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

First, I get the data and analyze it. It depends on whether it's a huge amount of data, in which case I prefer to work with Spark or BigQuery, i.e. with something that can handle very large amounts of data. Otherwise, I work on the laptop with JupyterLab. And if I know that it's going to go in the direction of production anyway, i.e. if it's more than just a pure analysis, then I usually create a software project directly, because most things are done in Python, which means I then use tools like PyScaffold directly. When I program something, it's already in modules etc. and structured in a way that you can version it with Git and I don't have all my code floating around in some Jupyter notebooks. This project setup I do from the beginning, so you already have the structure, so you can do it as a Python package. This is first a skeleton and one directory of it is a notebook directory, where the notebooks are. That means you can work with both at the same time. So if I start with Jupyter Lab and have written my first function in the Lab and am satisfied with it, I can have PyCharm open and just copy-paste the stuff into the corresponding module from Jupyter Lab. I also wrote an article about how to work together. I think it's good if you start with a clean project structure, because you just start with Jupyter Lab, then to make that jump to create it for yourself later, then you have a glaring break, while you start directly with it and again from the beginning, if you are satisfied with a function that pulls over and imports, then it's easier, compared to the longer you wait with it. In my opinion, PyScaffold is a great fit. Simply because, if you think about the requirements, you want to have something to deploy at the end. That means you have to stick to the structure that you can build a wheel file with Python, for example, and there are also certain specifications that you need a package folder, that the modules are in there with these init files, that's all already a fixed requirement, so to speak. Then there are things like the source code should actually be in a src-folder, because then you have fewer problems when you work in a virtual Env, then you know that you really use the installed package and not the local package. Python first imports what is in the actual folder and only then looks in the search path in the Python path and therefore it is better to first have a folder like src, which means it is already clear that the package must be located under src and then a few best practices have also emerged over the years. For example, that you have a test folder, so the unit tests are separate and then, if you use Sphinx for the documentation of the source code, that has also more or less prevailed. Sphinx for Python, MkDocs there is also, but some prefer that, but Sphinx is already one of the main tools. That is to have everything under a docs folder and there the corresponding setuzp so that Sphinx takes the doc strings from the source code and makes an API reference out of it, that also helps incredibly when you work in a team. You write a function, you always document a bit about the doc-strings and if someone who is familiar with the source code can directly read this Sphinx doc. That's already predefined. In addition, there is something like setup, config, etc. and all this stuff and that the whole thing has to be in a git repo. Git is also more or less the one and only tool for version control and that you then somehow also use the Git tags to do the versioning, so with productive code you mostly want the semantic versioning, so that you have these bugfix releases and the minor/major releases, so that you can do the pinning etc. sensibly. When you deploy it is also a very important thing to say ok, I want versions, everything between version 1.x and minor version 2 because I want to take new features with me. But breaking changes I want to prevent quasi jumping up automatically. The next time I build my virtual Env for example, when I have that in my deployment process. That means a lot of the actual structure simply results from these requirements and PyScaffold tries to accommodate that eo a best practice structure somehow. I mean, there are other things like CookieCutter, Hatch, Poetry.

But in my opinion, there is not so much outside and docs and test folder and that really concentrates only on the basics and this Data Science Extension of PyScaffold, which you can still install, then also creates a notebooks folder, where you then store the notebooks, so that they do not fly around somewhere, and they can import the functions, because everything else is already in the package and then you have a data folder, where you have data files, if you work locally with the data and then subdivided into raw data, processed data, external data and much is simply best practices that have arisen. So it's important to have everything structured a bit, and in that respect I really like using the tool.

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

Because it is so great blurred with me, so is actually rather an advantage that you have no hard cut between that is my prototype and now I translate everything again. For example, if I have new ideas, I would open up a new notebook, briefly integrate the complete functionality that is already there and, for example, replace a part or write a new model, whatever, and see how it now works overall in the overall network. And that allows me to continue working and iterating quickly, because if I had a prototype and something in production, then both worlds. In my opinion, they always have to be together, because otherwise I always have this break. And the worst thing is, if I have a prototype, and assuming someone then translates it into proper software and then someone continues to work on the old prototype, then your huge break can arise and this iteration, which you actually want to have after CRISP-DM, becomes slower and slower and suddenly the person who is developing here assumes that the world looks like this and it is actually quite different in the meantime, because something else has been changed here and therefore, this must actually be together and I think you can only achieve this if you think directly from the beginning: this must become a package in the long run. So it's difficult from a method or process point of view when the stacks drift apart in such a way that they are brought back into line with each other, so to speak. Yes, so in my opinion you have to think about production right from the start, otherwise I'll make a prototype and somehow it will be done right again when it works, I think that's super hard. I have already experienced with so many customer projects, if one has this separation, then that wanders sometime except you have always more expenditure. With every new version, you have additional work, while at the same time the management thinks yes, that's already there, that actually has to become faster and faster. Every feature should actually be integrated faster, because so much is already there and you actually notice that it's getting slower and slower and that's why my solution was at some point when I said I had to have a code base from the very beginning, from further development to production and that must never move apart. It also depends on what this pipeline is all about, so if you now have such a complete ETL route in e.g. Airflow and then there is usually only a single part of the model that you are working on, which is a step of this pipeline and then the pipeline itself would rather be in another repo and I would then just have to coordinate with the person. That or I have control over the pipeline myself and I then show, that's where my model is then incorporated in the newer version, or that talks to the web service. Actually, you want to decouple it as much as possible. In the best case, whatever I build is its own web service and is then simply used by a larger ETL pipeline in some place and completely independent of my source code, so actually it must be a clear API, how to talk to my service and then the question is if I have deployed a container somewhere, then it is just against a rest API for example or if it is more a batch job, then this part of the pipeline may have created some virtual environment and

