Welcome to this course on Machine Learning on CDP. Let's look at the content of this course. So first we will studied a CML tool. So CML is the machine learning dedicated service that sits on top of the CDP. It contains such features as a workbench, which is a a notebook environment and also applied machine learning prototypes, which get you started with a cutting edge machine learning technologies. It's also plugged into a shared data experience layer. So while you are doing machine learning, your data is still tracked for data governance, and the Ranger plugin also manages to authorizations through to your data. CML also contains a tool for visualization so that you can easily do dashboards who tell your story. Also, CML interfaces with the Admin Flow Experiments module so that you can track the experiments you will be doing in order to find the best machine learning model. So after this, we will review Spark dataframe operations in the context of data science, so how to transform your data so that it's available for machine learning algorithms. And then we will study the machine learning library from Spark, which allows you to do machine learning at a petabyte scale. So we look at some your usual machine learning use cases such as regression, classification, clustering, recommendation, hyperparameter tuning, pipelining. So, we do all this and will come up with several models, not all of them successful, but the goal is not to solve the problems, but to learn how to use the machine learning library. And then we'll see how you can deploy those models using CML, using two techniques, mainly, and then we'll come back to CML, the tool, to see how we can leverage the auto scaling because it's the Spark we'll be doing is Spark in communities. So it has auto scaling capabilities and also it can leverage GPUs. So we'll study the implications of using GPUs and how easy it is to set a GPU range for your workspace. Last will study how to continuously monitor the performance of your deployed models so that you know when to retrain your model. And for this we will use the evidently library. So in the next chapter I'll introduce CML the tool.

In this chapter, we will study CML. So CML is a data service that sits on top of Cloudera Data Platform, so it needs a Kubernetes (?) layer, which can be Cloudera's own or and OpenShift Kubernetes layer. It's the successor to CDSW, so it's a tool that is dedicated to data science themes, which goal is to streamline all the activities around machine learning from accessing the data, transforming the data, training modules, and deploying model; so the full lifecycle of a machine learning project. This is a traditional view of machine learning. You analyze data, looking for insights and then based on those insights, you choose a machine learning algorithms that you try to tune to get the best accuracy for your machine learning model. This process is driven by experiments. Eventually, you can also change the machine learning models or combine the machine learning algorithms to create the best model. These train models books is based on iterations, and to keep track of those iterations, you need some kind of experiments feature that allows you to track not only your (?) parameters, but also the accuracy that you get. And then you want to not put your machine learning model on a shelf, but to deploy it and serve it to to consumers so that they can pull via an API and get predictions. And all these activities you want

them to be on top of a of a system that allows your data science teams to collaborate in a secure environment. So with secure access to data. So that's another view of the same workflow. This you add the, the dimension of the edge. So getting data from the edge using data flow and streaming. And then yes, there's data engineering, which is on the critical path of all activities. But then there's machine learning but also data warehouse. So some- some answers or insights can, can be obtained by traditional SQL queries and you can also store your data in a operational database and create data products. But all of these activities can be tracked for data governance, but also for authorization via Ranger. You know, Atlas, you can track the lineage of your models and in the data catalog, you can also get a consolidated view of the access to your data and to the metadata. So it's an asset oriented view which combines data from Ranger and data from Atlas. So CML is a very open tool, so it can work with all these systems, which is a good thing because this area of big data is very vibrant. So there are a lot of new projects that arrive. So it's a good thing to be open. A CML doesn't enforce the use of Spark. It can work also on small data and data scientists' favourite packages like a Scalar or Tidyverse. So this is another view of the same journey with the roles. I said that data engineering is on the critical path, so is the role of data engineer. One thing to note on this slide is that the difference with all the other roles of data scientist is the science path. So you expect the data scientist to have a background in statistics, and that's not the case for the other roles. So that's what the distinctive feature that sets data scientist a part.

I told you that CML is the big brother of CDSW. CDSW was the previous tool for data scientists or CDP. But it's- CDSW isn't a data service, so it doesn't sit on top of Kubernetes. It doesn't have auto scaling, and it doesn't have all the new features are developed for CML and CDSW, now adays, doesn't get any new features. We look to in closer detail. So CDSW is close to the end of life and CML is where the R&D goes. The other main difference is that when you using Spark with CML, you are using Spark on Kubernetes whether you are going to use Spark on YARN or CDSW. And so in CML, what you have is auto scaling and you don't have auto scaling in CDSW. That's the benefit of using Kubernetes.

An administrative entity in CML is a workspace. To do anything, you need a workspace, and in workspace you will create projects. That's the hierarchy of data abstractions in CML. So what is a workspace? Well, it's defined by settings of a range of CPU and the range of GPUs and the kinds of of machines that will be involved. So you see is it is to define CPU settings and GPU settings and you see the symmetry between the two. To access a workspace then you have to- when you're on the landing page of CDP, you need to click on Machine Learning. So this is the screenshot at the top of the slide is from the public cloud version of CDP because you have all those data services, including Dataflow and Operational database. So when you click on Machine Learning, you land on the machine learning workspace landing page, and you have your tabular representation of all your workspaces. So within a workspace, then you

get projects and in projects define a scope for the libraries for the team that work on the project. So a project contains code, data, and assets. And project code can be written in by Spark or Python or R or Java. So you can manage the project files like you would any file system. One caveat though: be careful about deleting files and directories within CML. There is no undo, so be careful. Models are deployed within a project, so a project is also a scope for models. If you have a a file that contains a function that can perform predictions, to create a model, you use the model option in the on the left hand side: navigation pane. And then you fill this form that you see on the right hand side that points to the predict function within the predict.py file. You provide an example input and example output, and then you have the- your model will be deployed. So CML relies on the Docker container. So what what CML will do, it will copy all the files and assets from your project into Docker container for which you have also specified the and the- we don't see this in the screenshot, but you specify also the CPU and the home of the container. And that container is built with the Docker file as and this container also contains- sorry, endpoints. Well, one endpoint that is secured and that will allow you to call the public function using appropriate Python code or R code or call in shell. So those are all the data abstractions you find within a project. So you find sessions to those are interactive sessions where you you can write the code and execute the code. Experiments is a bit like sessions, but nothing interactive and should contain code that allow you to track files and metrics. Models: that's the model that you the model feature that we just described. Jobs is an unattended session, and you can create pipelines of jobs you want. Applications is one way that you can deliver your machine learning models to business users using a UI. So in applications you'll find a UI that will call a model and get displayed the results to the business users.

So there are some roles in workspaces. So you have roles for the environment and roles for the workspace. To do it, the roles are if a user has more than one role, the role with highest level of permissions takes precedence. And if a user is a member of more than one group, and the user gets the sum of all permissions. So the environment roles are MLAdmin, MLUser, and MLBusinessUser. So you see the characteristics of each all on this slide. The business user is pretty much read only kind of role. the MLAdmin id what you expect from an admin. And in between the two you have the MLUser which can run workloads on all workspace provisioned. So for a given workspace. Then you have the the same three levels the read only one for the admin workspace user, then admin for the workspace, and the user for the workspace. So for a given workspace you can access- manage the access using the actions menu on the right hand side and manage access option, and you select the user and check the boxes appropriately.

Projects are independent (unintelligible). As I said before, they define the scope and the allow for collaboration with other users. So you can either- the project can be created by a team or project can be created by an individual user, and that user can add other users to his project. So for a new project, you have all these characteristics to provide; so the name, the description, which is optional.

This is where you choose whether the the project will be private or public. And then for the initial setup, you have several choices. So you can start from a blank slate or you can use a template or an applied machine learning prototype, or you can use local files, or you can also use git to initialize your project. Furthermore, you can choose the the runtime that you will use, so you have the choice of the kernel. So in this example, we are using Python 3.7 and you see that we have a checkbox that will allow you to add a GPU enabled Runtime variant, so you can manage the visibility of your project so that the choice between public and private: if you make your project public, that you will provide read level access to everyone with access to the CML; if your project is private, you must explicitly add someone as a project collaborator and there are several levels of visibility when you add a colleague. But the one that allows your colleague to work on the project is contributor. When you make your project private, then you have to invite your colleagues to the to the project and then you have to set the the access that you will grant them. So you have the read only viewer to the admin which complete access, the operator which allows the user to start and stop, and the contributor which gives the the user write level access to the project. In projects that are owned by the team, you need to specify who work together and you can create and manage the teams to modify the account profile and manage members and also the team has SSH keys so you can modify them as an administrator.

