Audio file

Your Recording 24.wav

Transcript

00:00:01 Speaker 2

We're going to look at Jupiter notebooks. We need to understand what the problem is that Jupiter notebooks solves. We're going to see how Jupiter encourages experimentation and collaboration, particularly for the data scientist. We're going to look at ways that we can run Jupyter notebooks. Now, this will lead us towards Sagemaker, but we don't need Sagemaker for Jupyter. It's just that it's a good idea. And we'll look at some key takeaways and the results we expect to achieve by using Jupyter Notebooks. The most common development language used in data science is Python. So if we are going to develop some models, we're going.

00:00:31 Speaker 2

Need a Python environment.

00:00:32 Speaker 2

But what kind of Python environment do we need? We could start out with a Python shell. This is often referred to as a read, evaluate, print, loop shell. Now that means that if you have Python installed, let's say on your local computer, you can literally type Python And enter the interactive shell. Now from there you could run Python commands like print, hello and you would see the results instantly in the interactive shell. Now this is helpful when you're learning Python to try out Python commands and immediately see their results. But it's not really intended as a development environment for you to save your multi line code if I want to.

00:01:02 Speaker 2

Write code that is maybe 1020 or 100 lines long. Then generally we're writing Python scripts and we're going to need something better than the Python shell. So what are the alternatives for running a Python environment? Well, we could use an integrated development environment or IDE. There are many options available to us. A common one and one that I like to use is Visual Studio Code. Another might be the Python native IDLE IDE. Another favorite one is Intellij PyCharm. Now why these tools are well liked by Python developers is they provide us with a number of productivity assistance for.

00:01:33 Speaker 2

Example code completion. So I start to type a particular Python function name and I get auto completion showing me for example the input parameters required to a particular function. Like AI suggestions I get syntax highlighting so if I mistyped something I've got a typo, I'll get a little squiggly line and a suggestion on how to change that into valid syntax code. These Ides give me debugging tools so I can do things like set a watch in a variable and if I do that then that means I can look at the

value of the variable and as I stick through my code line by line I can see the variable value change and I can see and understand my.

00:02:03 Speaker 2

Better, or I could set breakpoints, meaning that the code will run uninterrupted until the breakpoint, after which I could run it line by line again, typically watching my variables at that point. Wonderful debugging tools. So these tools that are in an IDE are all about making you, the Python developer, more productive and helping you troubleshoot quicker. So that's great. And we absolutely could use an IDE for the development of our Python code for data science, that is given that could be your approach.

00:02:28 Speaker 2

It's just that there's something else as well. Firstly, data exploration by a data scientist is going to require multiple.

00:02:34 Speaker 2

Python tools, in particular with visualization, when I'm getting to know my data and I want to understand my features that are correlation, that distribution, I'm generally going to want to visualize that. And there are two packages in particular that I would use for that. One is called matplotlib and the other is called seaborn. Both fantastic at producing visualizations around my data. But that's going to help me understand my data better so I can perform my feature engineering better and better prepare my data for ingestion by the training process and the algorithm. And we think that the read.

00:03:04 Speaker 2

To print loop environments like the Python shell or a developer IDE, they're not quite enough for reproducibility. And what we mean by that is, imagine I'm collaborating with other data scientists and creating an experiment where I think with this particular feature engineered data set and this particular algorithm, I'm going to get good results. Well, I want to be able to share that experiment with another data scientist who can look at my working and see how did I get to that point and maybe make suggestions or make iterations on their own variation of my experiment. So if I want to be able to reproduce someone's experiment, how did I get to my point, if ever?

00:03:34 Speaker 2

School, you had that position of you must answer a question and you just simply gave the answer, but the teacher or the professor would be saying yes, but you must show you're working, show you're working, show how you got to that point. And the environment that we want will enable us to do that. So the solution for the data scientists to have a good Python environment that supports their needs is something called Jupiter.

00:03:53 Speaker 2

Now Jupiter provides us with an interactive web-based coding environment that is ideal for experimentation. Is ideal because of what it provides.

00:04:03 Speaker 2

It allows us to break our code up into what we call cells or chunks. So rather than having maybe 100 lines of Python to do something, I could maybe have well, my first few lines of Python that are performing some data cleanup tasks. And then I might have a separate set of cells that are related to the task for feature engineering. And then I might have a different cell which starts a training job. And the great thing here is as I can run each cell, each cell might be 1 Python line or it might be 10 Python lines, You decide. But as I run a cell, I will get the standard output of that cell in line within the Jupiter document. And This is why.

