assignment_04_solution

March 17, 2023

Please fill in your name and that of your teammate.

You:

Teammate:

1 Introduction

Welcome to the fourth lab. This week the math load is lower on purpose to give you a chance to catch up on the first 3 labs before we start to *use* all the foundations you learned so far. You may want to go back and fix your submissions with the help of the solutions, you will need the material for the exam.

This week we introduce the major data analysis library in Python: `pandas`. You can think of it as providing **feature-rich data containers**: load your data in a `pandas` object and you will have fast access, manipulation, statistics, even math with `numpy` and plotting with `matplotlib`, all well integrated.
Unfortunately it also has a fame of being frustrating, unintuitive and stubborn for the newcomers, so I suggest you spend some time practicing and download the documentation as offline pdf for the exam.

The two main classes are `Series`, for one-dimensional data, and `DataFrame` for *tensors*. You may have heard this word before: you can think of a tensor as a generic structure for feature-based data. A zero-dimensional tensor is a scalar; a 1D tensor is a vector; a 2D tensor is a matrix; you can imagine a 3D tensor as a cube, or as a list of matrices. Higher dimensions are of course hypercubes. You will start mostly using `DataFrame`s with 2D tables/matrices, but some methods will return (kind-of) higher-dimensional tensors (e.g. be careful with `groupby()`! Takes time to grasp).

This library is best learned hands-on, but before starting it is important to understand how the data is stored and accessed.

1.1 The feature's perspective

Let's look again at our dataset of snakes. It has three fields: `head size`, `length` (in cm) and whether it is `poisonous` or not. It looks like this:

```
snakes = [['small', 38, False],
          ['small', 62, True],
          ['medium', 55, True]]
```

This form puts the emphasis on the data points (the rows), which is an intuitive approach at first for humans to manually write down the data. We can scroll to the middle of a dataset in a CSV

or text file and read one line – in principle, if the columns are few and we remember their order. Accessing the data by indices can also be confusing: if you want to know whether the second snake is poisonous you should use `snakes[1][2]`, which is prone to the type of misunderstandings that lead to bugs (e.g. writing instead `[2][1]`).
From the machine perspective this is not an issue, but this way of storing has the distinct (performance) disadvantage of having **multiple types** in the same data structure: here we have lists with strings, floats and booleans.

An alternative form to represent the data which is also very common (you already encountered it e.g. for data plotting) gives instead priority to the columns or *features*. It contains the same data, but rows and columns are *transposed*:

```
snakes = [['small', 'small', 'medium'],
          [38, 62, 55],
          [False, True, True]]
```

As you can see each feature is not in its own list, and all types are the same, meaning we could store it in a specialized array casted to the correct data type, and enjoy a big performance boost. Moreover, we can now switch to a hash map (a `dict` in Python), allowing us to include the feature name and use them for indexing, making the data and its description self-contained:

```
import numpy as np
snakes = {'head_size' : np.array(['small', 'small', 'medium'], dtype='str'),
          'length'    : np.array([38, 62, 55], dtype='float'),
          'poisonous' : np.array([False, True, True], dtype='bool')}
```

This has the added advantage that you can now call the vast library of `numpy` methods on your features, such as: `snakes['length'].mean()`. Create a new cell below, copy+paste the code and give it a try!

Data access readability is also improved by using explicit feature names: `snakes['poisonous'][1]` does not leave room for misunderstandings, it is the `poisonous` field of the first data point. Writing `snakes[1]['poisonous']` will now raise an error (remember: errors are your friends when debugging, what is dangerous is a silent bug).
One last feature we could add is to add an explicit extra feature called `index` with incremental numbers for the data points, which allows us to explicitly sort the data or access by index as we did before:

```
import numpy as np
snakes = {'index'     : np.array([1, 2, 3], dtype='int'),
          'head_size' : np.array(['small', 'small', 'medium'], dtype='str'),
          'length'    : np.array([38, 62, 55], dtype='float'),
          'poisonous' : np.array([False, True, True], dtype='bool')}
```

Isn't that neat? Congratulations, you just derived a naïve implementation of a Pandas `DataFrame`. Get used to this name because you will be using it all the time over the next weeks.

1.2 Pandas objects

Let's actually convert our data to a `DataFrame` so we can play with it:

```
import pandas as pd
df = pd.DataFrame(snakes) # the constructor accept most sensible inputs
df.head() # this prints the first few lines of your (potentially large) dataframe
```

A few tips before we start: - DataFrames are composed of `Series` to hold the features: Series wrap the Numpy array to extend with more methods, so anything you can do with a Numpy array can be done with most Pandas structures, and much more. - Data always has an `index` (actually of class `Index`, a special Series), and if you took a database course you should know why (hint: primary key). You can provide an index explicitly, or it will automatically generate one on creation from a counter. - Accessing data with `[]` (e.g. `df['length']`) does not give direct pointer access at the stored data, but returns a copy-on-write handler. This means that creation is free, read access is fast, but trying to write or edit the data first generates a *copy* of the data, and then edits that. The original is unchanged by `[]`. The full truth is actually even more complex, e.g. try creating a new column `df['new_col'] = 5`, this requires no copy) - To access the data directly by column name and index, you should use `df.loc[]` (by index and column) and `df.iloc[]` (by indices). Yes they are both methods but they take square parenthesis (ah, python…). Data access is even more confusing than this, and you will need some serious practice to reliably get Pandas to do what you want. Do not skip on this! I suggest you follow this tutorial. - Sometimes you need to access ranges of values. Ranges in Python are also weird: keep in mind that you can use the explicit `range()` method, or more commonly the `:` format: `start:end:step`. Funny part, all three are optional: `:k` means from the first to the k^{th} element (excluded: the `end` is never included in a Python range, careful); `p:` means from element number `p` until the end; `::2` means (from beginning to end but only) every second element; and most importantly `:` means all elements. This is necessary as `iloc` always requires rows as first argument, so if you want the third column (implicitly: *values of all rows* for the third column) you need to write `df.iloc[:,2]` (remember indices start from 0). Again, complex to grasp, but super easy and obvious once you get it: practice it! - You can access the data by "conditions", by generating a boolean array that will hard-select the rows or columns you want. Yeah even more confusing :) bear with me. Try writing `df['length']<60`: this will return a *boolean numpy array*, with values `[True, False, True]`. This is the answer to the boolean question you asked. Now to get the lines of the dataframe that answered True, you need to write `df[df['length']<60]`. Read it and try it until it makes sense before moving forward. Also do yourself a favor and check also the second part of the tutorial above (boolean indexing): all the time you invest now in Pandas will pay dividends later in the harder assignments. - You can plot the dataframe data directly using the function `df.plot()` (takes `type=` as a parameter), which is sometimes useful for a quick peek. In almost every practical case though it is easier (and produces better results) to simply use `seaborn`, because it integrates DataFrames natively! Finally a good news! try `sns.barplot(data=df, y='length', x=df.index, hue='poisonous')` - Most DataFrame methods also return the result of a computation as a "copy" or "view" on the Pandas object. If you want your method calls to have persistent consequences, which means you want to modify the original data, you should either (i) capture the output in a variable and use that, or (ii) use the optional argument `inplace=True` which is available for most methods and forces modifying the original data.

