EEF pilot study 1 on statistics with computing

This is a report of a first full pilot course from the EEF grant, for teaching statistics through computing from July 18th through the 20th.

Summary

This was the third iteration of this course, where the first two are:

- First pilot (non-EEF).
- Pre-pilot (EEF).

As for the previous courses, I was teaching basic Python programming, including conditionals, loops and functions, in order to solve problems of probability and inference, through simulation and resampling. We covered basic simulation with random numbers, and builds up to a full permutation test.

For this iteration, I spent more time on the basic programming. I also rewrote the exam to give more scaffolding for the first question.

This time the course felt a little less rushed, although I sprinted through the permutation test at the end.

I was more effective in covering `for` loops this time, with the benefit of some reflection after my failure in the previous course.

The exam marks were better, which partly reflects my rewrite of the exam.

Feedback scores hinted that the students were more confident about doing data analysis, than for the previous iteration.

I can think of two directions to go for the next iteration:

- Drop permutation testing, continue to concentrate on basic programming.
- Switch to using arrays for the simulations, to avoid for loops and conditionals, then proceed to permutation.

The next step is to discuss this with the steering group.

Background

Please see the background in the report on the first (not EEF) pilot course at the course website.

See also the report from the pre-pilot course.

To prepare for this course, I scanned some novice textbooks on programming, and particularly How to Think Like a Computer Scientist.

This course differed from the pre-pilot course in:

- Recasting the introductory talk as motivation for data science, and learning to code, rather than concentrating on reproducibility.
- Early introduction of "three girl" problem.
- Slightly slower pace.
- More time devoted to basic programming constructs - in particular - the `for` loop.
- Homeworks were standard coding exercises in Python.
- Extended range of exercises, starting at easy and ramping up to exercises needing a fair amount of independent discovery.
- More scaffolding for the first question in the exam.

Methods

See the course web page and sources at github repository.

The course consisted of:

- 12 hours of face-to-face teaching (4 hours on each of 3 days).
- 2 hours of homework.

First day

- Introduction talk, covering: the nature of data science; data science and reproducibility; thoughts on learning how to program.
- Introduction to the Hansard post-Brexit survey.
- Brief trot through Brexit survey analysis with Pandas, live coding, see resulting notebook.
- Introduction to simulation with the "three girls" problem. "If a family has 4 children, what is the probability that the family has 3 girls?". Simulation with coin tosses. Analysis of the algorithm.
- Brief discussion of debugging.
- Introduction to variable assignment.
- Introductions to Strings, functions, types of things.
- Introduction to Lists.

- Comparisons, True and False.
- If and For.

Homework was a series of introductory exercises from the standard LearnPython site.

Second day

- Revision of If and For.
- and, or in expressions.
- writing functions.
- Solving the three girls problem with Python code.

Homework was another series of introductory Python exercises from the CodingBat site.

Third day

- Revision of three girls problem.
- Introduction to plotting.
- The sum function.
- Exercise to solve the Brexit proportions problem.
- Introducing the shuffle function.
- Adding lists.
- A permutation test on the Brexit survey ages.

Exam

The exam was three notebooks, as for the pre-pilot course.

The only difference was that the `rainfall.ipynb` notebook had a lot more scaffolding, starting with easier questions, and building up to the same question as the pre-pilot versions.

- `rainfall.ipynb` - a version of the classic programming problem (Seppälä et al. 2015). The student gets a series of numbers, and has to take the mean of the numbers preceding the first instance of `99`. Build up questions asked for the sum of all the numbers, a count of the instances of `99`, and the sum of all the numbers except instances of `99`. The exercise tested iteration, lists and conditional statements.
- `widgets.ipynb` - using simulation to estimate weights and variability of bag of widgets, of different weights. This tested functions generating outcomes with given probabilities, and assembling samples from this function to get and test sampling distributions.

- `sprinting.ipynb` - implementing a permutation test similar to the one we had done in class, with a new dataset.

Results

See results notebook.

The exam scores (out of 100) were higher than the pre-pilot, and with greater spread., Min, median and max scores were 10, 27, 65, and the mean was 30.4. 4 of 9 students made very little headway.

The question "How much computer code have you written?" was related to the exam scores. The options (mean scores) were:

- "I've never written any code" (17.7);
- "I've played with code but never wrote anything useful" (28.5);
- "I have written a small amount of useful code" (34.0);
- "I have written a lot of useful code" (62.0).

The only student who had taken GCSE computer science scored near the mean of the group, suggesting that the GCSE wasn't very useful as an introduction to programming.

Motivation for taking the course may have been important. The only student admitting to doing the course entirely for the bursary, rather than learning about data analysis, was one of 4 students who got near-baseline scores on the exam.

