lambda size: np.random.default_rng(seed=1).logistic(0, 0.01, size),
    lambda size: np.random.default_rng(seed=1).exponential(3, size),
    lambda size: np.random.default_rng(seed=1).exponential(0.3, size),
    lambda size: np.random.default_rng(seed=1).lognormal(0, 1, size),
    lambda size: np.random.default_rng(seed=1).lognormal(2, 2, size),
  ]

means = [1, 0, 3, 0, 3, 0.3, np.exp(0 + 1/2), np.exp(2 + 2)]
[sim_cov(10, 10000, dist, mean) for dist, mean in zip(dists, means)]
```

```
[sim_cov(100, 10000, dist, mean) for dist, mean in zip(dists, means)]
```

The confidence intervals are not affected by the parameters for logistic, exponential, and normal, and the coverage is close to 95% for all of them, and also for all n. The coverage for the log-normal is bad, especially for the parameters $(2, 2)$.