1.2.1 How to pass the lab?

Below you find the exercise questions. Each question awarding points is numbered and states the number of points like this: **[0pt]**. To answer a question, fill the cell below with your answer (markdown for text, code for implementation). Incorrect or incomplete answers are in principle worth 0 points: to assign partial reward is only up to teacher discretion. Over-complete answers do not award extra points (though they are appreciated and will be kept under consideration). Save your work frequently! (`ctrl+s`)

You need at least 18 points (out of 27 available) to pass (66%).

2 1. Fundamentals

By now you should be familiar with this. We start easy:

1.1 [2pt] Write a small DataFrame with at least 3 of the common problems you can find when dealing with unknown data. If you need inspiration you can reuse and expand your cats and dogs dataset from lab. Loading into a dataframe should be straightforward: here is the documentation, pass the table as `data=` and the labels as `columns=` to the constructor.

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
sns.set(rc={'figure.figsize':(8,6)}, style="whitegrid")

feature_names = ['likes_dogs', 'agility', 'height']
data = [[True, 8, 25],
        [True, 4, 50],
#         [False, 7, 30],
        [False, 7, None], # missing value
        [False, 9, 20],
        [True, 2, 65],
#         [False, 8, 30],
        ["Maybe", 8, 30], # wrong type
        [True, 6, 25],
#         [True, 8, 40],
        [True, 999, 40], # clipping
        [True, 9, 45],
        [False, 6, 35]]
labels = ['cat', 'dog', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog', 'cat']

df = pd.DataFrame(data=data, columns=feature_names)
df['labels'] = labels
df
```

[1]:
```
  likes_dogs  agility  height labels
0       True        8    25.0    cat
1       True        4    50.0    dog
2      False        7     NaN    cat
3      False        9    20.0    cat
4       True        2    65.0    dog
```

```
5       Maybe        8     30.0     cat
6        True        6     25.0     cat
7        True      999     40.0     dog
8        True        9     45.0     dog
9       False        6     35.0     cat
```

1.2 [1pt] Explain with your own word why data defects mitigation is important and why it cannot ever be perfect. Here "own word" means that copying and pasting from the slides will actually fail the question. Feel free to discuss around the concepts if the words come hard, just prove that you got the meaning and that you can talk about it. Write as long as you need to show your skills, but no longer (brevity is also a skill that takes practice).

Mitigation is important because the tolerance to data defect of any ML algorithm is limited, and the algorithm performance depends on the data quality. On the other hand perfect reconstruction of the data is impossible without, well, having the perfect data in the first place. If we had a way of generating perfect data, that would make a great model, and we would not need ML here. Incidentally, if the data we have does not cover the process we want to model, we need to go back to the source and collect new data on purpose: we are using the actual process to generate data of "higher quality" (for our purpose), which we could not do from the data alone.

1.3 [1pt] Give two reasons why to drop lines with defects rather than fixing them.

(i) Faster; (ii) Often we have enough data anyway because of redundancy so it would be a wasted effort.

1.4 [1pt] Why does data quality impact learning more than data quantity? Mention one of the algorithms seen so far as an example. Because basic supervised learning model learn from all data points equally, without bias. There is no way for the algorithms we saw so far to distinguish outliers and decide to "learn less" from them. For example linear regression will return a wildly skewed model if the data has an outlier due to clipping, severely lowering its performance.

1.5 [1pt] Convert (by hand!) a categorical feature from your hand-written dataset into a numerical one using binary encoding. This means that you need to rewrite the hand-written dataset, but a previously categorical feature needs to be written in binary encoding.
If your dataset did not have a categorical feature yet, go ahead and simply add one.
IMPORTANT: you may want to go ahead and fix the defects that you introduced earlier, or you may encounter problems.

[2]:
```python
feature_names = ['likes_dogs', 'agility', 'height']
data = [[1, 8, 25],
        [1, 4, 50],
        [0, 7, 30],
        [0, 9, 20],
        [1, 2, 65],
        [0, 8, 30],
        [1, 6, 25],
        [1, 8, 40],
        [1, 9, 45],
        [0, 6, 35]]
labels = ['cat', 'dog', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog', 'cat']

df = pd.DataFrame(data=data, columns=feature_names)
df['labels'] = labels
df
```

[2]:
```
   likes_dogs  agility  height labels
0           1        8      25    cat
1           1        4      50    dog
2           0        7      30    cat
3           0        9      20    cat
4           1        2      65    dog
5           0        8      30    cat
6           1        6      25    cat
7           1        8      40    dog
8           1        9      45    dog
9           0        6      35    cat
```

1.6 [1pt] Consider 5-fold cross-validation: how many models does it train? Now take the first fold: how many of the models that were trained did in fact use this particular fold as part of their training set? 5-fold cross-validation will train 5 models. For any given fold, 4 of those models will use it for training, while the one that didn't will use it for testing.

3 2. Loading data

To simplify the process, here's a CSV string rather than a separate file. You are free to copy this string into a file if you like, the call below simply emulates an Input Output object (such a file) from a String using a `StringIO` object. It is often a useful trick to have on your tool belt.

IMPORTANT: do not edit the string! The defects and problems are on purpose! It simulates problematic data that was passed to you unclean, potentially too big to read for a human, and you need to use only Pandas to load and clean the data.

[3]:
```
from io import StringIO
```

