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 exclusively Python 3.6.
- 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.

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_F^2 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[1] 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.

[1] using e.g. `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html`