There are several settings menu in CML, which can be somewhat confusing when you are at the project level, you have project settings; and when you are at the workspace level, you have user settings. The user settings of these tabs- if you need to use environment variables, this is where you would set them for the user. For the project settings, then you have also all these tabs available and this is where you will have also in the runtime engine, you can select the engine version and add the third-party editors In advance tab, you can set environment variables, and strangely, if you want to delete the project, it's in the delete project tab. So this is a- this choice of UI is specific to CML, so you want to project that's where you would go to project settings. For teams those are the tabs. You provide the team description, you manage the team members and then display the outbounf. SSH keys. So when you want to add a member to your team, you choose the access level on that member. For the site administration-- so that's another menu--you have- you have all these tabs. And what's interesting is the usage gives you a graphical view of the usage of the the workspace, but does allow you- allows the admin to configure to view GPU and memory quotas for users of admin workspace. So that's interesting. Can also look at the models, manage the users, choose the runtime engine, manage the data connections, and also some security settings, general settings. You can manage your applied Machine learning prototypes entries.

So what- we talked about runtimes, if you were a CDSW user, maybe you were using engines and you wonder what is the difference between the two. Difference is a runtime is the vector of three characteristics:. the editor, the kernel, and the edition, and the edition can be either Standard or NVIDIA GPU. So it's one

combination as opposed to engines that were- that grouped all the combinations. So there were bigger containers with things that were not used most of the time. This is a leaner and more efficient. So when you want to create a new session, provide the name, and then you have to choose those three characteristics. What will be your editor? Whether or not you're going to use the Standard library or the NVIDIA GPU library and the resource profile, including GPUs eventually. So this will specify Docker container that is going to be built for this session. So if you go to the runtime catalog, you can see all the runtimes available, so all the combinations available, but the difference between the runtime and the engines, the legacy engines in CDSW: a runtime is a small image contains a single vector of editor, kernel, and libraries. This is what you should be using moving forward, because the difference is that the legacy engine contains four Engine interpreters, whereas the runtime contains only one.

During this class, you will be doing exercises. So those exercises are based on the fictitious company called Le DuoCar or DuoCar for short. So it's a fictitious company. And the business of DuoCar is the same business as Uber and Lyft; it's a ride sharing company. And the data you will get is data based on its first operations in the States, in Fargo, North Dakota. DuoCar has several services. So that will be a categorical variable that we will use. and the data we'll use is centered around rides and rides has a driver and a rider; has a route and can have reviews. That's the main datasets. External data sets are demographics, and your job as a data scientist in DuoCar is to discover and communicate insights to help you develop data products. So what DuoCar as a business is concerned about is these bullet points. But we will try to do in our models is to predict the duration of a ride as a function: the distance, and then try to predict whether a given ride will have a five star rating. Of course, companies like Uber have a very powerful data science team that provides a lot of machine learning models and also contribute a lot to open source projects. As many data science team. The data science team in DuoCar is a mix of Python and R data scientists. But since the business grows, the leadership has decided to buy CDP and install CML. So some data will be stored in S3. Some data will be stored in DFS. The tools for processing analytic workloads are Spark, Hive, and Impala, and for business users, they have Hue and for data centers, they have CML as a tool to work on machine learning models.

What you will do in this exercise is log into CDP. View the Cloudera Machine Learning workspaces, create a CML project, create a new session, delete a CML project. First, you were asked to log on to the CDP. Okay, I'm already connected. I'm using one. Then you had to click on machine Learning, data service, and click on a workspace where you would see several projects depending on your progress. Next, you can check the Learning Hub. In the Learning Hub, you will find three tabs: on the first one, you will find blog posts about machine learning and data science; then on the second tab, you'll find research reports, those are provided by the team. at first forward Labs. They talk about cutting edge machine learning, and they're not as light reading as the blog posts but they're interesting nonetheless. And also you have access to documentation. So

the Learning Hub is a great resource for learning how to use machine learning techniques. Next, you were asked to go to the site administration, but as a simple user, you cannot do that. So I am logged. in as an admin on another window which allows me now to see the administration option in the menu and I get all the tabs. So this is the other view of the workspace. How many users do we have? Some. We'll likely find more users Teams- we don't have teams. This is where you see the usage. Okay, so this is a very useful tab. You consider usage in terms of in terms of CPU memory, GPU, you see the runs, and lags of Kubernetes concept. It measures the latency between needing resources and getting the resources. Then you can choose to set quotas for CPU memory and GPUs on users. For the time being, we don't have models. We can see the runtimes available. Our data connections: so we do have two data connections. We have a connections to a Hive Virtual Warehouse and we have the connections to our Spark Data Lake. Come Security settings; applied machine learning models that are available, and this is also where you can add a new one. Learning Hub with all the description of the content. Some general settings. and access to support. Going back to being a simple dev now, and I can go to the runtime catalog and see all the combinations of runtimes available for me. So you see that the addition can be standard or in the GPU. And you have all the supported Python versions. Note that it does with Python 3.7. We have the same for R and Scala Next, we can look at the Applied machine Learning prototypes. So AMPs for short, and you see all the AMPs that are available. So again, those AMPs allow you to quickly discover a machine learning technique or a machine learning framework like Evidently or Streamlit and understand how they work very quickly to add them to your set of tools. You can look at the User Settings you have the outbound SSH key is for connecting to external resources, for example, GitHub. So what I'm going to do now is create a new project. To create a new project that click on New Project. I put in the name. I keep the project visibly private. I will use the template initial setup. I choose the Python and click on Create Project. Now I can create a session which I will name First session. I would choose JupyterLab as the editor and that's it. I start the session. So I get some messages from Kubernetes. I get a panel to show me how to create Spark session. I will not need this now So, I'm going to open this notebook and that is this. And it shows me the content of the notebook so I can go back to the project by clicking on Project and look at my sessions, so my first session is running. Can delete the project by going to project settings, selecting the project and confirming. I did it in the project, and that is the end of the exercise.

First, we talk about the Workbench. The Workbench is an editor that allows you to edit your scripts and run them and see the results on the right hand side of the CML window. There is also a terminal which you can use. How do you work with a- with the Workbench? You create a file, or you upload a file. Then for the the session, you need to specify what kind of engine profile or runtime profile that you will use. Select the language, and then you can run the code from the the lines in your file or a session prompt, or you can also open a

terminal and run commands. And when you're done, you can stop the session, or it will automatically timeout according to a setting of the environment. So this is how you start a new session. It's a good practice to provide a name for the session because by default the name is untitled session. And when you have more than one then it becomes confusing. So you provide a name for the session. A good practice also is to use the name of the file that you're running. It's less ambiguous that way. So you need to specify the editor that you will use, whether or not you're going to use Spark. You see that there's a switch. So Spark is not a mandate for CML. It's something that you can use or if you're not interested in distributed processing because you're processing small data and then you don't need Spark. And you specify the source profile. Here we do using a very small kernel, but sometimes you can get a bigger kernel or even GPUs, in which case you would need to change the edition from Standard to NVIDIA. So you can select the code in the editor and press Ctrl+Enter, or cmd+Enter in macOS to run the lines or choose them from the menu: run lines. Or you can run all the notebook by clicking on the arrow icon at the top of the screen. The workbench supports code autocomplete for Python and R using the tab to activate the feature. Your project files will be stored to a persistent storage that will be in the path shown here. So it's /var/lib/cdsw/current/ \ projects/<your project name>. CML can be configured to work with third-party editors So browser-based IDEs such as Jupyter or local IDEs that run on your machines. The configuration for an IDE depends on which type of editor you want to use. You can only edit and run code interactively with the IDEs, and tasks such as creating a project or deploying a model require the CML web UI or the API. If you want to interact with the system, that would be the system in the session on the container that you specified, you can prefix a shell command with a bang. You can access the logs of the different machines that you're using by going to the Logs tab. So you see the engine. In this example, you see there are three machines involved in the processing one called Engine and three Spark executors, and you can choose which log you want to see. In the sessions menu you have a tabular representation of the sessions on your project. You can drill down in one by clicking on one session, so that will open the session, or you can create a new session, or you can or you can delete sessions.

Next, we're going to look at how we can use Git with CML. CML is designed to be used with Git, so you can create a project by cloning a Git repository and then you can perform Git operations with your remote repository. You have full access to the git on the command line that means that you once you have created your project with git, you interact with your git repository via the command line, either at the bottom right hand side of the CML screen or by opening a terminal. If you're not familiar with Git, we have a dedicated git course called just in of Git On-Demand To start using git, you can create a project by cloning a repository. You will need the public SSH key for your CML account. The SSH key is available in your user settings, but once you get to this tab, copy the CML key and then you can add it to your GitHub account like you would

any SSH key. CML accepts two types of git URLs: you can use https:// URLs or SSH URLs. And once the your project has been initialized with git, you can, from the terminal or from the session, use all the git commands that you would in a normal setting. And this allows you to work in teams on a machine learning project using CML and Git. You use pulling and pushing commands like you would in a regular project.