00:04:33 Speaker 2

Is kind of a hybrid between a redevaluate print loop and a traditional IDE. You're able to see the cell, edit it, run it, see the output, but it just extends the length of the document. Now when we save our Jupyter document, we call this a Jupyter notebook. Then not only is the code that you've written is kept, but the output that you obtained when you ran the cell is also kept. Now you might not want that, and you can choose to clear previous output if you wish, But if I was wanting to show my experiment and maybe get the consideration and input from another data scientist, I could send them my notebook.

00:05:03 Speaker 2

You could open it up and you can see the code that ran and the output that I got because the actual output of each cell was stored as part of the notebook. So again, think of science experimentation. You might have an hypothesis that you think that this algorithm, this feature engineered data set together, I think will give good results. Did it? Yeah, they were. OK, let's iterate again. So you could save the notebook and even version control it in a Git repository. So you can always get back to a point in time and know how you got those particular results. Within a Jupyter notebook, we have this concept of cells. So Jupyter Notebook.

00:05:33 Speaker 2

Is made-up of one or more cells, typically multiple cells. Now, my Python code will run in what is called a code cell, OK, and then when that Python code runs any standard output that it produces, for example a print statement, or if you call the visualization library to draw a chart, that would be produced in line within the document and therefore would be saved. When you save the Jupyter notebook, it's typical that your document will have more than one code cell.

00:05:56 Speaker 2

But it gets better because you can add a cell to a Jupyter notebook that is called a markdown cell. And markdown cell allows you to annotate your code. Now, this is so much more than just commenting your code, which we should always do anyway. This is instead allowing you to create a Latex compatible description of what you're doing and why you're doing it. So you might have a paragraph about what you're trying to achieve in your experiment. You might have some explanations around the data sources or any tasks that the next Python code cell will perform, and any instructions for other data scientists who are following in your footsteps.

00:06:25 Speaker 2

So for example, I might want to highlight that watch out for the learning rate using this particular algorithm because so you're embedding knowledge into the notebook so that others may leverage your work. So here we've got another Python code cell. And again, we would have the standard output of that Python code cell produced in line into the notebook. In the second example here, maybe that second Python code cell was calling seaborn or matplotlib library to generate a nice chart showing us distributions of our features. And that graphical chart would be again produced in line to the document. And if I chose to save the whole Python document, then I.

00:06:55 Speaker 2

Be able to open up and see that visualization later on.

00:06:59 Speaker 2

It's important to understand that my Jupiter notebook does not execute automatically. It is up to me to run a Python code cell interactively. So I choose to run Python code cell one, look at its output, make a decision if I needed to make changes or modify it. I might need to run it again to get a different output. Once I'm happy, I could then move on and maybe add another code cell or another markdown cell to explain what it is that I am doing. So let's now compare the different Python environments we've discussed here. We can see how Jupyter notebooks compare in feature set compared to a Python shell or a typical integrated development.

00:07:28 Speaker 2

Environment and we can see that the needs of the data scientists are met better by the Jupiter notebook. We can see that that interactive coding capability that we really like to experiment with. Yes, the Python shell has it, but because we can save the output of standard out in line with our Python And Jupyter notebook. The notebook that wins there, the inline support for data visualization and the fact that we can annotate what we're doing in an experiment using Markdown cells make your Jupyter notebook highly reproducible. Environment is cloud friendly and in fact we'll see shortly how we could run that within Sagemaker AI. We can support different kernels.

00:07:59 Speaker 2

We've got more about what we mean by that shortly, and we can support teaching and tutorials in a way better way because of the annotation support. If there was a disadvantage though to Jupyter notebooks, it would be that the debugging support is not as powerful as what you would find in an IDE.

00:08:12 Speaker 2

Python shells, as we know, are this redeveloped print loop. Yes, they're interactive, but they don't really support having data visualization and they're not ideal for reproducibility or collaboration. Ides like Highcharm. They have some capabilities in the space of data science for interactive coding and visualization support, but where they excel is in their debugging support, in that ability to support the developer writing good code, syntax highlighting, AI integration, code generation, and setting watch points and variables step over step into type functionality.

00:08:42 Speaker 2

If I'm writing an application in Python, like a Flask or Django application, I would be choosing PyCharm or Dscode, absolutely. But that interactive nature of experimentation and data science, Jupyter notebook would be my preferred environment. So Jupyter notebooks are the way that we are going to run our Python code in order to experiment in data science. When we look at the Jupyter product, you'll find, however, that there are kind of two versions. There is Jupyter and there is Jupyter Lab. Now, Jupyter was launched first.