```
csv_str =␣
 ↪StringIO('sepal_length$sepal_width$petal_length$petal_width$species$is_flower\n0$5.
 ↪1$-3.5$1.4$0.2$setosa$True\n1$-3.0$1.4$setosa$True\n2$5.0$-3.6$1.4$0.
 ↪2$setosa$True\n3$4.6$-3.1$1.5$0.2$setosa$I guess\n4$4.7$-3.2$1.3$0.
 ↪2$zetosa$True\n5$5.4$-3.9$1.7$0.4$setosa$True\n6$4.6$-3.4$1.4$0.
 ↪3$setosa$True\n7$5.0$-3.4$1.5$0.2$setosa$Fals\n8$4.4$-2.9$1.4$0.
 ↪2$setosa$True\n9$4.9$-3.1$1.5$0.1$setosa$True\n10$5.4$-3.7$1.5$0.2$setosa$I␣
 ↪guess\n11$4.8$-3.4$1.6$0.2$setosa$True\n12$4.8$-3.0$1.4$0.
 ↪1$setosa$True\n13$4.3$-3.0$1.1$0.1$setosa$True\n14$5.8$-4.0$1.2$0.
 ↪2$setosa$True\n15$-4.4$1.5$setosa$True\n16$5.4$-3.9$1.3$0.
 ↪4$setosa$True\n17$5.1$-3.5$1.4$0.3$setosa$True\n18$5.7$-3.8$1.7$0.
 ↪3$setosa$True\n19$5.1$-3.8$1.5$0.3$setosa$True\n20$5.4$-3.4$1.7$0.
 ↪2$setosa$True\n21$5.1$-3.7$1.5$0.4$setosa$True\n22$4.6$-3.6$1.0$0.
 ↪2$setosa$True\n23$5.1$-3.3$1.7$0.5$setosa$Fals\n24$4.8$-3.4$1.9$0.
 ↪2$setosa$True\n25$5.0$-3.0$1.6$0.2$setosa$True\n26$5.0$-3.4$1.6$0.
 ↪4$setosa$True\n27$5.2$-3.5$1.5$0.2$setosa$True\n28$5.2$-3.4$1.4$0.
 ↪2$setosa$True\n29$4.7$-3.2$1.6$0.2$setosa$True\n30$4.8$-3.1$1.6$0.
 ↪2$setosa$True\n31$5.4$-3.4$1.5$0.4$setosa$True\n32$5.2$-4.1$1.5$0.
 ↪1$setosa$Fals\n33$5.5$-4.2$1.4$0.2$setosa$True\n34$4.9$-3.1$1.5$0.
 ↪2$setosa$True\n35$5.0$-3.2$1.2$0.2$setosa$True\n36$5.5$-3.5$1.3$0.
 ↪2$setosa$True\n37$4.9$-3.6$1.4$0.1$setosa$True\n38$4.4$-3.0$1.3$0.
 ↪2$setosa$True\n39$5.1$-3.4$1.5$0.2$setosa$True\n40$5.0$-3.5$1.3$0.
 ↪3$setosa$True\n41$4.5$-2.3$1.3$0.3$setosa$True\n42$4.4$-3.2$1.3$0.
 ↪2$setosa$True\n43$5.0$-3.5$1.6$0.6$setosa$True\n44$5.1$-3.8$1.9$0.
```

```
print('Data loaded')
```

```
Data loaded
```

2.1 [2pt] Load the data despite the errors.

- Be patient
- Peruse the documentation
- If Python raises an error, read carefully the message at the end of the output cell
- You should not need a `try... except`: make Pandas get the format until there are no errors
- You will need to skip the lines with errors that impede loading (it will still generate warnings, but that is fine for now)
- To print the DataFrame, a simple `df` often looks better in Jupyter notebooks than an explicit `print(df)`

```
[4]: df = pd.read_csv(csv_str, sep='$', on_bad_lines='skip') # try using 'warn'␣
     ↪instead to see where the problems are!
     df
```

```
[4]:      sepal_length  sepal_width petal_length petal_width    species is_flower
     0             5.1         -3.5          1.4         0.2     setosa      True
     1            -3.0          1.4       setosa        True        NaN       NaN
     2             5.0         -3.6          1.4         0.2     setosa      True
     3             4.6         -3.1          1.5         0.2     setosa   I guess
     4             4.7         -3.2          1.3         0.2     zetosa      True
     ..            ...          ...          ...         ...        ...       ...
     145           6.7         -3.0          5.2         2.3  virginica      True
     146           6.3         -2.5          5.0         1.9  virginica      True
     147           6.5         -3.0          5.2         2.0  virginica      True
     148           6.2         -3.4          5.4         2.3  virginica      True
     149           5.9         -3.0          5.1         1.8  virginica      True

     [148 rows x 6 columns]
```

2.2 [1pt] Explore the DataFrame, and try to spot the defects. Mitigate two by hand.

You may want to learn about the method `head()`. Also calling `unique()` on a Series will give you the set of different values available. Remember the goal of the question is to fix only two errors for now.

