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.

Your next task is to get familiar with algorithm data and how to model an algorithms behavior on toy data as well as scenarios from the Algorithm Selection library (ASlib, `www.aslib.net`).

For this exercise assignment, we provide two (simplified) scenarios from ASlib: `SAT11-INDU` and `SAT11-RAND`. Each scenario consists of three files:

- `algorithm_runs.arff`: performance of each algorithm on each instance
- `feature_values.arff`: instance features of each instance
- `cv.arff`: cross-validation splits

We recommend to use the Python package `liac-arff` to read these files[1].
Both scenarios are runtime scenarios with a runtime cutoff (κ) of 5 000 seconds. To simplify your tasks, we will ignore costs induced by using instance features. Please note that some instance features can be missing. We provided you with code that replaces missing values with the mean feature value.

1. **Single Best and Virtual Best Performance** [4 points]
   Given an ASlib scenario, your task is to read the `algorithm_runs` files and to compute the PAR10[2] performance of the Single Best algorithm (SB) and the Virtual Best (VB) performance.
   (a) Your implementation has to pass the `test_stats_toy_data` unittests. This test is designed such that you have easy to interpret data when solving the exercise. [2pt.]
   (b) Your implementation has to pass the `test_stats_real_data` unittests. This test is setup such that you use real ASLib data. From this result you can observe that the difference between SB and VB can be quite dramatic for real applications. [2pt.]

2. **Models for Individual Algorithms** [5 points]
   The lecture taught you about how individual regression models can be used for algorithm selection. Fit a regression model per algorithm to the data (either the ASlib scenarios or the toy-data from the unittests). Use these models to predict the expected performances and select which algorithm should solve an instance, given its features. As regression model you can use anything available in `sklearn` (You are not required to adjust the hyperparameters of the chosen regressor).

   Again we provide simple toy data on which you can quickly evaluate the correctness of your approach. Your approach has to pass the unittests in `test_individual_models` (locally and remotely).

[1] See `https://github.com/renatopp/liac-arff#documentation`
[2] PAR10 is the penalized average runtime where a timeout is counted as $10 \cdot \kappa$.

3. **Hybrid Models** [5 points]

   Further you have learned about hybrid models. Implement pairwise classification models to vote on which algorithm should be chosen to solve an instance. Similar to the prior exercise you can use `sklearn` to choose any classifier you want to use.

   Make sure you implementation passes the unittest in `test_hybrid_models` k(locally and remotely). Note: We encourage you to apply your implementations to the provided ASLib data. However, this might require parameter tuning or more sophisticated methods to train a meaningful selector.

4. **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 10.11.2021 (14:00). Submit your solution for the tasks by pushing your codes and the txt file to your groups repository. Teams of at most 3 students are allowed.

[3] former pep8