00:09:07 Speaker 2

It's a simple interface that shows one file at a time. So if I create a Jupyter notebook.

00:09:13 Speaker 2

That has multiple code cells and markdown cells. I can open that notebook in Jupyter. I can run it, I can edit it, save it and close it. But I can only work on one Jupyter notebook at a time. It's a simpler layout. It's a cleaner interface for editing these Jupyter Notebook files. Now, Jupyter notebook files will have the file extension ipymb, so it's very easily identified. So how does that differ from Jupyter Lab? Well, a few years after Jupyter was launched, Jupyter Lab was an enhanced version of Jupyter.

00:09:40 Speaker 2

Now this was a more modern user interface. It felt a little bit more like an IDE.

00:09:46 Speaker 2

But yet still maintained that capability of having interactive elements, having code cells and visualization support markdown cells, and saving standard output. So the notebooks themselves are the same. It's simply that the IDE looks and feels a little bit more like how maybe VS Code or Piechart looks. It's got better support for integration with things like git repositories, but crucially it gives us multi-tab multi pane support. Now that means I can work on multiple files at the same time and just flip between the two tabs, or three tabs, however many files you have open at the same time. I find this.

00:10:16 Speaker 2

Incredibly helpful because I might have a Jupiter notebook open from a previous project as well as my current project. If I know I've solved the problem in the past, maybe I'm just going to copy and paste some code. Or maybe I want to open up a CSV file and expect it directly in the Jupiter Lab interface. So having the ability to open multiple files at the same time, yeah, that's definitely going to be required. So this will enable me to edit my notebooks text files and have these open side by side. I can even open up a terminal. Now what that means is that I get a direct terminal with the host running the Jupiter Lab environment. Now this can be helpful if you need additional Python.

00:10:46 Speaker 2

Packages Now to install Python packages, we tend to use something called pip where I could say pip install and then choose the package like matplotlib for example. So if I need to be able to do that, how do I get a command line with the host that is running the Jupiter lab process so I can have a terminal tab in my Jupyter lab so I can install the process, a package and then flip back to my notebook? Jupyter lab though also introduced extensibility of the UI. So there is a complete and

massive marketplace of extensions for Jupyterlab so that we can bring in additional functionality into the user interface.

00:11:15 Speaker 2

Now that might be called generation capabilities or code linting, or documentation capabilities, Git integration. Numerous different vendors and open source providers have written plugins that you can enable into that interface to provide a more integrated development experience that will include AWS themselves. So if I want to make use of Amazon's large language model, Amazon Queue, so I can ask questions of the AWS large language library, then I can get that direct kind of chat interface directly within the Jupyter Lab interface. I can only do that because Jupyter Lab is.

00:11:45 Speaker 2

With these plugins, let's take a look at the Jupyter Notebook interface. Here we can see we're looking at a Jupyter notebook. Now I can see straight away that I have a number of cells. In the first cell I have Welcome to demystifying Sagemaker and I can see that that's a markdown cell. So I can use things like the pound sign or hash symbol to indicate it's a title. If I want to run that cell, in other words, render that markdown cell, I would just use shift enter on my keyboard and it would be displayed appropriate to its titling. In the first code cell, I can see I'm assigning 2 variables, A to 10, B to five, and then.

00:12:15 Speaker 2

The result of the sum of these two values. So again those I can click into the code cell, I can make changes as I would in any other IDE and then I can just hit shift enter and the cell will run. Now I can see in that code cell that when I do shift enter and it runs, what will happen is an asterisk will appear in where it shows number one just to the left of the code cell where it shows number one right now that will be an asterisk while the code cell is running and then it will change into a number and that number will be an ordinal number indicating the order in which the cells were run. So if that was run first.

00:12:45 Speaker 2

Number one, I can then see the output of that code. So because I had a print statement, it would print to standard output that's captured in line to the Jupyter notebook. So if I save the Jupyter notebook, then I save not only the code and the markdown, but also that value as well. And the save icon just up there in the ribbon bar. I can see in my second cell, it looks like I'm creating a list of values called prices, and then I'm iterating over each element in that list. So for price in prices, I'm printing what the current price is. So here I'm just getting, again, using print, I'm employing standard output.

00:13:16 Speaker 2

Again, has been captured. When I run that code cell in the main body of the document, I can see each iteration because I run that second code cell. After I run the first one, I can see on the left hand side of the code cell the indicator that that's the second execution. Notice as well that your file is an ipynb file. So Jupyter notebooks, whether you're using the original Jupyter user interface or

the Jupyter Lab interface with multiple tabs, the actual Jupyter notebooks themselves are the same. They are idymb files.