```
[5]: df.drop(1, inplace=True)
     df.loc[4, 'species'] = 'setosa'
     df
```

```
[5]:      sepal_length  sepal_width petal_length petal_width    species is_flower
     0             5.1         -3.5          1.4         0.2     setosa      True
     2             5.0         -3.6          1.4         0.2     setosa      True
     3             4.6         -3.1          1.5         0.2     setosa   I guess
     4             4.7         -3.2          1.3         0.2     setosa      True
```

```
5            5.4        -3.9         1.7        0.4     setosa     True
..           …          …        …          …       …       …
145          6.7        -3.0         5.2        2.3  virginica     True
146          6.3        -2.5         5.0        1.9  virginica     True
147          6.5        -3.0         5.2        2.0  virginica     True
148          6.2        -3.4         5.4        2.3  virginica     True
149          5.9        -3.0         5.1        1.8  virginica     True

[147 rows x 6 columns]
```

4 3. Cleaning the data

The following questions require you to either write a Python function or some automated Python code: manually selecting (like last question) for example rows with missing values is not accepted here, you need to provide code that will work with any (unknown) dataset and rows.
NOTE: no two students ever followed the same process in this exercises, but for those who copy from last year's solutions. As per University policy on plagiarism, students have been expelled from their course of study for something as cheap as an ungraded assignment. Unbelievable. Please not again.

3.1 [1pt] Remove all rows with missing values. This can help (and geeksforgeeks.org is a great resource to keep in mind). Also consider customizing the Pandas display options, for example: `pd.set_option("display.max_rows", 200)` to see more than just a few lines (see more options in the method's documentation).

[6]:
```
df.dropna(inplace=True)
df['is_flower'].unique()
```

[6]:
```
array(['True', 'I guess', 'Fals'], dtype=object)
```

[7]:
```
# For the next question, here are the correct types for the columns:
corr_types = {'sepal_length' : 'float',
              'sepal_width' : 'float',
              'petal_length' : 'float',
              'petal_width' : 'float',
              'species' : 'string',
              'is_flower' : 'bool'
             }
```

3.2 [1pt] Check the column types. Find the reason for the columns having wrong type, and write it. Mitigate the defects and cast the column to the correct type.

[8]:
```
print(df.dtypes)
df = df.astype(corr_types)
print()
print(df.dtypes)
```

```
sepal_length    float64
sepal_width     float64
petal_length     object
petal_width      object
species          object
is_flower        object
dtype: object

sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          string
is_flower          bool
dtype: object
```

3.3 [1pt] Remove all columns with constant values. I used `loc` and a boolean array with `nunique`.

[9]:
```
df = df.loc[:, df.nunique()>1]
df
```

[9]:
```
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1         -3.5           1.4          0.2     setosa
2             5.0         -3.6           1.4          0.2     setosa
3             4.6         -3.1           1.5          0.2     setosa
4             4.7         -3.2           1.3          0.2     setosa
5             5.4         -3.9           1.7          0.4     setosa
..            ...          ...           ...          ...        ...
145           6.7         -3.0           5.2          2.3  virginica
146           6.3         -2.5           5.0          1.9  virginica
147           6.5         -3.0           5.2          2.0  virginica
148           6.2         -3.4           5.4          2.3  virginica
149           5.9         -3.0           5.1          1.8  virginica