Next, we're going to talk about embedded web applications. One of the problems that has been identified early with machine learning is that organization overlooked a lot of the tasks involved in selling a model. This has been documented in a Google paper, and that paper is famous for this diagram. You see that organizations focused on the small ML rectangle and often overlook all the activities that are involved in solving a model. CML provides a means to streamline all those activities. And as part of this, provides an application feature that allows data scientists to deliver the machine learning models to end users. And that involves usually a UI. So applications are long running web applications that make calls to deployed models. It's the last mile. So you as data scientist, you created a machine learning model. And for business users to use that model, you provide a UI that makes a call to your model and provides the feedback from your model to the user. Applications are part of the project and they do not

So once they are deployed/ran, then if you want to stop that, you have to do this manually. It can be interactive or non-interactive. It can provide a dashboard of visualizations, or they can pull a model and provide the prediction. Of course, you need to test your application like you would anything but this point through to a page in the documentation that will guide you through this activity. Creating an application requires you to be uniform so you need to provide a name and that name needs to be unique. The subdomain which will be used for the construct of the URL, description, the script that you're running, And that script needs to either use one of those two ports, the kernel and computing resources. And if you are using environment variable, then you need to specify the name of and value for those variables. Applications can be accessed via the two ports described here. Any user can access an application to through this port. If you want to make your application public, then you need to do several things. As an admin user, you need to check the flag that allows application to be configured with authenticated access. When you create the application, you need to also enable the same authenticated access or for your existing users, you need to switch the flag. If you want to prevent users from creating public applications, you need to go to If you want to prevent users from creating public applications, you need to go to

Next, we're going to talk about applied machine learning prototypes or AMPs for short. AMPs are a means for Cloudera- enable our users to get faster to the results-- shorten the time or duration between the idea and the result. By prototyping a machine learning techniques and libraries so that it's easy for a user to be inspired by the working example. This is another view of the

Google paper on the hidden cost of machine learning. It usually takes months to create a machine learning model and put it in production. The idea is to cut this duration to weeks instead of months by using AMPs. AMPs capture what makes a library or machine learning technique tick. And so it shows you are working example that you can use to build your own successful applications. Often those AMPs come from our first forward lab entity as a demonstrator for a given machine learning tool or technique. And that machine learning tool or technique is also analyzed and documented in a corresponding paper. But we have dozens of AMPs to choose from, so you click on one. So you have to select the AMPs menu on the left hand side navigation pane and then click on the AMP that you want to use and it will start installing a project using a method data file that describes all the steps that you see here. So creating an AMP is a matter of writing this metadata file. This activity is also a explained in the documentation if you follow the corresponding URL and it's not a difficult thing to do. We used an AMP to package our exercises for the course.

In this video, I'm going to walk you through the solution of the Streamlit on CML exercise. You start from the CDP landing page. Click on the machine learning icon. Click on your workspace, click on the AMPs. Select Streamlit. Next, we can have a look at the project on GitHub, and we can look at the project metadata file where we can see that the set up of the AMP is made up of two tasks: one to install the dependencies and one for the Streamlit AMP. Let's go back to the project and configure the project. I do not need Spark for this application but just click on launch project and this will start the setup of the AMP and execute the two steps we saw in the YAML file. The first one being the installation of the requirements. This is likely going to take 5 minutes, so I'm going to pause the recording Our AMP has been set up, so let's go and click on overview. We see that we are files now in our project. We have no models yet and the project creation was successful. Let's do the details of this. Click on the logs, and here I can see the command line that starts the Streamlit application. I go back to the other view and view the status page and look at the details for step 2. Look at the logs and get to the same point where I see the command line that sends the application using the CDSW_APP_PORT. Let's go back to the files and open up the app.py. What we see here is Streamlit being used, so it's imported and then there are some codes to Streamlit to display titles, markdown, and a joint plot. What we can do is have a look at the application, and this is the execution of this app.py. You see the text, the table, and the joint plot. And let's go ahead and modify some of the items in the Streamlit application. This is the the text I am going to modify this. So I need to open the session, give it a name, start the session. This is from Spark users. I'm not using Spark so I don't need this. Let's see if I modify some of the texts here or if I had- "quite misleading." And look and open this and refresh the page. And now I can see dynamically that I have changed the text. Let's go back here. I see that I have a joined plot. I look at the documentation for joint plot. I see all the kinds of plots I can use, but currently I'm using KDE. I'm going to switch to scatter. Likewise, I'm going to open this. So now I'm

seeing KDE and if I refresh the page I see scatter display. Okay. There was let's look at the sessions in the session tab. We see two sessions: one that was terminated with success and that was part of the setup and that the one that installed dependencies and we can look at the logs of this. So this I see all the dependencies being installed, and this is my session that is currently running to modify the code. So now I'm done with this demo, I'm going to delete the project for this. I go to project settings, you click on Delete Project, click on Delete Project, and confirm. And this wraps up my work through the solution of Streamlit on CML exercise.

But what is SDX? SDX is the layer that is core to CDP. It contains all the security and data governance information. It allows bursting to the cloud because of this. Usually when you burst to the cloud, you're going to- you have something like SDX, you're going to make a sacrifice on security or data governance. And most probably you're not going to burst the cloud. But with SDX, bursting through the cloud becomes a reality. All our clusters are plugged in as a whole in CDP. A key component inside Cloudera SDX is the shared data catalog. It gives you at the same time information about the data and the metadata of your tables. And it's a precious tool for data scientists, because it answers all the questions that are listed here. SDX is made of several Apache software tools like for Authorization and Audit, it's Ranger; for data lineage, it's Atlas. And then, as I did, some value added tools like data catalog, application manager and workload manager. Apache Ranger is the open source application to manage security policies and also track the access to the resources. Atlas is also an open source application for managing metadata. It's a graph oriented database with a semantic layer for data governance, and this graph oriented database is automatically populated by Kafka topics sent from the various services that manipulate data, so that the graph database of Atlas is always up to date. . . . Data Catalog is a service from CDP that is- that are state oriented. So it gives you a unique consolidated view of your assets, including data and metadata. It also contains features such as profilers, which are basically Spark jobs with regular expressions, searching for items that will help you find and tag potentially sensitive data.

There's the Data Catalog overview. The scope of the data catalog is an environment, and within an environment, you will find the data lake for storage and workloads or compute. For the profilers to run, they use dedicated workload resources. To get to the data catalog, you click on the Data Catalog icon on the home page of your CDP. It gives you a view of- for a specific data lake and access to Atlas and Ranger UI, so you can search for your data assets using several filters. When you select-- what I said, for instance, Hive table, the statistical profiler will provide you with statistics for each column. Then you get also access to the data lineage that comes from Atlas and also metadata about your table, such as who created the table, how it was created, and details of access to the data and metadata. So in this screenshot we can see that for the ww_customers Hive table, we looking at the audit tab, and we see that the table was created by a user and read by the same user, and then there was a

metadata operation from user hive and another one.

Authorizations are managed by Ranger; prerequisite for authorization is authentication. If you don't have authentication, then you cannot do authorizations. So one perquisite for Ranger is that your cluster is kerberized, and Ranger has a plugin architecture that allows Ranger to control the several services. So you see some of those services on the landing page of Ranger. And authorizations are specified by policies and policies use the concept of roles which are defined inside Ranger and groups and users which can be synchronized to your LDAP policy will identify who can use the resource or perform specific actions. So they are resource-based policies and they are also attribute or tag-based policies. So the tag they come from Atlas, they are synchronized. So in Atlas they are called classifications. Once the classification is created in a class, a daemon will link the classifications to a Ranger tag and then you can leverage this tag to define authorizations based on this tag across several services. So this is a view of the Hadoop SQL plugin, and you see a tabular representation of all the policies. When you drill down on one of those policies, you will see that you have four sections to express the policy on the scope that is defined. So above this section there is a scope and identification banner for the policy, and then there are four sections that are grouped by allow and deny. And for allow, you have to exclude from allow, and for deny you have exclude from deny. Here we see that there is no exclusion from allow. And then you see that the role admins and this user I want to look up in plain admin of all the permissions on whatever is defined in the scope of this policy. The overview provide you with some information on the policies. There are also policies that mask data. The different access to data you do to deny access to the data, but you deny access to seeing all the data. Here we see that the street address has been redacted and the password has been hashed, and you define those the masking policies similarly to what you do for the access policies. By defining the scope of the policy and then identifying the role groups and users and the way you're going to mask the columns. Those are the options for masking. On top of the masking options, you have a custom one which allows you to define in a JavaScript an expressions that will be executed when the masking policy will be triggered. This is the evaluation flow of Ranger. So what Ranger tries to do first is to deny access to the resource by scanning all the policies. Finding a policy that matches the scope of the resource that is under consideration, and then evaluating whether the policy is yes or no. If the policy is denied access, then the workflow stops. If Ranger cannot find a policy that explicitly denies access, it will try to find a policy that explicitly allows access. So if it finds a rule that explicitly allows access, it will grant access; if it cannot find either a policy that denies access or grants access, depending on the plugin, it will deny your pass-through. So if it's Hadoop SQL, it will deny access to the resource. If it's HDFS for instance, it will pass-through. Pass-through means that it will let HDFS and its own set of authorizations, so using the permissions in the (unintelligible), it will let HDFS decide whether the user should have access or not. All this a role triggering activity can be packed in the Ranger audit page. Every time a policy is exam-

