23/03/2023 Questions:

Best ways to corrupt the data: A lot of different approaches – just for missing values:

Other methods could be:  
• Set the pixel value to the "opposite" of what it was before.  
• Set the pixel values to random values.  
• Distribute the pixels chosen across all the images, such that some images are blurred more  
than others, but the total amount of blurriness is the same as if they had equal amount spread  
across the images.  
• Remove one set of color from the image (if the data is not black and white). Look at OpenCV  
masks.  
• Swap out pixel values in the image / scramble the image.  
• Not randomizing the indexes of the chosen pixels from image to image but keeping them  
the same - could refer to a mechanical error in a camera for example, producing the same blind spots in the images every time. Following up on this, if instead the chosen pixels are  
spread across the whole image, perhaps center them around a given area to better simulate  
this mechanical error.

“Wrong labels” is implemented by providing the randomly chosen indexes with a random label from amongst the other options. Perhaps here it should also mimic real life behaviour, where a person might be more inclined to mess up on the same kind of labels every time rather than at random, such that specific labels are more often wrong than others.

At which point in the process should the data be corrupted? Before training or purely for testing?

Implement my own federated learning algorithm.

Mention the other methods of data corruption:

* Outliers: Might be hard to implement for the specific models.
* Feature Noise: Feature noise occurs when there is random or irrelevant information in the  
  data that is not related to the target variable. This can cause problems for machine learning models, as it can make it more difficult for the model to find meaningful patterns in the data.

**Meeting: 23-03-2023**

Save file – reload model

Clean cache – clean garbage

Memory issues.

Problems with scheduling to clusters here.

We only test on true data (not corrupted data).

Robustness of the model.

Feature distribution of the images corruption.

Logistic regression is succebtable to corruption.

Don’t bother too much on the outliers

Higher percentage of corruption : 70%

0 mean static.

Greyscale – alter some, not across 50% - majority vote.

Pytorch Federated Learning.

Huggingface for models

OpenML

Keras – might be hard to unpacked.

Mechanical torque data

Omit the privacy (will drop performance).

**31-03-2023**

Altered the federated averaging implementations to also be able to use corrupted data (use the corruption algorithms. Now there is another parameter “corrupt” which determines if the data used for the training of the local models should be corrupted or not.

* There does not seem to be a test dataset implemented. The final accuracy is still on the training set. **Important**
* The pipeline for the federated averaging algorithms should probably also be fitted, such that it works as just 1 function. Probably requires a lot of different parameter settings for the function, if no better solution can be found.

Already now, the 1 hidden layer NN model shows great performance difference over the simple logistic regression model, even though the data corruption parameter is set to 90% for the label scrambling. The NN model reaches a final accuracy (on the training set) of 65%, whereas the logistic regression model reaches at max around 30%.

**Questions:**

* Optimizing the models? Specific parameters? Learning Rate, Number of hidden layers etc. Many different models require different optimization procedures and parameters. Are there specific values for the parameters that would be better suited for corrupted data e.g., would the learning rate be better off being low when the data is worse?

**01-04-2023**

The pipeline now works effectively such that the whole FedAvg algorithm is just one function which can be called upon different models with different parameters for said models, using the \*\*kwargs option for the models. The training parameters are also kept as a dict and instead called as variables using the exec command inside the algorithm.

**04-04-2023**

The method of calling the parameters inside the function / models did not work as intended and has been altered to work as a simple dictionary instead, this is however still “dynamic” and serves its purpose. Currently implementing CNN and trying it out for the CIFAR10 dataset.

Now a CNN is working for both the MNIST and the CIFAR10 dataset.   
The other models are also working for the CIFAR10 dataset.  
  
The CNN model requires dimension sizes that depend on the shape of the data, so currently this is solved by having two different models. One for each dataset. But maybe this can also be made dynamic? Also impacting the training loop, by how the output is called, as the input dimensions here also needs to fit.

In the fedAvg function, the different datasets also require different transforms, which are then hardcoded atm inside the function.

How about the learning parameters? Optimization of the models? Loss functions? Just use best case for data with certain models, specified from sources?

**Questions**

* Make it more dynamic somehow. What are the expectations to this aspect?
* Expectations for after the break?
* Optimizing the models? Specific parameters? Learning Rate, Number of hidden layers. Loss functions etc. Many different models require different optimization procedures and parameters. Are there specific values for the parameters that would be better suited for corrupted data e.g., would the learning rate be better off being low when the data is worse?
* There does not seem to be a test dataset implemented. The final accuracy is still on the training set. **Important**
* Help with calling the servers.
* Random Forest / KNN in PyTorch – Scikit-learn implementations are accessible but are infeasible to use in the pipeline directly as their training is different (easier, by just calling fit).