Reminders to finish last exercises and to our way of working: -1, 0
Prepare before class such that you can present your result: 1, 2, 3, 4
Worked on in class, but you can prepare them at home: 5, 6

Carefully read the instructions below!

(-1) Finalize last exercise sheet

Finalize all exercises of the last exercise sheet and upload the results to your defined GitHub repository.

(0) Setup a git project on `GitHub`

In order to get a bit of `git` training done we work for all the exercises on GitHub.

(a) Use the project created for the previous exercises or create a new **private** project in GitHub

(b) Give the instructor (kandolfp) access to the project (see GitHub documentation for help)

(c) Create a `pdm` project in your repository and commit the necessary files.

(d) Create an appropriate structure in your repository for the rest of the exercise sheet (maybe have a look at the exercises to have a better idea first), not everything should be in the main folder.

(e) Try to structure your work on the exercises with git, i.e.

- Don't commit things that do not belong together in one single commit. Each exercise can be considered as a separate thing. Subparts of an exercise might be independent as well.
- Use meaningful commit messages `https://www.conventionalcommits.org/en/v1.0.0/`
- Make sure that you do not commit something that does not work - produces an error. If you have difficulties with an exercise you can also commit your best effort in this case.

(f) Add a README.md that explains what you are doing, how to run the exercises and anything else that is necessary (quick guide to `pdm`), maybe note your name somewhere.

(g) Optional: Work with issues, you can reference the issue in the commit message, GitHub documentation

(1) Law of large numbers We have seen the *law of large numbers* in action in the Monte Carlo Method computing π but how do we simulate a biased coin toss as discussed the lecture?

(a) Create a way to simulate a slightly biased coin toss with 51/49.

(b) Use `numpy` to have a fast algorithm

(c) Illustrate simulations for a growing number of coin tosses and include the reference of the bias.

(d) Can you make it such that the bias can be prescribed between 50 and 100?

(e) Visualize your results, including a line for the expected result.

(2) ENSEMBLE LEARNING I

We continue working on the palmerpenguins data set. Select two classes to work on for an `VotingClassifier` form `sklearn.ensemble`.

(a) Split your dataset into a training and testing set.

(b) Define three different classifiers with the classifiers we saw last time (e.g. SVC, DecisionTreeClassifier, LogisticRegression).

(c) Train and evaluate the `VotingClassifier`.

(d) Show the scores of the individual classifiers and the final result.

(e) Switch to soft voting and see if you get better results.

(3) ENSEMBLE LEARNING II

We continue working on the palmerpenguins data set. Select two classes to work on for an `BaggingClassifier` form `sklearn.ensemble`.

(a) Split your dataset into a training and testing set.

(b) Train a BaggingClassifier with the three different classifiers you used above

(c) Can you visualize the results?

(d) Combine the three BaggingClassifiers via an VotingClassifier and check your results.

(4) GRADIENT BOOSTING

Recall the regression exercises of the previous exercise sheets. Lets work on this with gradient boosting by successive training with DecisionTreeRegressor.

(a) Follow the steps from the lecture to train on the residuals with 5 levels.

(b) Improve on it with the correct class form `sklearn.ensemble`

(5) STOCHASTIC GRADIENT DESCENT

Implement a stochastic gradient descent method.

Instead of using the entire matrix X in the previous algorithm we randomly select k values in each iteration.

(a) implement the algorithm

(b) use $k = 1, 5$ and check how the convergence is progressing

(c) use the result of the stochastic gradient descent algorithm as initial guess for the gradient decent algorithm to finalize your result.

(6) RIDGE AND LASSO

The Ridge and LASSO methods uses a regularization term for the MSE function, as

$$Ridge(\Theta) = \frac{1}{m}\sum_m(\Theta^\intercal x^{(i)} - y^{(i)})^2 + \alpha\|\Theta\|_2^2.$$

and

$$LASSO(\Theta) = \frac{1}{m}\sum_m(\Theta^\intercal x^{(i)} - y^{(i)})^2 + \alpha\|\Theta\|_1.$$

We can find implementations in `sklearn.linear_model`.

Use these methods to fit

```
np.random.seed(42)
m = 20
X = 3 * np.random.rand(m, 1)
y = 1 + 0.5 * X + np.random.randn(m, 1) / 1.5
X_new = np.linspace(0, 3, 100).reshape(100, 1)
```

with different α values.