ined by Ranger, provided you switch the flag, Ranger will log the results of the policy in a large database. Auditing provides you with a multi-criteria search engine on this database. Here is a screenshot of the this page, and you can see that the policy 80 was examined and that joe_analyst was denied the access to one column. joe_analyst tried to get to see the zip code, insurance I.D., and blood type from ww_customers and was denied access because of policy 80. If we want it, we could click on the 80 which is a link and see what was the reason.

Lineage is the job of Atlas. Atlas comes with some entities that are defined for data science, such as the models, so that you can track lineage of your models in Atlas. As I said before, Atlas is at the core a graph oriented database. You can see a subset of the graph centered around each entity. So here it's the claim_savings. It's Hive table, and you can see what comes before all the lineage and what comes after the entity, which is called the impact. And you can navigate using this graph. So it's a subset of the graph that is centered around the entity, but it's not to overwhelm your display. That display can be controlled by several icons. And you can click on one entity to see the corresponding key-value pairs for this entity. So for a graph oriented database, each entity can hold a dictionary of key-value pairs. So that's what you're seeing in this panel. And you can use this to navigate by clicking on the GUID of the entity to navigate to this entity or panel. So you can choose to hide the processes between the entities if you want, and choose the depth of how many assets you want before or after. So Atlas is also plugged into Ranger for security. So there are policies by default for the admin. The rangertagsync, the rangerlookup, the public, and the users.

In this video, I will walk you through the solution of the Data Access, Audit, and Mask exercise. First, we access the Data Warehouse service, and we click on Hue. We will make sure we are using the Hive editor. And let's look at the tables available. So we select duocar, and we are going to execute the query. For instance, SELECT *, and I can use the autocompletion, FROM drivers and as always with big data, it is a good idea to add a limit. I get the results for my driver. Next, I'm going to create a table. So since we are all sharing the same databases, I'm using a specific name and we have to wait for the test station to start. Okay, we got successful results and we can confirm this with a SELECT statement. So let's go back to the Data Warehouse tab and click on the main menu icon. Click on Data catalog. I can select my table and drill down. I can view the table lineage and look at the schema. Do we have classifications? We'll see them, and that's the end of the exercise.

In this chapter, we are going to study data visualization in CML. To visualization is key in two stages of your data journey: first, to quickly and easily explore data, and secondly, to explain insights to your business stakeholders. Visualizations are all the more critical for big data as sifting through rows of columns is no longer a viable option. We need a very condensed representation of all this data, and that's what visualization gives us. The data visualization tool is available in two data services in CDW, in the data warehouse data service, and

in CML. In CML, the tool is an application that can be accessed via the data option in your menu. It comes with sample data and ready-made dashboards built on those samples to get you started. Here, we see the two stages in the life-cycle of a machine learning model where visualization is critical. It's critical for the exploratory data analysis, and it's critical to build interactive dashboards and visualize model predictions. As such, it's a collaboration tool between two teams: the data science teams and the business stakeholders. Those are the requirements you want to have from a visualization tool. You want the tool to be fast and provide you with self-service data exploration. You want the tool to support cross-team collaboration and to provide proper data analytical automation. Of course, that is the case of the Cloudera visualization tool. So the benefits of this tool is that it's integrated with CML. Other than that, it's similar to other tools, but the big benefit comes from the integration in CDP. As such, it allows access to all the CDP sources, such as Impala, Hive, and machine learning models.

But what are the concepts implemented into this tool? First, this is some of the features of this tool. So we have over 32 visual types. The tool provides also a logical semantic layer and can power custom interactive dashboards and applications ... So now for the concepts. So the concepts are the following: connections, datasets, visuals, dashboards, and applications. I will go through all of them. Connections. So this is the start of your workflow. This is where you connect to databases or data sources. In the context of CML, then you can connect to Hive Data warehouse. The dataset is the next step in your workflow. You define a semantic layer on top of your data tables and views. Basically, you define a query and which will have metrics and dimensions that you will use to define your visuals. Here are the available visuals, including- so you have the usual pie charts, histograms, and heatmaps; but you also have maps, wordclouds. So you should be able to cover all your visualization needs with these visuals. You group your visuals into dashboards and those dashboards can be parameterized with filters so that they are dynamic and then you bundle up your dashboard into an application to distribute your dashboard to stakeholders.

I have switched to a training environment to look at the data visualization tool. So when you're inside a project, you click on data here, and this is what you see when you land in the tool. You see counters of the different connections, datasets, and apps and dashboards that you have built, your recent connections. You see sample dashboards created from sample dataset that come with the tool to get you started. Then samples and you see your connections also for building your own dashboards. So this is the general workflow for creating dashboards. You create your data connection, you define the dataset using your data connection, you create a dashboard based on your dataset, that dashboard contains visuals and you bundle your dashboard or your dashboards in an application that you can share with business users.

In this video, I am going to go through the solution of the Visualize Duocar Data exercise. So first, we create a new project to give it a specific name because we

are all using the same workspace. So now I'm in my project; I can click on data and launch the data application. This is the home page of the data application. I can see the counters at the top of the screen. Recent dashboards. Recent connections. I am going to click on datasets here. So we have sample datasets, and those are the datasets that are used in the sample dashboards, but we are going to create our own dashboard. And for this, I am going to connect with our own database. Oh, so this error is to be expected. The data application needs a workload password to access the data warehouse. So we are going to create a workload based world. For this, we need to go back to the main page, click on our user, look at the profile, and set the workload password here. Okay, my workload password is set. Let me go back to the workspaces, go to the user settings, click on environment variables and provide my workload password. I need to restart the application. The I click on applications and I would say restart application. Okay, so now we should be good. Let's check. So, clicking on datasets. Aha! No more error message. I'm creating a new dataset. I will call "Duocar Joined," and I selecte the Duocar database and the joined table. Click on create. Okay. So now I can see my my dataset, and I can visualize the data model. So it's a very simple one. It's based one table called joined. I can ask the tool to show me the data and yes, it shows me the data. Well, now I'm going to create visuals and ask for a new dashboard. So the title will be Ride Dashboard. I'm going to create a new visual based on a bar chart. So this dimensional will be service, and I will drop this on the x axis, and I drop Record Count on the y axis. I can now ask to refresh the visuals. So the first time you make a connection to the data, the data warehouse, (unintelligible) click on the main menu, data warehouse and it is starting. That's why- taking time, and now it is running. And my visuals have been refreshed. Let's do another visual. And let's use a pie and let's use star_rating in the dimensions and Record Count into measures. Let's enter the title Star rating, and let's refresh the visual. Okay so to move things around, let's add a new visual using packed bubbles and rider_student as the dimension and Record Count as the measure. I'm going to resize these. And all these student riders. Okay, starting to look good. I'm going to add another visual. This time it's going to be a calendar heat map. I'm going to use the date_time as the date and again the Record Count as the measure. Going to give this a title also, going to call it "Ride Heatmap," so we have an information icon. If you click on it, we see that only 5000 roles have been fetched. Okay, so we are going to click on Settings, the visuals. And we're going to switch this to 50,000 ... now we have no more warning and more data visualized. Let's add a filter to this dashboard. Now we have a distance filter at the top and another one and now we added rider_student. ... Okay, let's save this dashboard. We can go to view, and we can use the filter, the distance. Maybe we want to see whether students do long distance rides, and we see that it is not the case, They take small rides? Yes, indeed. Can we look at the day of the week where there are more rides for students? It seems that it's a Thursday. Thursday seems to be a popular day for students to ride In this menu, I can export the dashboard as a file, and if I open the file with a text editor, I see that JSON representation.

