Assignment 1: Basic data analysis and simulating probability distributions

In this assignment, you will first analyze some real estate data, and then simulate some random processes corresponding to common statistical distributions and models.

Work in groups of one, two or three and solve the tasks described below. Write a short report containing your answers, including the plots and create a zip file containing the report and your Python code. Alternatively, write a Jupyter notebook including your code, plots, and comments. Submit your solution through this page (click "Submit Assignment" in the top right).

Didactic purpose of this assignment:

- practice some basic analysis of different types of data, using statistical libraries in Python;
- get a gut feeling for the scenarios underlying some of the most common models used in statistics and data science;
- get some experience in generating synthetic data by simulating in (simplified) models.

References

- In Lecture 1, we saw how to plot histograms and compute basic descriptive statistics.
  - Matplotlib reference documentationLänkar till en externa sida..
  - Pandas reference documentationLänkar till en externa sida..
- For random number generation, see:
  - NumPy random documentationLänkar till en externa sida..

Part 1: Real estate prices (3 pts)

Here is a CSV (comma-separated values) file listing real estate sales in England between 1995 and 2016. (Actually, to make things a bit faster it's only a subset.)

- Load the CSV file into Python. Use the Pandas function read_csv or use one of the techniques you learned in the course *Introduction to Data Science*.
- The second column in the CSV file represents the price of the property. Compute basic descriptive statistics about the prices in the whole dataset: mean, median, standard deviation, minimum, and maximum.
- Plot a histogram that shows the distribution of the prices. **Hint**: why is it so ugly? What can you do to make it more informative?
- Is real estate more expensive in London? Plot histograms for the two subsets of properties inside and outside London, respectively. For practical purposes, we can define "inside London" to mean that the string in the 14th column (Python indexing column 13) includes the string LONDON.
- **Optional task**. Make a plot that shows the average price per year.

Part 2: Histogram and quantile (4 pts)

In this part of the assignment, you will compute histograms and quantiles to illustrate your understanding. You will work with the Titanic data set (download here).

Description

- Wikipedia page Länkar till en externa sida.and the movieLänkar till en externa sida.Länkar till en externa sida.
- There are 14 columns in the table:

1.
    1. Passenger age (Age)
    2. Cabin number (Cabin)
    3. Port of embarkation (Embarked)
    4. Fare (Fare)
    5. Passenger name (Name)
    6. Parents/children aboard (Parch)
    7. Passenger ID (PassengerId)
    8. Passenger class (PClass)
    9. Passenger sex (Sex)
    10. Siblings/spouses aboard (SibSp)
    11. Survival indicator (Survived)
    12. Ticket number (Ticket)
    13. Passenger title, i.e. Mrs., Mr., etc. (Title)
    14. Family size (Family_Size)

- You will look at these four columns in this exercise: **Embarked**, **PClass**, **Parch**, **Fare**

(a) Identify the data types of these four columns and plot their distributions

(b) Histogram

Compute the histogram for columns **Parch** and **Fare** and report the results (the values of both x and y axes). Show that the histograms you have computed return the same value as np.histogramLänkar till en externa sida.. Note: for continuous data, choose **bins=20**.

When you compute the histograms:

- **cannot** use:
    - Any of the built-in hist functions, e.g. np.histogramLänkar till en externa sida.
    - np.digitizeLänkar till en externa sida. (you need to do the quantization yourself)
- can use:
    -
    - collections.CounterLänkar till en externa sida.

(c) Data quantile

Compute the data quantiles of **Fare** for **p = 0.2** and **p=0.5.** You cannot use any built-in quantile function to compute the quantiles. After you compute the quantiles, use the assert function to make sure that your values are the same as the values produced by np.quantileLänkar till en externa sida..

Part 3: Generating data from probabilistic models (3 pts)

Here we simulate a few probabilistic models to investigate certain situations and the corresponding random variable distributions. When you are done feel free to change the various parameters to see how they affect the results.

Please note. When you have implemented the code for these three scenarios, please reflect about how well you think the models correspond to the real world. What are the simplifying assumptions? Please discuss in your report.

(a) Consider the random number generation functions in NumPy, documented hereLänkar till en externa sida..

- Generate a set of random numbers using the function rand and plot its histogram. What is the shape of this histogram and why?
- Investigate how the shape of the histogram is affected by the number of random numbers you have generated.
- Instead of using rand (which corresponds to a *uniform* distribution), generate numbers using some other distribution and plot a histogram. What is the shape now? For instance, with normal, the normal (or Gaussian) distribution, you should get the familiar bell shape

(b) Modeling a student at an exam

Let's make a model of a student that answers questions in an exam. The exam consists of a fixed set of questions, and a student answers each question correctly or incorrectly with some fixed probability. We will now implement this model in a step-by-step fashion.

Answering a single question

Write a Python function that simulates that the student answers a single question either correctly or incorrectly. The function should return a Boolean value (that is, True or False) that says whether the question was answered correctly. You can assume that the probability of a correct answer is a given parameter p_success.

```
def success(p_success):

    ... YOUR CODE HERE ...
```

Run this function a few times and check that it seems to work correctly.

Note: Formally, we say that this function simulates a random variable with a *Bernoulli* distribution. The metaphor typically used is that of a coin toss with an unfair coin.

How many correctly answered questions?

Next, we make another function called exam_score that simulates a scenario where the student answers a fixed set of questions. We assume that all questions are equally difficult. As inputs, your function needs the number of questions, as well as the probability of a correct answer. The function should return the number of correctly answered questions. To implement this, it seems natural to use the function success that you developed previously.

```
def exam_score(p_correct, n_instances):

    ... YOUR CODE HERE ...
```

Again, run the function a few times and check that it seems to work as it should.

Investigating the distribution

Write some code to call exam_score several times, and collect the result of all the calls in a simple Python list, NumPy array, or Pandas Series.

Let the value of p_correct be 0.8 and n_instances be 20. Run exam_score 10,000 times and collect the results. Then plot a histogram of the results.

Note: This type of scenario corresponds to the *binomial* distribution. The typical explanation is that we toss an unfair coin a given number times and count the number of times the heads side came up.

(c) The persistent student

We will now simulate a scenario where a student takes an exam repeatedly, until passing.

If a student does not pass an exam, the University of Gothenburg allows the student to go to an unlimited number of re-sit exams. Let's assume that students never give up, so that they will go to the exam again and again until they finally pass. Write a function that simulates a student going to exams until passing, and returns the number of attempts the student needed before passing. You can assume that the probability of passing a single exam is a constant p_pass. If you want, you can reuse your function success from the previous task: in this case, this would mean a passed exam, not just a correctly answered question.

```
def number_of_attempts(p_pass):

    ... YOUR CODE HERE ...
```

Investigating the distribution

Simulate this model multiple times, as in (a). For instance, let p_pass be 0.4. Plot the result using a histogram.

Note: This type of scenario corresponds to the *geometric* distribution.