Lecture 4
Introduction to Python for Data Science

Q1. See L4_LAB_Questions.py section Q1

Q2. COIN FLIP QN [use L4_LAB_Questions.py section Q2 as template for answers]

Diversification Coin Toss Game Example.

Consider the following coin-toss game:

- Win £2 if you get heads
- Lose £1 if you get tails

The expected payoff is

$$0.5 * £2 + 0.5 * (-£1) = £0.5$$

The standard deviation of the payoff is

$$[0.5 * (£2 - £0.5)^2 + 0.5 * (-£1 - £0.5)^2]^{1/2} = £1.5$$

What if we split the payoff over two coin tosses?

- Win £1 every time you get heads
- Lose £0.5 every time you get tails

The expected payoff is the same

$$0.25*(£1+£1)+2*0.25*(£1-£0.5)+0.25*(-£0.5-£0.5) = £0.5+£0.25-£0.25 = £0.5$$

The standard deviation of the payoff is

$$[0.25 * (£2 - £0.5)^2 + 0.5 * (0.5 - £0.5)^2 + 0.25 * (-£1 - £0.5)^2]^{1/2} = £1.06$$

The standard deviation/risk has been reduced while keeping the expected payoff the same: that's the power of **diversification**!

In fact, in this example, as we divide the bet over more and more coin tosses, risk decreases quite rapidly and in the limit (infinite number of bets) we can even eliminate it completely.

Your ultimate task for this question is to provide a plot for n = 1-10 coin tosses plotted against the standard deviation for the payoff of such a game.

a) Set up the game by using Python as a calculator to write out game structure. This will convert the current set up into code. Here you should simply define variables such as:

   `n` (number of coin tosses)
   `p` (probability of H = T for a single realisation of n coin tosses e.g. n = 2 tosses p = 0.5 * 0.5, for n = 3 tosses p = 0.5 * 0.5 * 0.5)

`H` (Heads amount won)
`T` (Tail amount lost)
Then using the above variables and the above explanation reproduce the payoff and standard deviation results for n = 1 and n = 2.

Report:
```
print("Payoff:", payoff, "StdPayoff:", '%.2f' % std_PO)
```

b) We have worked with number of tosses n=1, n=2, lets expand. Provide a `for` loop solution to the expected payoff and its std dev, which produce results for a game with a generalised number of tosses, n (use less than 10) e.g. `n = 3`.

Note: here you will need to create 2 more variables:

`s = (H, T)` tuple which specifies a set of all possible outcomes of the experiment of tossing a coin: Head or Tails.

```
from itertools import product
outcomes = list(product(s, repeat = n))
```
combination of all possible outcomes of `n` coin tosses.

```
print("PayoffN:", '%.2f' % payoffN, "StdPayoff:", '%.2f' % std_PO_N)
```

c) Convert the above `for` loop solution to a `numpy/pandas` solution, i.e. by just using libraries methods. This code should run just for a chosen `n`.

d) Extend solution from c) such that you create a function which allows you to test any `n` (recommended to go up to 10 only since converting `product` to outcomes by typecasting it to a `list` is computationally expensive).

Run this code such that you obtain a std dev for each `n=1,…,10` and use Matplotlib to display the plot of `n` vs std dev of the expected payoff.

What can you conclude regarding the relationship between increasing number of tosses into which the game is split and the standard deviation of the expected payoff?

Q3. KNN Question [use L4_LAB_Questions.py section Q3 as template for answers]
K Nearest Neighbours

We are going to work with K=3 multiclass classification problem using Iris dataset.

a) Load the libraries and run provided lines of code line by line examining output.
b) Check the first 5 observations in the `data` (this should be a 5x4 array)
c) Examine `target` variable contents (are observations randomly arranged?)

Data pre-processing and visualising.

d) Decide whether data needs standardizing / normalising?
e) Do observations in the data need random shuffling? If which function can perform this job?
f) Split data into 75-25% training/testing, using `random_state=1234` (reproducibility).

g) Observe the pair plot using `seaborn` visualisation library with the provided code.
h) Create an instance of the estimator class `KNeighborsClassifier` to fit a `K=3` neighbours optional parameter and call it `knn`.
i) Fit the training data to the `knn` model.
j) Predict classes on unseen `X_test` data and report accuracy.
k) You model now is used in the real world and a botanist sent you the following data on a new previously unseen Iris flower: `x_new = np.array([[5, 2.9, 1, 0.2]])`. The botanist is uncertain as to the class of this Iris and is asking you to make a prediction. What class does your model predict this new iris flower to belong to?
l) How do you know if this answer is correct?

[EXTRA] **Own KNN Code.**

Perform KNN classification by writing your won code, rather than relying on using the `sklearn` library.

m) Create 2 nested for loops, *outer loop* to loop over the number of points in the test set, and *inner loop* to loop over the number of points in the train set.

n) For each test data point (i.e. row of test data), will loop over the inner loop and obtain the Euclidian distance from the test point to all the training data points.

o) After the inner loop is finished, sort the distance vector (and corresponding training labels) in ascending order. Obtain first K shortest distances. Based on the count of classes to which the points corresponding to shortest distances belong to, make the prediction for that test data point.

p) After the outer for loop is also finished, calculate the accuracy of your algorithm. It should be exactly the same as that obtained from the `sklearn KNeighborsClassifier` function.