In this video I'm going to present how to use MLFlow experiments with CML. So why do we need 'experiments'? Well, because the development of a model is an iterative process. Each iteration can be an experiment where you try a different combination of data algorithm and AI parmeters. So you create multiple versions of your models, and you need to keep track of the input and output of your training. So MLFlow is an open source platform to manage the machine learning lifecycle. It includes four major features: experimentation, the one that we will study; reproducibility; deployment; and a central modal registry. Reproducibility and deployment are already available in CML, so we don't need MLFLow for this, and the model of registry is being added and will be soon available as a feature in CML. For the time being, we will focus on the first item. To use MLFlow, you need to add the MLFlow library into your requirements.txt file, and then you need to instrument your code, which is called to MLFlow that starts with the import of the MLFlow library and then your run is encapsulated in a with mlflow.start_run statement in which you log the parameters mlflow.log_param and at the end of the run you log the metrics that you get with the mlflow.log_metrics. Additionally, you can log the model, calling mlflow.sklearn.log_model. when you run your code like this, then you can go to the MLFlow UI, which is an application in CML, and you can drill down on your runs and copy boilerplate code to serve your model. In the next exercise, you will be using MLFlow inside CML.

In this video, I will walk you through the solution of the Using MLFLow Experiments in CML exercise. First, we go to our workspace and here we click on AMPs. There's an MLFlow tracking AMP that we're going to use. Let's configure the project. We need to enable Spark and then launch the project. So this is going to take a lot of time, so I'm going to pause the video. Okay, the setup is completed. We can go to the applications menu and then we have the MLFlow UI and click on this. And it takes us to the MLFlow. We see that we have two runs, and we can click on one, and we can see the parameters and the metrics. And we can see the code that allows to serve the model in two ways using a Pandas dataframe or Spark dataframe. And for the spark dataframe, you wrap your calls to the module in a UDF because that model was created with- in Python. So the model has been pkl'ed, and that's the way that you can use a mlruns generated models within Spark by wrapping them into a UDF. We can go and see the other run. So this is a clustering exercise and I think we get the metrics and we can see the code to call the models either using Pandas dataframe or in- with distributed processing in Spark. The next we are going to run a job. So we are going to create a new job, and we're going to use the KNeighbors algorithm. But we need to provide the value for K. Okay, so that would be our hyperparameter. We will be using the workbench as the editor. And we're going to use Python 3.9, and then we are going to run the job. The job is created, and I can ask to run it The job is successful. So if I go back to the MLFlow UI. I have a new run. This is the parameter I have provided, and this is the result I get. I still have the same snippets of code to copy and paste. And that is the end of this exercise.

In this video, I'm going to give you an overview of Spark. So Spark is a general purpose, large-scale, data processing engine. It's a very popular open source project, and one of its appeal to organizations is that it offers a wide range of different data processing workloads. It can be your one stop shop for all your data processing. It's very open, so it can run on many types of clusters. In CDP, we use Apache Hadoop Yarn and Kubernetes. Spark on Kubernetes is available in the CDE and CML data services. Since data services are built on top of a Kubernetes enabled layer, then we run Apache Spark on Kubernetes to benefit from the auto-scaling features. So a Spark application has a master/worker architecture. In Spark, this translates to a driver/executors architecture. On a young cluster, an application master will be assigned to the Spark application. And if we enable dynamic resource allocation then executors can be launched and terminated during the life of an application. What you will be using in the exercises is Spark on Kubernetes because of CML and therefore, in the default behavior on Spark on Kubernetes is dynamic resource allocation.

One of the benefits of Spark, and I mentioned this before, is that you can do all your processing using the same language: can do SQL processing, you can do machine learning, can do streaming, and graph oriented processing also. What we'll see in this course is the first two libraries. First we'll study Spark SQl for data engineering purposes and then we'll look at the content of the machine learning library. Spark Streaming content is also available elsewhere, but for graph oriented processing, Cloudera does not support GraphX. Spark SQL is a Spark library for working with structured data. So what do you do if you have unstructured data? Well, you can use regular expressions to make it structured and then you can use data frames and Spark SQL. Spark SQL provides a data frame API, SQL query engine, and a catalyst optimizer that optimizes your queries. And what we'll be using for this course is a Spark 3.2 or 3.3. So we'll benefit from the latest features of Spark. In Spark SQl, the main data abstraction is a data frame, so this kind of representation comes from R first and then was also used in Pandas data frames and was added to Spark with Spark 1.3. So underneath a data frame is an RDD and what the data frame brings to the RDD is column names and column types. A data frame is a collection of all objects and needs to have a schema. The need for data frames to have a schema will be the source of a an exception in the otherwise lazy execution of Spark. Data frames are immutable, because we are doing this super deep processing using a show-nothing module, and therefore we cannot have objects that mutate because otherwise it would create problems for parallel execution. So data frames are immutable. What you do is you transform a data frame into a new data frame.

Selecting a format for a data set involves several considerations, but for a HDFS, the characteristics you want from a format, you want the format to be splittable, you want the format to be compressed, and you want the format to document the schema of the data it contains. Only three formats fill those requirements ORC, Parquet, and AVRO. Of course, when you ingest data, usually the format is a text-based format. The text file formats are human-readable, they are

easy to read, and that's good for debugging, but they're also very inefficient at large scale. It's okay to ingest data in a text file format, but one of the next steps would be to store the data in a better format. Such as, ORC, AVRO, or Parquet. Parquet is a very popular format; it's good for batch operations like ORC. AVRO is more for streaming purposes. The reason for this is that Parquet and ORC both share columnar file format and that provides a lot of compression opportunities. But this cannot work when you're receiving data or records one by one, such as is the case in a streaming context. For that context, AVRO is better because it's doing a row compression. Parquet and ORC are in the same ballpark for compression and overall performance. ORC is mandatory for Hive managed tables. When you're using Hive managed tables, you don't have the choice, but otherwise Parquet is really popular.

Another cool feature of Spark is that it's possible to write Spark code using four different languages: Scala, Java, Python, and R. R is the latest addition to the languages that Spark can use. It was added in Spark 1.6. The way that the SparkR was designed did not please the people from the R community, because it was not compatible with the tidyverse school of programing. So the people from the RStudio community developed another version of SparkR called SparklyR which this term is compatible with dplyr. For this course we will cover PySpark.

PySpark is the Spark's Python API. You have full access to the Spark API. There's an interactive shell application. For Spark, it will be PySpark3 In PySpark, so you have the Spark SQL library that allows you to work with data frames with methods such as select() filter, where(), orderBy(), sort() withColumn(), groupBy(), etc. And it's because those methods have meaningful names that the optimizer, that is called Catalyst, can leverage those names to perform optimizations. That's the big difference between data frames and RDDs. With RRDs, you process RDDs with generic functions like map or flat map. And there's no way a optimizer can guess what's going to happen and perform optimizations. This machine learning library contains, besides the usual machine learning algorithms, utilities such as feature transformers. There are ways that you can also work with a Pandas dataframe and Scikit-learn by converting your Spark data frame to Pandas DataFrame and then leveraging all the ecosystem of Python such as Seaborn, Matplotlib, Scikit-learn, NumPy, Pandas. But you need to be aware that when you're doing this, you will be working on a single machine on the driver. So if you know what you're doing and the data that you are processing is small enough to fit on one machine, then you can switch back and forth between Pandas DataFrame and Spark data frames.

So there are two main types of dataframe operations. There are transformations. A transformation will create a dataframe from an existing dataframe, and then there are actions that trigger the materialization of the dataframe and thus the execution of all the logical graph that leads to that dataframe. We define a job as a sequence of transformation followed by an action. So here are some Spark code line

and line 9 is an action. So that means that the file will be read when line 9 will be executed. Before that all those variables ABCDEFGH, are registered in the driver memory together with their lineage, but nothing happens on the executors. It's only when you execute or write h.write.csv(...) that write is an action. The driver needs to materialize H and therefore needs to materialize all the previous data frames. So this way of processing is called lazy execution. You could write the previous code using this notation so you will have less intermediate dataframe and the footprint in the driver memory will be less. That's one benefit. If you want to reuse some of those intermediate dataframes and maybe use caching or improve the code clarity, it would be a good idea to use an intermediate dataframe. So it's a tradeoff. And here's an example: we define H and then we say persist on line 9. And the Spark interpreter, which is line 9, what happens is that the dataframe whose name is H is earmarked to be not evicted the next time it will be materialized in memory. In other words, it's not in the cache yet. Then we come to line 10, and line 10 forces the materialization of H because it's going to write H. So after line 10, H is in memory, and then H is collected from memory on line 11. The takeaway from this is that persistence is lazy also.