[144 rows x 5 columns]
```

3.4 [3pt] There are still errors in the data: fix them all. This time, I suggest you do not just "answer" this question in order. It is easier to move to the next questions, see what errors you get, come back and write code that fixes them (remember: data analysis is an iterative process). If you can answer all questions then there should be no errors left. Also keep in mind that Python 3 strings handle UTF-8 characters if you have not noticed already

[10]:
```
df.loc[:,'sepal_width'] = df.sepal_width.abs()
df.loc[df['species'] == ' ', 'species'] = 'versicolor'
df
```

[10]:
```
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
2             5.0          3.6           1.4          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             4.7          3.2           1.3          0.2     setosa
5             5.4          3.9           1.7          0.4     setosa
..            …            …             …            …          …
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[144 rows x 5 columns]
```

5 4. Visualizing the data

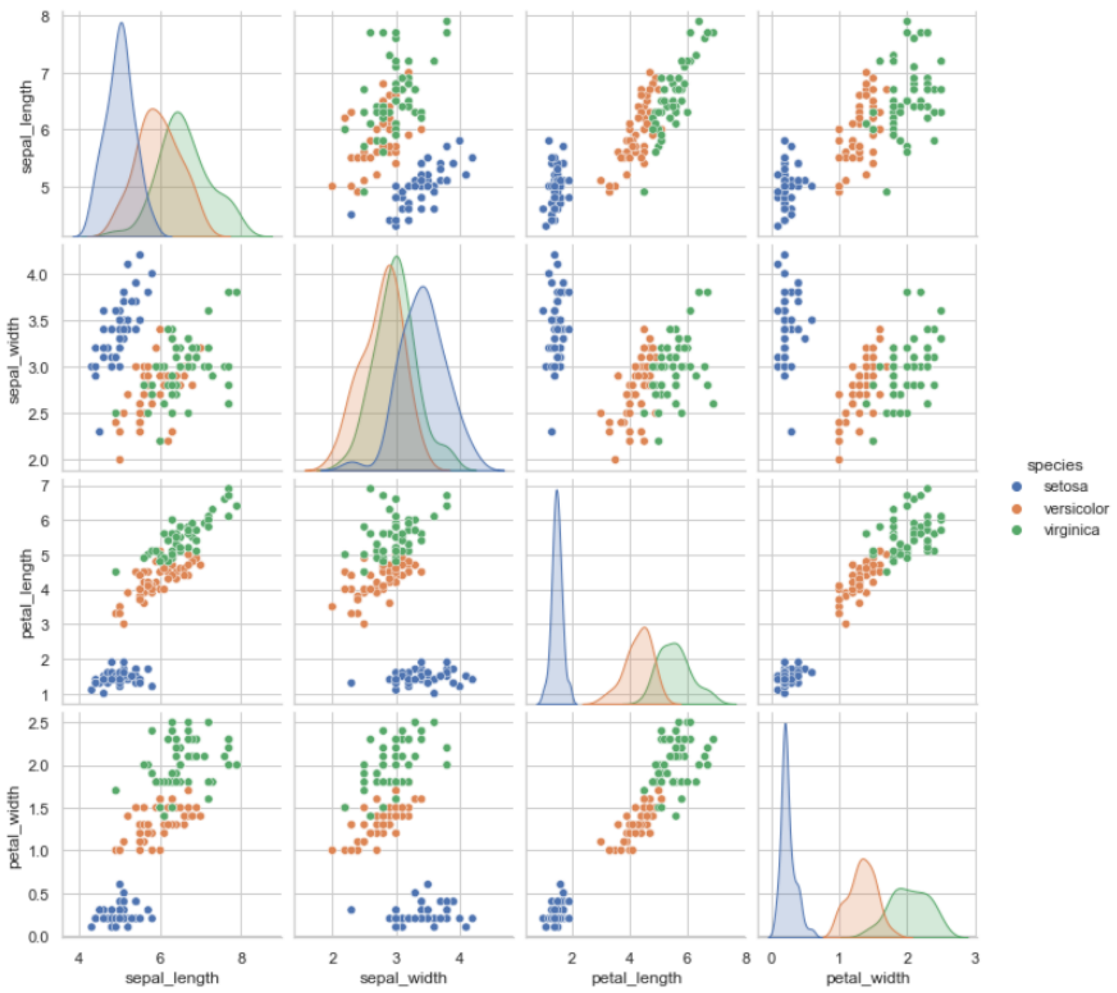
If you are still stuck with some error and need to give up on question 3.4, you can use the correct dataframe from `df = sns.load_dataset('iris')` for the following parts. But it would be really great if you were able to clean the data at the previous question and use that instead. As I mentioned: never underestimate the data cleaning.

[11]:
```
# I hope you will not uncomment the following line; but if you need to, it is␣
↪fine, do not worry:
# df = sns.load_dataset('iris'); df
```

4.1 [1pt] Start with plotting the pair-plot, using different colors depending on species. Explain the plots on the diagonal. Hint: a common error in the past has been if the types have been altered by the fixing in 3.4. To avoid that: use `loc` or `iloc` specifying *always* both row and column. To fix it with bruteforce: re-cast the Series to the correct `dtype` as you did in 3.1.

[12]:
```
sns.pairplot(data=df, hue='species')
```

[12]:
```
<seaborn.axisgrid.PairGrid at 0x7f75b0114520>
```

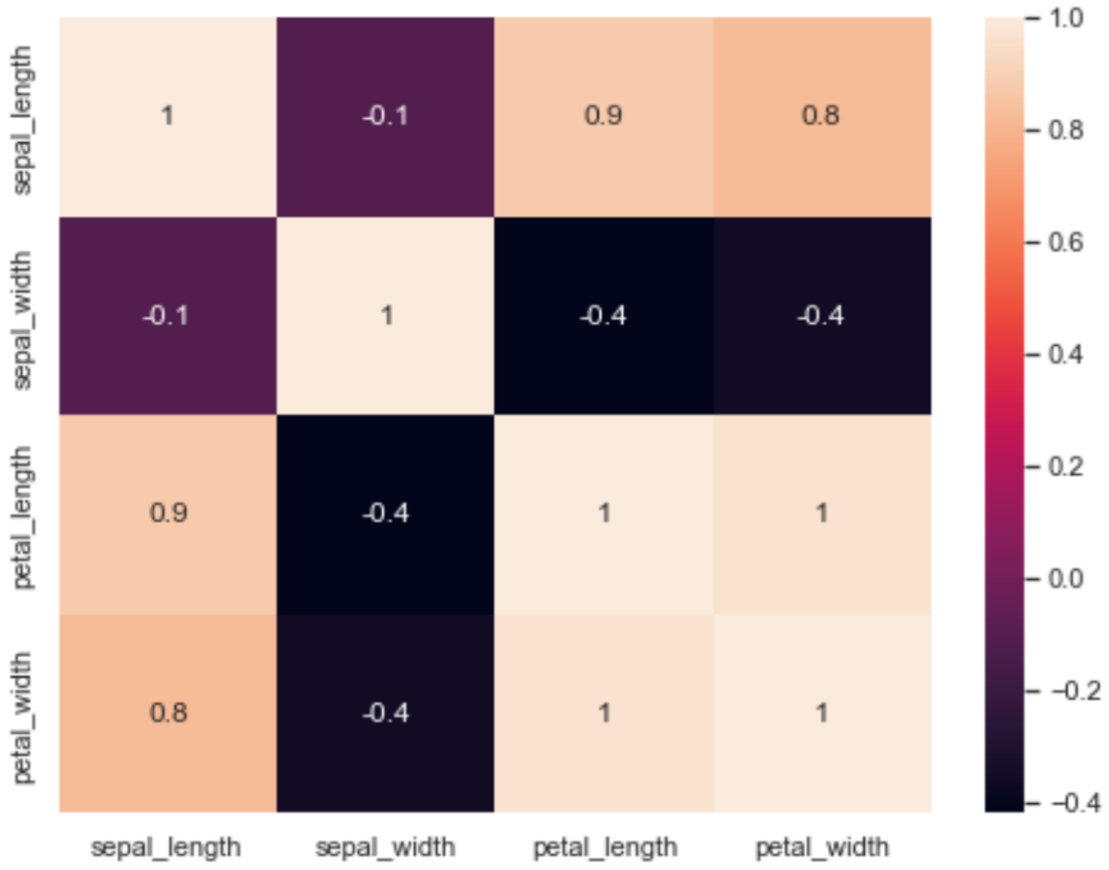


The plots on the diagonal show the distribution of the (one-dimensional) values for each class. Distributions that intersect correspond to data points that are not linearly separable.

4.2 [2pt] Plot the correlation matrix as a heatmap, adding the values on the squares as seen in the lecture's example. Explain what high values of covariance (>0.9 or <-0.9) mean from a statistical perspective, and the implications for the learning process. Hint: could it be a "common data problem"? Also add the parameter `fmt=` to round to 1 decimal.

