General constraints for code submissions Please adhere to these rules to make our and your life easier! We will deduct points if your solution does not fulfill the following:

- If not stated otherwise, we will use Python 3.7 and greater.
- If not stated otherwise, we expect a Python script, which we will invoke exactly as stated on the exercise sheet.
- Your solution exactly returns the required output (neither less nor more) – you can implement a `--verbose` option to increase the verbosity level for developing.
- Add comments and docstrings, so we can understand your solution.
- (If applicable) The `README` describes how to install requirements or provides addition information.
- (If applicable) Add required additional packages to `requirements.txt`. Explain in your `README` what this package does, why you use that package and provide a link to it's documentation or GitHub page.
- (If applicable) All prepared unittests have to pass.
- (If applicable) You can (and sometimes have to) reuse code from previous exercises.

The automated machine learning methods you will learn about in this course help you to improve your skill and knowledge for applying machine learning in practice. The goal of this first exercise is to set up teams and learn about git and the workflow for future exercises.

1. **Form teams of up to 3 students** [0.5 points]

   Most exercises will require you to implement some of the techniques you learn during the course (in python 3.6)[1]. *Git* is one of the most widely used **version control systems** and allows you to easily collaborate with others on code from the same repository.

   Exercises have to be handed in *teams of up to* 3 students. When you have found your partner, open the following link (which will be later posted in Mattermost), create a group (you will have to name the group yourself) and both join that group. This will allow you to clone the template repository in which you can add your solutions to this exercise sheet.

   Note 1: If you have never worked with *git* before we suggest you take a look at this simple guide `http://rogerdudler.github.io/git-guide/`.
   Note 2: Make sure you and your team-mate are happy with each other. GitHub Classroom does not allow to change your groups mid semester.

2. **Get familiar with git and GitHub Classroom** [0.5 points]

   To show that you are familiar with the standard git *add*, *commit*, *push* steps add a file called `members.txt` to your repository. The file should contain the names of all members in the following way:

   ```
   member 1: name1
   member 2: name2
   member 3: name3
   ```

   We make use of GitHub Classrooms autograde functionality. Essentially, for most exercise sheets we will require you to pass unit tests which are automatically evaluated whenever you push to GitHub. To demonstrate this process, for this exercise we run a test that expects the above file to be present and to contain three lines as above (make sure to replace name1, name2 and name3. If your group has less than 3 students, just add any name you like).

   You will be informed if the tests executed successfully or not. (To run tests locally you can always use the provide Makefile via `make all`)

[1] We recommend you use anaconda to set up a virtual environment for the lecture, see `https://conda.io/projects/conda/en/latest/user-guide/getting-started.html#managing-environments`

Next, having learned about different ways to empirically evaluate the performances of algorithms and AutoML systems, in this exercise you will now implement some of these techniques. Add your code creating plots and outputting statistics to main.py (callable as python main.py).

We provide a simple Makefile which you can use to install all packages listed in your requirements file (`make init`).

1. **McNemar Test** [3 points]
   Two models are trained to classify images of cats and dogs. The result is stored in *MCTestData.csv* with $n = 500$ images. The function *load_data_MNTest()* loads the data as an $n \times 3$ *numpy array*, where the first column represents the ground truth. The 2nd and the 3rd columns represent the output from model 1 and 2 respectively.

   Implement a *McNemer Test* to determine whether the two models perform equally well on the dataset. In your solution state what is H_0, H_1 and return χ^2 for this evaluation.

2. **Two-Matched Samples t-Test** [3 points]
   TMStTestData.csv contains *error* values of two algorithms on $n = 419$ datasets, the function *load_data_TMStTest()* loads the data as an $n \times 2$ *numpy array*.

   Implement a *Two-Matched-Samples t-Test* to determine whether the two algorithms perform equally well on the dataset and return the test statistic t value for this evaluation.

3. **Friedman Test** [3 points]
   FTestData.csv contains *error* values of $k = 5$ algorithms on $n = 15$ datasets, the function *load_data_FTest()* loads the data as an $n \times k$ *numpy matrix* Err, where Err_{ij} represents the error of the jth algorithm on the ith dataset.

   Implement a *Friedman Test* to determine if all algorithms are equivalent in their performance and return χ^2_F for this evaluation. If this hypothesis is not rejected, you can skip the next question.

4. **Post-hoc Nemenyi Test** [3 points]
   Having found that all the algorithms are not ranked equally, now we need to utilize the *Post-hoc Nemenyi Test* to find the best-performing algorithm.

   Compute the the test statistic for all the algorithms pairs $\{j_1, j_2\}$. The results should be stored in a upper triangular matrix $\mathbf{Q}$, where $Q_{m,n}$ is the q value between the algorithms j_m and j_n

5. **Boxplots** [2 points]
   Create a boxplot[2] for error value of the algorithms which have the best and the worst average ranks stored in *FTestData.csv*. This exercise will not be automatically graded. To let us quickly grade your exercises please add your solutions to a PDF and upload that PDF to your repository. Further we expect all plots to have axes labels and a legend.

6. **Code Style** [1 point]
   On every exercise sheet we will also make use of *pycodestyle*[3] to adhere to a common python standard. Your code will be automatically evaluated on every push and you will be informed if the test fails. To check it locally, first *run pip install pycodestyle* and then *run pycodestyle –max-line-length=120 src/* to check your source file folder. Alternatively run *make checkstyle*

This assignment is due on 29.04.21 (23:59). Submit your solution for the tasks by uploading a PDF, a txt file and the codes to your groups repository. The PDF has to include the name of the submitter(s). Teams of at most 3 students are allowed.

[2] using e.g. `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html`
[3] former pep8