So there's no distributed processing. If your file is not partitioned when you are using data frames, the catalyst optimizer will automatically choose the input splits using an algorithm based on the number and size of files. In other words, Catalyst knows the best number of partitions. And the number of partitions is critical to the performance of your Spark job. That's another interesting feature of Catalyst. It knows that the right number of partitions. In this example, we are using what we call "narrow operations." So narrow operation do not involve the transfer of records between machines. It's pure parallel processing that happens in the memory of each executor. So this is where Spark shines and that can be done for transformations such as read, select, filter. Those do not require the data to be shuffled. So this is narrow operations. Unfortunately, the transformations we need to do are usually more involved than just doing select and filter. We do like to do groupBys, joins, and things like this. And those operations are wide because they force the redistribution of records. This is the problem number one of distributed processing. It's called the shuffle. And it's a problem because you're going to use the network and the network is a shared resource that is scarce, and therefore you can become the noisy neighbor for your colleagues, or you can be the victim of your noisy neighbor colleagues. Another problem that occurs for this transformation is that the initial number of partitions was correctly identified by Catalyst. But until Spark Adaptive Query Execution, the number of going partitions after the shuffle was unknown and therefore there was a default value which is 200. And of course 200 is never the right number for your partitions: it's a magic number that has been solved with Spark Adaptive Query Execution. A wide operation can be a groupby(), a join() a distinct(). Wide operations are expensive. Narrow operations are not expensive, but wide operations are expensive. But they cannot be avoided. But the impact can be mitigated by using best practices in distributed processing

and Spark does that in the Catalyst Optimizer. It already includes some of the best practices of distributed processing. The second problem of distributed processing can be also seen here. When you do a groupBy(), you groupBy() a column that that has some business meaning for you. And usually business data is not evenly distributed. Therefore, whereas at the beginning of the processing it was H partitioning. So it was random. Now when you're doing groupBy(), you include your business logic and the skew in your business logic. Skew means that your data is not evenly distributed, and that's the the curse of all business data: you have more clients in this city than this other city. You have more business on this day than this other day of the week, and so on and so forth. Here when we represent this with processing, we always use evenly sized rectangles, but in reality those rectangles can be very different- of very different sizes, and sometimes the skew can be extreme and it can overwhelm the executor that will receive the skewed partition. The motivation for using distributed processing is that you have data that is larger than what can fit on a single machine. So if you send all this data to a single machine, it will be overwhelmed. And that can occur when your data is heavily skewed. So that's the second problem of distributed processing. And Spark3 with the Adaptive Query Engine introduced some ways to deal with skew. Last, we do an orderBy(), and that's the third problem of distributed processing. Since we are using a shell-nothing model, meaning that executives do not talk to each other, when we want to compare two values we need them to be on the same machine. Happens for orderBy(), but also for distinct(). So those operations are expensive. And you cannot orderBy billions of records, you can do orderBy()s of reasonable numbers like top ten or even top hundred and bottom, ten and bottom hundred, but ordering the whole dataset of billions of records-- that's not going to work. Last we write our data frame to a file. And we see in this diagram that there are four orderBy()s. We can assume there are four executors involved in the processing in parallel. And they're right. Each executor writes its file in the destination folder. The consequence of this is that the order- if you had an implicit order in your file to start with, that order was destroyed the first time by H partitioning and then it was further destroyed by the the fact that each executor will dump its file when it's finished, and you cannot control the other in which the executors will complete their transformations. So the take-away from this is that if the implicit order in your file mattered to you, then you have to make it explicit.

In this video, I'm going to go through the solution of Using Workbench for Lecture and Exercises exercise. I'm going to create a new project. We'll- be called Student #1 and the initial setup will come from an AMP. I need to provide a URL of the amp. This is the one, and I can click on Create Project. I'm going to Enable Spark 3.2 using Python 3.9 and then I will launch the project. As usual, this is going to take some time. Okay. The AMP has completed the setup. So now I have a project called Student #1. That's me. And I have some code files. And I can open the first one. Okay. If I want to run this code, I need a session. I know this code doesn't need to Spark. I'm going to use Python 3.9. I don't need GPUs or fancy resource provider. I can start that session. I will

provide a meaningful name to the session. Before this, I'm going to close this open, and that's for Spark. And we already have the code as the connection, so we won't be needing this. My name, my session after the file that is running. And now, I can run the code using the triangle icon and arrange the display to get more real estate. Okay, so the code ran. It's just a little world from CML. So this is just Python code, so I can use Seaborn. I can also interact with the system in the session container. Do pwd, list the files, access HDFS, list to the Python packages installed. Yea, quite a lot of them, but there's not the one I want, so I can install folium which I then use to initially look at the a map of Fargo where the action of our labs will take place. And this notebook also shows the all the features of Markdown, including LaTex expressions, bulleted lists, numbered lists, links, images, code blocks. That's it for this exercise.

In this video, I'm going to go through a quick overview of machine learning. So we use a definition of machine learning. It's a good one. So machine learning is the use of algorithms to automatically discover relationships or patterns in historical data that generalize to future data. If I remember correctly, this was the winning definition coming from an internal challenge in Cloudera. My definition came third, so I still got some prize, and my definition is machine learning is programing with data. What I mean by that is that in traditional programing you describe the logic of your processing. Sometimes it's called imperative programing. In machine learning, you prepare data and then you will launch an algorithm which has some levels called hyperparameters. But basically the algorithm is set and you launch the algorithm on your data, so the algorithm bounces on your data and gives you back prediction. But what changes your output is mainly the data that you prepare. This is a new paradigm for computing, and it allows to answer different set of questions. We would count things in conditional programing mainly. If you in banking, you are counting money. If you in telco, you're counting phone calls and so on and so forth. In machine learning we can answer a different set of questions and that opens a new world of possibilities. Like what is in this picture? Is it a cat? Is it a dog? Is it a cookie? What is the next word in this sentence? What we do, we train a model with data and then we use the model to make predictions. Here we have a a picture of a scatterplot, and this is what we are going to study next, is the the duration of a ride as a function of the distance. So it's a simple regression. We can see that already. It matches our experience of doing rides. Some vocabulary that we will use. So in this example, a distance can be called an input variable, a predictor variable, or a feature. I will use the term feature consistently and so does the Spark in my lib. So that's fortunate. Of course, a model can have more than one input variable or feature, but to start with we will use the simplest model with one feature. Duration is the output variable, can also be called the response variable or label, and we will use the term label A model typically as only one output variable. And what we try to achieve is a model like the blue line in the diagram that feeds the cloud of points and that generalizes well, and that has no bias so that the points are well distributed on either side of the blue line. The model we saw was the just right model. But if we use just a slope, it

will be underfitted. And here we have- we see in the left hand side diagram that there is a problem with short rides: they are all on the right hand side of the line. So there is a bias for short rides. So this is a quick model to make. Okay, you can use Excel to create this regression, but it will not be accurate for short rides. On the other side we have the overfitted model where we use several orders of the input variable. And then of course it goes for a lot of the black points, but it does not generalize well. So if we have longer rides then the prediction will not be very accurate. Although we strike a balance between underfitting and overfitting, what we can use is a technique called validation, whereby we use a subset of the training set to check the performance of a model. We check the performance with metrics that depend on the the type of problem that we are dealing with. For regression, for instance, it can be the R^2 measure. The R^2 is a convenient measure because its range is between zero and one, and we know that the model is better if the measure is closer to one. And the way we can explain the measure is that it's the proportion of the variance in the data that is explained by the model. So the closer to one it is, the better. At the beginning of this lesson, I said there are some levels that we could use on a set algorithm that are actually learning algorithms. Those are called hyperparameters. Basically, a machine learning algorithm is like a recipe, and a hyperparameter would be the number of chilies that you add to your recipe. so it's the same dish, but with more or less chilies. Depending on the machine learning algorithm, there will be more or less hyper- parameters to tune the behavior of the algorithm. And so your job as a data scientist is to find the best combination of hyperparameters to give you the maximum accuracy for your model. This is something that you get better at with experience, but if you don't have experience, you can still use brute force computing by doing hypergrid searches. So building a grid of different combination of hyperparameters and exploring the grid with brute force computing. In turn, you will develop an intuition for the the range of those hyperparameters. Traditionally, we distinguish two types of machine learning algorithm: supervised learning and unsupervised learning. So supervised learning is when you know the truth in the training data. So training data has a label and unsupervised learning is when the data has no label. We saw duration of the ride as a function of the distance. That was an example of supervised learning. Supervised learning is traditionally something like clustering where you get clouds of points and you trying to find a pattern in the cloud of points. So in this example, we have the different users of the Duocar service and we look at the their rides in terms of rush- hour rides per week and non-rushhour rides per week. So this is typically something that marketing would do to identify different segments of the population and perform targeted emails campaign. So within K-means clustering algorithm, which is popular for clustering, we could give one hyperparameter that would be the number of clusters we will- we would like to find. And the K-means algorithm will perform its magic and give us the corresponding results. We ask for four clusters and we have four clusters. And after that the people for marketing can come and label those cluster to distinguish the different populations. So we call the cluster number 1, infrequent riders; cluster number 2, commuters; cluster number

3, frequent riders; and cluster number 4 non-commuters. But that's the added value of the people exploiting the results of the algorithm. The algorithm gives you just four points, which would be the very centers of the four clusters.