```
[13]: sns.heatmap(df.corr(), annot=True, fmt='.1g')
```

```
[13]: <AxesSubplot:>
```

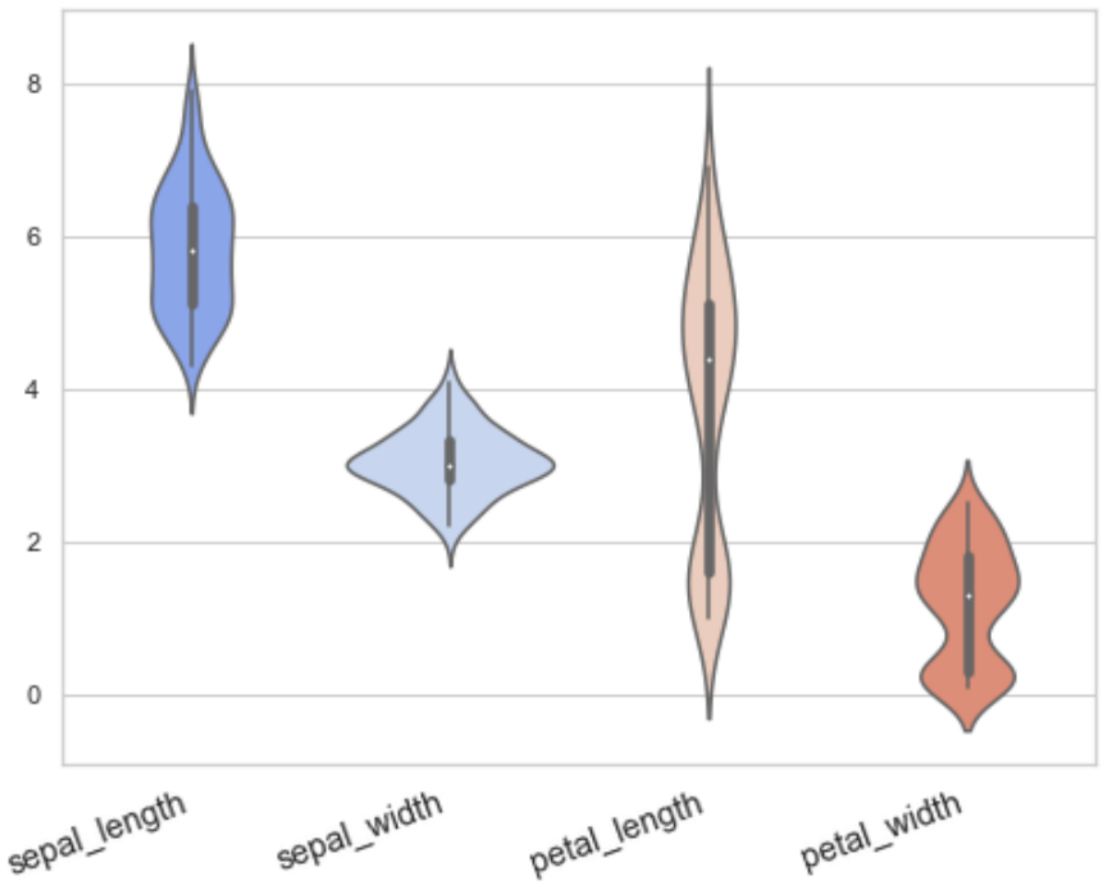


High (linear) correlation (in magnitude: close to $+1$ or -1) means that the two feature vary together in an indistinguishable manner. Here the high correlation between petal width and length means that the petal proportion is constant, and having both is redundant. We can learn the same if we drop one of the two.

4.3 [2pt] Plot a violin plot of the features. Use a custom palette. Briefly explain the advantage of the violin plot over the box plot. Palettes can be found [here]. You can use `plt.xticks(rotation=45)` (or similar) to rotate the x axis labels if they overlap.

[14]:
```
sns.violinplot(data=df,
               palette="coolwarm"
)
plt.xticks(
    rotation=20,
    horizontalalignment='right',
#     fontweight='bold',
    fontsize='large'
)
```

```
[14]: (array([0, 1, 2, 3]),
       [Text(0, 0, 'sepal_length'),
        Text(1, 0, 'sepal_width'),
        Text(2, 0, 'petal_length'),
        Text(3, 0, 'petal_width')])
```



The violin plot offers more details about the data distribution, especially when the feature is effectively composed of multiple clusters (e.g. both petal length and width).

6 5. Visualizing the performance

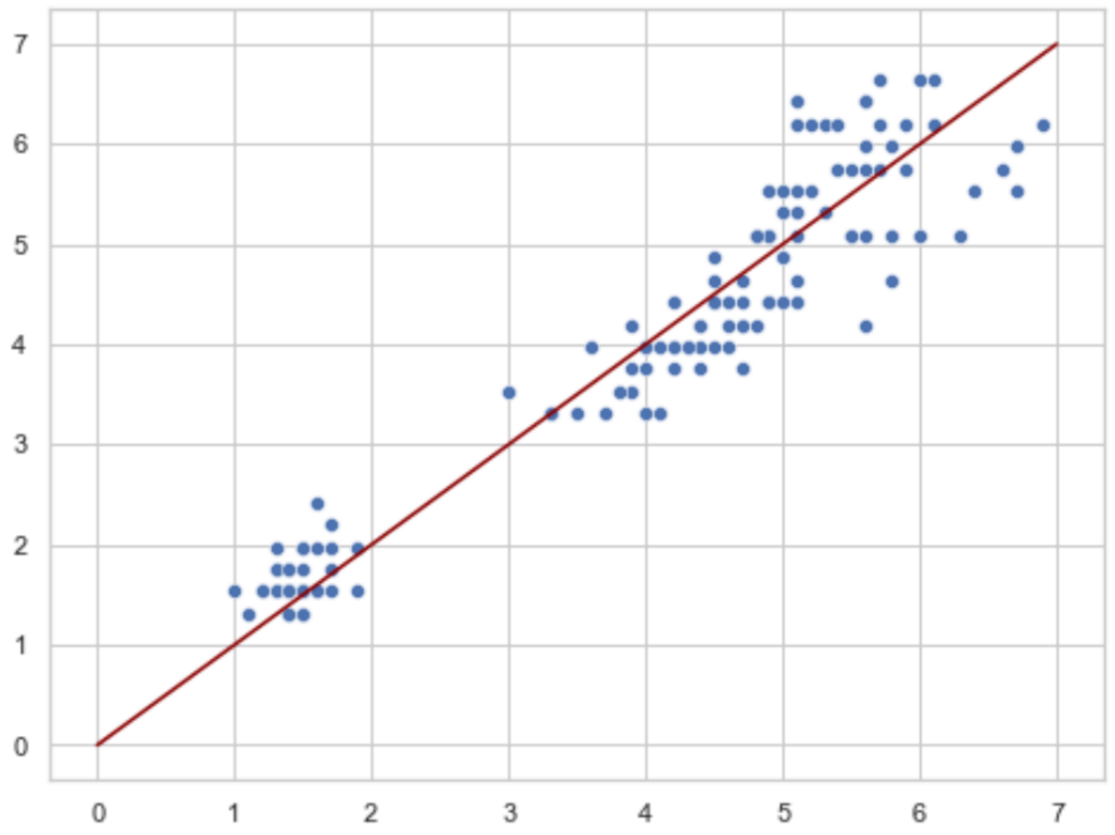
5.1 [2pt] Use scikit-learn to train a Linear Regression to predict the petal length ($\hat{y}$) from the petal width (x). Generate the Prediction over Observation plot. Careful, the integration between Pandas and Scikit-learn is relatively recent. If you find it confusing, stick to numpy arrays, for example:

```
x = df['petal_width'].to_numpy().reshape((-1, 1))
y = df['petal_length'].to_numpy()
```

Remember scikit-learn needs 2D input, so you need reshaping. To plot the diagonal line a simple line plot with two points will generate a segment, e.g. `sns.lineplot([0,7],[0,7])`.

```
[15]: from sklearn.linear_model import LinearRegression
      x = df['petal_width'].to_numpy().reshape((-1, 1))
      y = df['petal_length'].to_numpy()
      trained = LinearRegression().fit(x, y)
      preds = trained.predict(x)
      sns.scatterplot(x=y, y=preds)
      sns.lineplot(x=[0,7],y=[0,7], color='darkred')
```

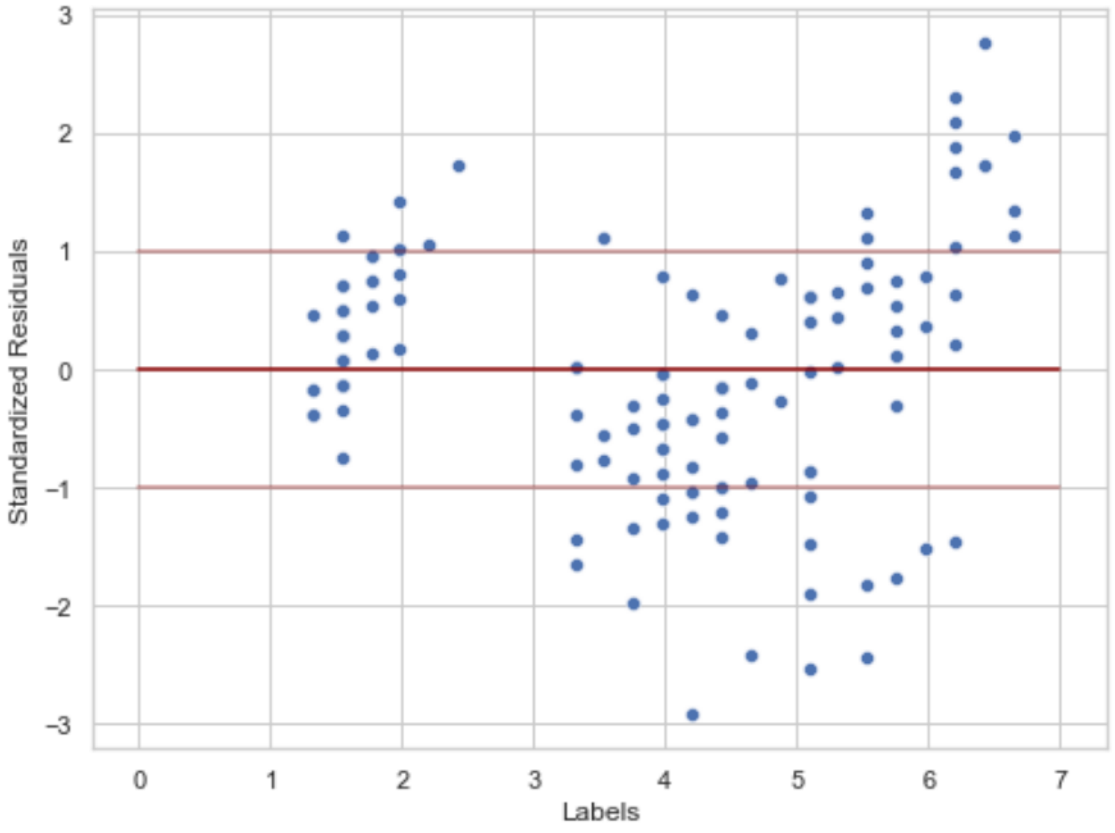
```
[15]: <AxesSubplot:>
```



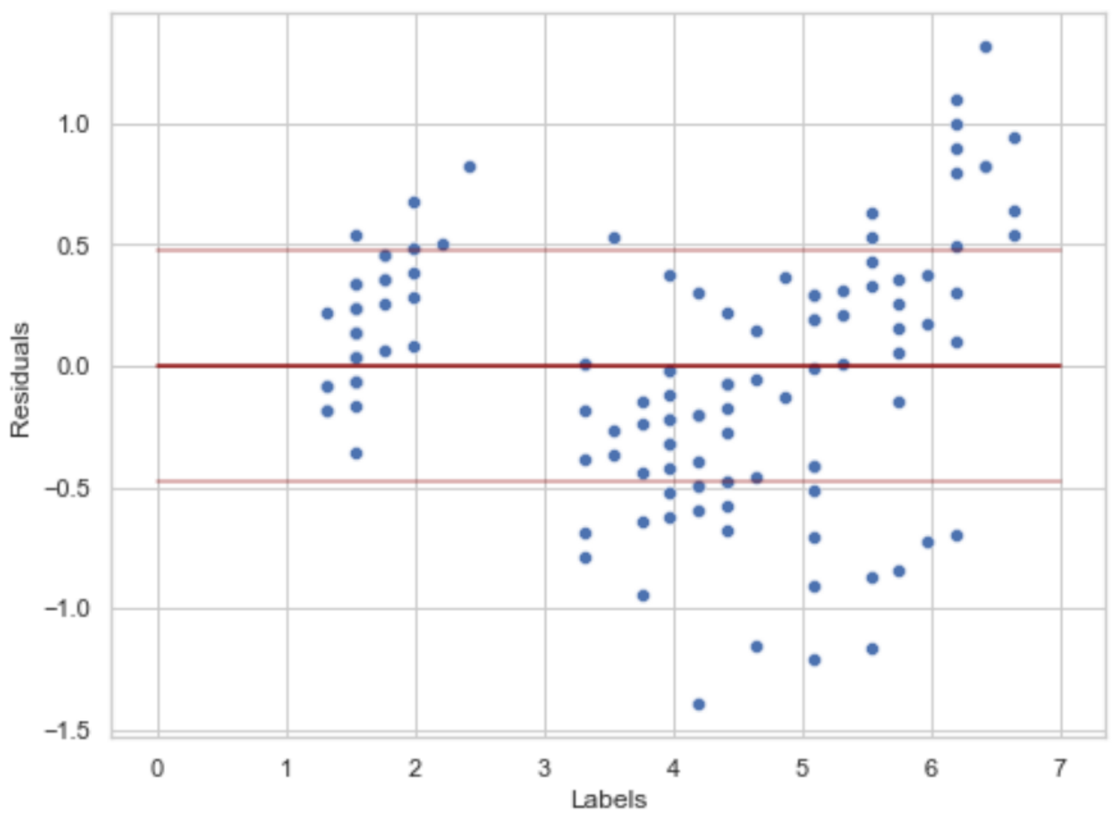
5.2 [2pt] Using the same model and predictions from the last point, generate the Errors over Predictions plot. Here you want at least one horizontal line for $y = 0$: use the segment trick from the previous question. Ideally it looks best with two more lines at $y = +1$ and $y = -1$ for the standard deviation. These can be made thinner or lighter in color to look even better :) I simply used the `alpha` option to make them slightly transparent.