Feedback was somewhat more positive than previous versions of the course. There was 4 / 5 average agreement with "I had a better understanding of statistics after the course", where the previous course score was equivalent to 3.9 / 5. This time there was 4.1 / 5 agreement with "After the course, I feel more confident about doing my own data analysis." compared to the previous 3.1 / 5. See the permutation test in the result for some evidence this might be a real difference. Ratings for the pace of the course were similar to previous versions, at an average of 2.5 / 5, with 1 being "Too fast".

Impressions

It was probably sensible to reduce the emphasis on Pandas analysis of the Brexit data. In the previous course I had the impression the students were wondering if they would have to reproduce this analysis, and therefore, trying hard to understand it, as I typed. Here I think the students accepted that this was advanced stuff that they did not need for this course.

I did not check whether the students did the homework, but my impression was that they did, and it appeared to help, both in allowing the students to practice the basics, and convincing them that I was teaching standard programming.

As the students noted in their feedback, the course was still a little rushed. It was hard to get to the permutation test, and I did not give them enough time to understand the code they needed.

The exam scores were better for this run, and this may have been because I gave more scaffolding for the first question, allowing some fairly easy marks. Nevertheless, 4 / 9 of the students were close to baseline.

I had the impression that even the students who did not do well in the exam did find themselves engaging with programming. To quote a comment from a lower-scoring student: "I would have never thought that coding would be something that would interest me but after this course I am planning to continue to learn how to code for data science and programming. It was intense which is good because we learnt as much as we could have in the space of 12 hours".

The students asked for printed notes to refer back to. I wasn't expecting them to do that, as all their course materials were available on the website, but it will be interesting to see if I can make some printed materials they will find useful.

Plan

It now seems clear that it is not practical to teach all of the machinery for the permutation test using a frontal assault like this. There are two options:

- Continue to iterate over this frontal assault approach, but drop the permutation testing from the end of the course. The students still learn about simulation, sampling distributions, the null hypothesis, and the meaning of the p value, but they do not reach the stage where they can do something like a t-test using this machinery.
- The other option is to try and flank the problem, by using another method of teaching simulation, in order to speed up the exposition. The Berkeley Foundations of Data Science course takes an interesting approach, which is go straight to arrays as the standard data structure for a sequence of values, rather than lists, as is more typical. Doing this would allow me to do the simulation work by making two-dimensional arrays of random numbers, with one row per trial and one column per element in the trial. For example, the "three girls" problem could be solved with code like this:

```
import numpy as np

values = np.random.uniform(size=(10000, 4))
```

```
girls = values < 0.5
girl_counts = girls.sum(axis=1)
proportion = np.sum(girl_counts == 3) / 10000
```

The brexit problem becomes:

```
# 1315 respondents expressed a preference
values = np.random.uniform(size=(10000, 1315))
# True proportion of Leave voters is 0.519
leavers = values < 0.519
leaver_counts = leavers.sum(axis=1)
# Here is the sampling distribution
leaver_proportions = leaver_counts / 1315
# Proportion of Leavers in the survey was 541 / 774 == 0.41
# So this is the p value of our observed result
p_value = np.sum(leaver_proportions <= 0.41) / 10000
```

This might indeed be easier to teach, but there are two problems:

- They will have a weaker foundation in programming, so they may be less able to take up new tasks that need programming.
- The permutation test still needs loops, slicing, and indexing, so there's still a hill to climb. On the other hand, we probably don't need to cover functions.

Here's the code for the permutatation test:

```
import numpy as np

import pandas as pd

voters = pd.read_csv('remain_leave.csv')
remainers = voters.loc[voters['brexit'] == 1]
leavers = voters.loc[voters['brexit'] == 2]

ages = np.concatenate((leavers['age'], remainers['age']))

n_leavers = len(leavers)

observed = ages[:n_leavers].mean() - ages[n_leavers:].mean()

n_iterations = 10000

sampling_distribution = np.zeros(n_iterations)
for i in range(n_iterations):
    # Shuffle up the ages of the Leavers and Remainers
    np.random.shuffle(ages)
    # Calculate the mean age difference for the new shuffled groups
    difference = ages[:n_leavers].mean() - ages[n_leavers:].mean()
    # Put them into the sampling distribution
    sampling_distribution[i] = difference

p_value = sum(sampling_distribution >= observed) / n_iterations
```

I need to discuss these options with the steering committee.

References

Seppälä, Otto, Petri Ihantola, Essi Isohanni, Juha Sorva, and Arto Vihavainen. 2015. "Do We Know How Difficult the Rainfall Problem Is?" In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, 87–96. ACM.