So, what tools are used for machine learning? Well, the machine learning algorithms have been around for a long time, most of them. What has changed lately is the availability of GPU machines. But the classic algorithms have been around for a long time, but they were not available in open source packages, and now they are available. So we have the combination of availability of software and availability of hardware to run the software and to the algorithm. And we can- when we go to the Spark documentation, we see that- we want to the list of algorithm, it's there and it's the usual suspects minus several exceptions. But you see we have algorithm to perform classification, regression, recommendation, clustering, topic modeling, and then we have utilities to do feature engineering, build machine learning pipelines, perform hyperparameter tuning, and store and and load our models, as well as some other utilities to perform some statistical or metrics operations. So the library is quite comprehensive. It's comprehensive, but for neural networks and deep learning, you're going to use different tools that are not covered in this course. So the most popular ones are TensorFlow associated with Keras. For open source, You have it MXNet and on top of MXNet, you have something like Keras (unintelligible). And then you have PyTorch. So there are a few packages available to do neural networks and deep learning.

In this video, I'm going to give you a quick overview of Spark Machine Learning Library. So the Spark library for machine learning is called MLlib, and what sets this library apart from the ones that you may be familiar with: the algorithm are implemented using distributed processing. So this library scales; that's the the main difference from the other machine learning libraries. This one is meant to scale, and that's why it matters for big data. It also uses the the concept of data- frames, which we are now familiar with. The reason for this is that initially it was built on top of the first concept that was implemented in Spark: RDDs. But then it was rewritten from the ground up using dataframes to benefit from the optimizers that come with dataframes. This library contains most of the popular machine learning algorithms, both supervised and unsupervised learning, as well as some key utilities for feature engineering. Last, you can also perform hyperparameter tuning with a Spark MLlib as well as creating pipelines. Let's review some vocabulary. Your machine learning algorithm is this orange box. It can be tuned using hyperparameters, and you feed this machine learning algorithm during the training phase, which is also called the fitting phase, inputs with your features and the labels in case you're doing supervised learning. So you call fit on the machine learning algorithm object. And this gives you a machine learning model in blue, which predicts y as a function of x. Next, you will use your machine learning model to perform predictions on new inputs, and that will give you a new output. Then your output will be called y-hat as a function of x-tilde(~) For the evaluation phase, we compare y-hat, the predicted output, with the label y. And we need a machine learning evaluator object on which we call

the evaluate method. And prior to this we would have set which performance metric we want to use. There are different metrics available for different types of machine learning problems. There would be a specific lesson on the subject of metrics. The Spark machine learning library uses high level abstractions. And this is why it's very consistent. So we have three virtual classes estimators, transformers, and evaluators. Estimators will be a machine learning algorithm. The transformer class will be your machine learning model, and the evaluator class will represent a collection of model performance metrics. So if we project the theory on Spark MLlib, our training data will be held in dataframes. We will feed that to an estimator object on which we will call the method fit() to obtain a machine learning model, which will be a transformer. This transformer will be used- using test data, which will be held also in a dataframe. We'll call the method transform() on the transformer, and we will obtain a new dataframe with the predicted label. For the evaluation phase, we have the evaluator object and we feed that object to DataFrame with the predicted label and we call the method evaluate() to obtain the model performance metric. And this one can be set prior to the call to the evaluate() method. The benefit of using these high level abstractions is that we can build pipelines or sequences of those objects. And we can use those pipelines as an entity on which we can call the fit() and transform() methods. So only instances of the previous objects can be included in a pipeline. So when we call fit() on a pipeline, the result of this is a pipeline model on which we can call transform().

In this video, we're going to look at some tools and techniques to explore a data set. This is the first step of machine learning. So get familiar with your data and try to develop some intuition for the machine learning model. We'll use a mix of the techniques we already studied and visualization using Python packages such as Seaborn. First, I need a Spark session and a dataframe with all our data. For visualization purposes, I'm going to create a Pandas dataframe from our Spark dataframe using sampling, and I'm going to sample 1% of the population withReplacements=false. We're going to look at some different cases of variables. So first we're going to look at a single variable. So variables can be categorical or continuous. We'll start with categorical. In order to set a good categorical variable is the service. We have already studied the distribution of the services and we have seen this data already. What is interesting is that the result of groupBy().count() is itself a small dataframe, and small dataframes can be converted to Pandas dataframe in the driver and visualize using Pandas, which gives us a better display, but also can be fed to visualization packages such as Seaborn. We get this distribution. Alternatively, we could have used the Pandas dataframe coming from the the sampling to do a visualization. And we see that this technique gives us the same results qualitatively. So you have two techniques: either you visualize the result of a groupby().count() done in Spark and then transferred to Pandas dataframe, or you create a Pandas dataframe with a percentage of the population and use that to do visualizations. Next, we're going to look at a continuous variable such as the distance. For continuous variables we have some functions from Spark that allow us to measure the the

skewness, the kurtosis, the mean, the max, the standard deviation. So we saw that already when we studied summarization, and we can get also a feel for the distribution using approximate quantiles. But for most human beings it's easier to look at a graphic. So in this example, we used the second technique. We create a Pandas dataframe using sampling from the Spark dataframe and feed that to Seaborn to give us the distribution of the distance. And since we are using Seaborn, we can leverage the features of the package and add a rug. Or we could also create a boxplot depending on the story you want to tell. You can use either of those visualizations to make your point. Next, we're going to study the explanation of a pair of variables, and there will be three cases. First is the categorical to categorical. So that translates in duocar to the rider, student, and the rider gender. We can always do a groupby().count() and get a table like this. Since this is a small dataframe, we can put it in a Pandas dataframe and display it using Seaborn, and we can improve this by dealing with the missing values. to get a more accurate representation of the distribution of gender. The next case is categorical to continuous. So that would be, for instance, the student status and the distance. groupBy() is always available. For instance, we can look at the count, the mean, and the standard deviation. But we could use a sampling technique. And look at a strip plot, which gives us the information that non-students are taking the long rides. This is a nice visualization, but there are other visualizations that you could use and we provide you with some supplements, and one of which is this one, explore supplements. So for the same data, we could have looked at a bar plot, a point plot, the strip plot we have seen; all this, swarm plot, and there's also boxplot or violin plot. So you have different kind of visualizations available. And again, it depends on what point you want to make with your visualization. Back to our main notebook. We are going to look at the continuous to continuous exploration. So for this, we can leverage statistics with correlations and covariance. And we already saw that there was a strong correlation between distance and duration, and we can see that more clearly with a Seaborn representation on the sampled Pandas dataframe. And since we are in Seaborn, then we can ask to see the regression. And we can also increase the order of the regression to maybe get a more accurate model for the short rides. So in our case, we already had the intuition that distance and duration are strongly correlated. If you don't know the data that you're dealing with, you can use pairplot. So here we are looking at the distance, duration, and the hour of the ride. And the only candidate we see for correlation is duration and distance. That's it for this notebook on exploration.