my library in version 1. 2 from some artifact store, but that it's really clear, this is version 1.2 and the thing knows which one is running and if there's a bug I know, ok, this affects version 1.2 and that I can continue to build on my stuff quasi independently until I'm happy again and then probably a version 1.2.1 release. There the coupling must not be strong in the sense, so I've often seen that then somehow the pipeline starts to clone the git repo and execute directly. Then it's just super dangerous, because you don't know what's currently running. You have to be very careful what you actually commit on which branch and that's why a typical software principle is to decouple as much as possible, so some things have to be tightly coupled, while others are extremely loosely coupled and the larger pipeline should, in my opinion, be as loosely coupled as possible from the individual steps, so that you can continue to develop without endangering a productive pipeline.

Q3. Which tools do you use?

already answered in Q1. [Interviewer asks if plugins are used] I use far too few. I don't know if I currently have any installed. For a while I use a plugin where you can collapse sections, so if you work with markups and headings you can collapse them nicely and next to it an overview. But I do a lot with the command line on the side, so also things for which there is certainly a plugin. I use Jupyter Lab really only because I find it cool that I see directly graphical evaluations, but so purely for development I use PyCharm, there you have much more and it is more responsive and therefore I actually prefer to stay in PyCharm.

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

What would be cool to automate is the whole dependency stuff, you always have to think about it. I mean I use a new library in the prototype and then I have to remember, ok I have to enter in the setup.cfg or in the pyproject.toml, or depending on where I maintain the things once, in addition maybe again in my Environment.yml, because I have to specify once the dependencies of the package and once those of the construction of my environment. This means that you always have to carry things back and forth and this could be automated. If you had a tool that looks at the source code all the time and says, hey you just used a new library, then I add it to the right places. So in the actual deployment at the end, even if you have now kept to the structure and have a proper Python package, there is always the question of whether you put it in a Docker container with an environment and whether you push the container somewhere and how you integrate it into a system landscape. That depends extremely on what kind of landscape is already there. If one had a standard there, it would be helpful. There are now a lot of MLOps tools and you get written to 1000 times on LinkedIn by people who think they have the perfect platform. As long as you only develop in that and use that, you can deploy things super fast, but I don't think there is a standard yet, that is only as long as you use the tools. If some guy would enforce a stack and there would be some standard there that I can best say at the end make me a Dockerfile and push it directly somewhere and run it as a web service. That is currently very individual and you have to write ultra much to set up this whole CD pipeline first and you have the feeling that you are actually always doing the same thing, but in a slightly different way and adapted to the customer landscape. But it's the same thing over and over again, so if there was a standard or that somehow became established, that would definitely be cool.

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

Reproducability is something I find super annoying with JN. That is how often you have the notebook new and suddenly come out completely different results, because it was some intermediate state, because you have the order, etc., I would find cool. Reusability of code I

have yes quasi by my PyCharm workaround that I then immediately pull over and then re-import, that works for me. But if I didn't have the workaround, that would probably be a huge problem for me too. Scalability. I mean, I can open as many notebooks as I want, but sometimes it also annoys me insanely that while such a calculation is running I really can't do anything like that. I can on another notebook, but that blocks. So for the evaluation of experiments I always use this Neptune AI and then I do it outside of a JN anyway, because I'm usually also interested in the runtimes and such and then I want to know quite clearly what actually ran there and also want to have it reproducible. Collaboration. So, if several people are working on the same notebook I don't want them to be in sync, that I can see someone typing like in Google Docs, but the problem is if I have 2 and I want to see the differences, then it is currently super difficult, there are also a few plugins that make it easier, but that is still not well solved.