```
[27]: errors = preds - y
      errors = errors / errors.std()
      sns.scatterplot(x=preds, y=errors)
```

```
ax = sns.lineplot(x=[0,7],y=[0,0], color='darkred')
sns.lineplot(x=[0,7],y=[1,1], color='darkred', alpha=0.4)
sns.lineplot(x=[0,7],y=[-1,-1], color='darkred', alpha=0.4)
ax.set_xlabel('Labels')
ax.set_ylabel('Standardized Residuals');
```



[29]:
```
# Alternative solution: normalization just changes the axis labels!
# Sometimes having the magnitude of the error helps (note the label)
errors = preds - y
std = errors.std()
sns.scatterplot(x=preds, y=errors)
ax = sns.lineplot(x=[0,7],y=[0,0], color='darkred')
sns.lineplot(x=[0,7],y=[std, std], color='darkred', alpha=0.4)
sns.lineplot(x=[0,7],y=[-std,-std], color='darkred', alpha=0.4)
ax.set_xlabel('Labels')
ax.set_ylabel('Residuals');
```



5.3 [2pt] For both Prediction over Observation and Errors over Prediction, write a few sentences commenting on the performance of the model learned by the Linear Regression.

- Prediction over Observation: the points correctly cluster around the diagonal, showing that the prediction minimizes the risk. There are no outliers.
- Errors over Prediction: Most error is in the range between -1 and 1 with few outliers out by little. The model's predictions will have an expected noise $\epsilon \sim \mathcal{N}(0, 1)$.

7 At the end of the exercise

Bonus question with no points! Answering this will have no influence on your scoring, not at the assignment and not towards the exam score – really feel free to ignore it with no consequence. But solving it will reward you with skills that will make the next lectures easier, give you real applications, and will be good practice towards the exam.

The solution for this questions will not be included in the regular lab solutions pdf, but you are welcome to open a discussion on the Moodle: we will support your addressing it, and you may meet other students that choose to solve this, and find a teammate for the next assignment that is willing to do things for fun and not only for score :)

BONUS [ZERO pt] Follow this tutorial to augment your answers to point 5 with 5-fold cross-validation.

BONUS [ZERO pt] Follow this tutorial but using our data, and generate all the plots of types that have not been mentioned in the lecture/lab. Comment a few sentences about which I think would be most useful to include in our set of basic plots for the next year.

BONUS [ZERO pt] Copy your Perceptron implementation from the second lab. Modify it so that at every update it computes the Risk (total Loss over all points) and saves it to a list (we did something similar at the first lab, remember?). Run your implementation on the data, then plot the Risk over Iterations, using a `sns.lineplot()`.
If you discuss these points on Moodle I will participate :) plus it will give you an opportunity to find other people in the class that interested with going beyond the points: I strongly encourage these to team together (remember you can change teammate at each assignment).

7.0.1 Final considerations

- This lecture+lab actually gives you most of what you need for the high-pay job of Data Analyst, so I guess there is one direct market application from this course after all :)
- You will need to be strong in all the topics seen so far to carry on with the course.
- Fixing and polishing your past assignments will make passing the exam so much easier. Now is the perfect chance to prepare the first 3.