Q1. Could you describe your process for developing an ML system?

The Crisp-DM process model is widely used. And that is actually what we use. That is, first understand the data through exploration, set up hypotheses, develop the model, test the model and, if necessary, sharpen the hypotheses or so and go back into the circle. And then deployment, if it is good or build on the deployed model and iteratively bring in improvements. There are also different versions. Sometimes deployment is part of the circle and sometimes it is outside. We don't really do separate prototyping, but it's often the case that it then really goes into production. Of course, some things are discarded, but that is rarely the case, at least in my experience. The first time you do the cycle takes the most time, because you first have to really understand the data and deal with it and initially develop a model. The subsequent cycles are then simply shorter, but yes, basically it's develop, release, notice that maybe there are still weaknesses somewhere, e.g. through monitoring or simply things that you already notice in the development that there might still be potential for improvement. We go back to experimentation when we notice that there are weaknesses in monitoring or that we want to try out a new idea. What can also happen from time to time is that either requirements change or requirements are added. In other words, simply by using the system in live operation, we either recognize or stakeholders recognize that certain aspects are perhaps still missing or should be different. This also leads to further development. It is less common, but also possible, to get new input via conferences or papers and then try out whether existing solutions can be improved as a result. It's actually always about improving metrics and KPIs that you have defined for yourself, and in the best case scenario you go back into further development.

Q2. Do you continue experiments after deploying the ML system? If so, how do you go about doing this?

We are already tempted to tackle the prototype again. So notebooks are only really used in the very first phase to analyze models. Ideally, this results in a package where only functions such as training are accessed from notebooks. Therefore, if you develop that further, you also have to adapt the package to it. Of course, this is not always the case, but I think the ideal process is to use production-oriented code already in development. This saves time afterwards and makes development in the team easier and more accessible. The current setup is that notebooks are part of the package, but you could keep it separate. But you go back into the package, you either develop on the notebook or depending on how it's customized, you can also - these repos also lead to pipelines - deploy the models or the product with a corresponding system around it (customize experiments, etc.) you might only have to customize small things and then you can deploy that into a dev system and then compare corresponding things. So this going back to notebooks I don't think is absolutely necessary, but in my opinion is super individual from person to person and from product to product. Ideally, everything is versioned so that you can see these differences relatively quickly between a version that is running in production, so to speak, and a version that is running in a development environment, and afterwards you can ideally say that if certain tests or other requirements are met, you actually only have to press a button in the development environment to deploy to the production system. In the current project we are actually not yet so far, there are a lot of new things and there is little further development, because we are still under construction, but there is - you read more often that MLflow is in use and provides, for example, such a model registry, to then say, if the further development is better, then we only run a release pipeline, then that is on Prod. We do not yet, but imagine that as an ideal procedure. From my point of view a Jupyter notebook is super important only in the early development phase to get first insights. Once you are at a state

that is "clean" according to software engineering standards, then you can also do relatively much without notebooks, so even changes when you are now working with configurations and only have to adjust small things in them to run different models against each other. Maybe you have to add a function or edit a bit but then you don't have to change 1000 lines of code again but only 50 or small x. I would see notebooks more as a tool for testing. It can also be that you try a completely new algorithm or get new data and have to adapt something. This model is then a package and you then adapt this package. That's why you don't have so many versions, or they don't diverge so much.

Q3. Which tools do you use?

So I personally am a fan of VSCode, which is somewhere between editor and IDE There are also Jupyter Notebooks integrated. So I continue to work with Jupyter Notebooks or with Jupyter Lab, which is now becoming more popular I think. I program completely in Python, I haven't used any other language for a long time. And otherwise standard tools like Git, if necessary or mostly some cloud systems for data access or to store results. And otherwise the standard Python libraries like scikit-learn and visualization libraries. [Interviewer asks if plugins are used] No I have installed some but never used them consistently.

Q4. What are some common challenges you face in this process form a tooling perspective?

What I find super difficult are model evaluations. It's more of a downstream step, but it's also important for prototyping. I think there is still too little to evaluate models consistently. This is a point where we have difficulties in the project, because everyone evaluates differently and from my point of view there is no clean tuning, which simplifies the whole thing. So the classic train, validation, test split on data sets - some don't do validation, some do. Then you compare models with different expressions in datasets. Something else has to happen there. It's problem-dependent, but I think you can simplify things in different subareas, whether classification models, TS prediction models. There's super little there to support you. Each library provides functions, methods are defined for the model classes, but there is nothing across the board.

Q5. Which issues do you encounter when working with Jupyter Notebooks?

There are actually already many solutions in the sense of workarounds, but you have to make them available for yourself first, e.g. start a local MLflow instance for experiment tracking, do the imports and so on and then write some code and then use the notebook with it accordingly. You just have to know how. Scalability is another important point. You currently have to switch between different libraries if you want to scale. If the Jupyter Notebook could do that by itself, so to speak, now I have a small amount of data here and later a large amount of data and that would still go through the same way in a cloud environment. I don't see Continuous Integration as a good practice for Jupyter Notebooks. Data quality checks would still be practical. The rest I would claim already exists somehow that you can use with appropriate preparations. But they are not accessible for all Data Scientists. But that doesn't depend so much on the technical background, but more on the motivation to further educate oneself. I think if you actively pursue it, then the stuff is already quite accessible, although you can certainly do more. You have to have the motivation to then keep looking for new things or try things out, and that varies.