In this video, we're going to take a peek under the hood of Spark and see what tools are available to monitor your Spark applications. I'm going to get the Spark session. And now that I have a Spark session, I have a new tab called Spark UI, which brings me to the Spark UI, where I can see the jobs and for the time being, no jobs have taken place. I can look at stages or so. But since there are no jobs, there are no stages. I can monitor the cache of Spark. I can look at the environments, including all those Spark properties that govern the behavior of Spark. I can look at my executors. So for the time being, I have a an

executo that is included with the driver. Bear in mind that we are using Spark on Kubernetes, so it's a slightly different behavior, and I have no SQL queries for the time being. So I am going to read the data from rides and I'm going to use the text version of the rides. This did not trigger distributed processing. It's just a definition that is now available in the driver memory. For the next time that I will ask to see rides, I can also ask to see the number of partitions of rides, so I need to get to the underlying RDD of the dataframe and then use the method, get new partitions. And I can also ask to see the schema of the dataframe. And the schema is the default schema. And this is why the definition of rides did not trigger any distributed processing. It's because it has the default schema. All of this occurred between the session and the Spark driver. At no time was there a Spark job launch with executors. And I can check this. And the reason for this is that we are dealing with metadata. This is the definition of the dataframe. This is the number of partitions; this is the schema. But we are not asking to see the actual rows of the dataframe When we do so, then of course, the data is located on the executors, so the executors are going to be contacted and there's going to be distributed processing. The reason for what-for which we have to wait as to do with the way Kubernetes works. Its uses dynamic allocation of executors. And when I spoke, it took more time than the timeout for executors. So it took a while and got one executor added. And now I have this job with this description that is generated by Spark, and I can drill down. And there was only one task. And I can drill down further and see the task and all its relevant metrics: so the durations, the garbage collection time. Okay. So for the time being, we have only one job. Now I'm going to do a count, and of course the count you need to access the data and it's going to trigger this, which is execution. And this time, because we have two partitions, as we saw before, there's going to be one stage where each partition will do a local count. It will be a shuffle where the intermediate counts will be shuffled and there will be another stage after the shuffle where the two intermediate counts will be consolidated. And we can see that in the Spark UI. So when I spoke too long, then the executor was removed and then it was added afterwards to do the count. And if we look at the number of jobs, so the way that Spark behaves now is because we have adaptive query execution. So we created one job to perform the local counts and then it finalized the count with another job. So one way to see that is to use the SQL tab. Or I can use this menu here. So you see the count is a query. Each takes two jobs. And here we see that we scanned the file. We perform a local aggregate. Then we shuffled indeterminate counts, and we finalized the count. This looks like this. And the reason for which it is broken in two jobs is because the adaptive query execution engine has now become the default behavior for Spark and now Spark creates a job for all the narrow operations. So the operations that are within a stage and that allows Spark to perform potential optimization for the next stage after the shuffle. Because this is a new behavior since Spark 3, we can write all data using the write() method applied to the data frame. And if we want to see the results of this. we can use it for HDFS. And we see that we have two files: part-00000 and part-00001. And that's the reason for this is that two executors were involved in the processing,

and each executor dumped its contribution when it completed the task. And you cannot control the order in which those executors will finish. So the other of your records will be shuffled once again because of this parallel processing. And note that the name of the file has no semantic value. The semantic value is at the folder level. In distributed processing files are just parts of folders and order is a challenge because of distributed processing. Okay. So next, we can look at the impact of performing repartitions. So generally I do not recommend playing with the number of partitions, but let's see the impact of this. So we do a repartition and that's just a definition; there's no processing involved. Okay. But when we launch the count, we can have a look at the Spark UI. This is one of the difficulties of, of the Spark UI when you do not label your jobs and descriptions. This is something I will show you afterwards, but we can go to the SQL tab. Our latest query was broken into three jobs: 4, 5, and 6. Maybe you can look at the DAG of this one. Yes. So this one tells the story. This is the repartition. So our two partitions are spread amongst six partitions and that accounts for one shuffle and then each partition gives its count and the six counts are shuffled across the network for the final addition. The bit that you're doing in this case is that it's going to be more efficient to do the parallel processing in stage 9. It's going to offset the time spent on repartitioning. Another method that you can use to play with the number of partitions is coalesce. Usually it's two coalesce to one so that the result of this gives you one file. And indeed, in this case, we get we get one file. Please note that this is not a good practice, because it's not scalable. That's like doing a collect on a worker. You use Spark and distributed processing because you expect your files to be larger than what a single machine can handle. It's a bit of a contradiction to do a coalesce on your files and dataframes. The better way to do this is to use a -getmerge instruction with HDFS. So this was a -getmerge: you get all the files in one given directory, merge into one file. So, you should not use the file names as a strategy to do your processing. If you do so, I suspect that you are using some Java based pattern that is going to be awkward in distributed processing. Next we we're going to look at all the Spark cache works. Let's look at the situation of the jobs. We are job Id number 7 I'm reading the CSV file, but this time I'm using header=True and inferSchema=True, and because of that I created two jobs. So the executor were asked to retrieve the header of the CSV files and also scan the files to infer the schema. We saw that already when we did the read notebook. Okay, now I have a ride, and I'm going to create an artificially large dataframe. So this is the definition. And what's the number of partition of this dataframe? It's 2. I'm creating an artificially elaborate set of transformations, which I call results. Still no processing involved. And then if I look at the number of partition, it has one partition. Okay, so now I'm going to say persist this dataframe. So if I look at the storage tab, there's nothing in it and that's because persistence is lazy. So what does this dataframe in the driver memory is earmarked to not be evicted from memory. The next time it will be materialized. I'm going to materialize the dataframe by doing a count. And if I go to the Spark UI and look at the storage tab, you can see that my dataframe has been materialized. What is interesting is the internal log of the

dataframe is its physical execution plan. So it's a bit long, but it's okay because it's internal. The number of partitions that were cached is 86, and I can drill down. And check where those partitions are stored. What else can I see? I can-can see that it's

So that's the default behavior from dataframes. It's using memory and disk, and it uses memory first and then spills to disk if it cannot store

Okay, so this took

Now let's perform the action again. And this time it's going to use the dataframe in memory. So it's much faster. And if we look at the Spark UI, the jobs themselves. Okay. Yes, this is the green dot that I was looking for. The green dot materializes the fact that the cache was used. So all these stages were skipped. Now I'm going to unpersist the dataframe and let's check the Spark UI one more time. It's gone. So persisting is lazy but unpersisting is eager--the opposite of lazy. I'm going to stop this Spark session. Next we are going to look how we can configure this Spark environment. So we build a new session, and we pass some parameters. So we want the driver memory to be 2 gigabytes. We can check that the value has been set correctly here. We can also check in the environment tab that it's also set to two gigabytes. Some configurations can be set dynamically in the middle of the code. Some configurations need to be set at the beginning when you create the session, and that is the case for the driver memory. You cannot increase the driver memory once session has started. Additionally, we can look at the default configuration for our Spark. So okay. And we don't have one. If you have properties that you want to set for all your Spark application, then this is a file where you would set those properties. Last thing I wanted to show you is labeling your actions. So let's check one action like this one. So you retrieve the Spark context, and you can set the strings for the group and description of your job. So basically, you have two placeholders for what you want to say. Now, let's execute this. And go back to this Spark. UI. You see now that I have a clean description and also a clean job Id. So this is for me, this is really helpful. It makes the Spark UI easier to to read and it helps you understand whether something will trigger this with processing or not. And that's it for this notebook on Spark monitoring.

In this video, we are going to train our first machine learning model. So we will create a machine learning model to predict the duration of a ride as a function of the distance. For this, we will use the Spark machine learning library, and the workflow we will be using is a template for all the machine learning applications we will be doing. I need a Spark session and some Python visualization packages. For this model we only need the distance and the duration and so we only need the rides dataset and we are going to use only the not cancelled rides. And in those rides we only use the distance and duration. So we filter the data we are going to use and that is one of the founding principles of processing Big data: is to filter early. Okay, now we are going to look at some visualization using a sampling technique. We have seen this cloud of points before, and this is where we got the intuition that indeed we could predict the duration of a

ride as a function of the distance. We are going to use some of the machine learning library functions to prepare the data for machine learning algorithm. The machine learning algorithm expects a vector of features, and for this we have a VectorAssembler object that we need to first import. Then we need to instantiate. In our case, the input columns will be the distance and the output columns will be the features. And this creates a transformer object that we can use to transform our data. The result is the following. Now, on top of having the original distance in duration, we have a vector of features, and we can see the features is comprised of only the distance in order to train our machine learning model, we need to split the data that we have, and we are going to split the data in two using industry-approved weights of 70% and 30%,

Next, we are going to use a linear regression algorithm, so we need to import the object from the library. We create an instance of this linear regression object that is specific to a context. So we provide thw=e object with the features column and the duration for the label. Other hyper- parameters are available. But for this first machine learning model, we keep things simple and only use what we need. To train our machine learning model, we call the fit() method on our machine learning algorithm using the train dataset. The result of this is a linear regression model. The linear regression instance is an example of an estimate, and the result is a transformer, so a machine learning model is a transformer. We can examine the model parameters. So for a regression linear regression, it's simple. It's this slope and the intercept. So we have the intercept and the slope is a DenseVector of only one number. So if we needed to have this in a NumPy array, we would call this toArray method. There are other ways to examine the various model performance measures. Like most machine learning models, objects, the linear regression model has a summary attribute. And from these attribute we can examine the R^2 metric as well as the root mean squared. Therefore, we get the R^2 measure of

So the R^2 metric is meaningful because we know the range goes from 0 to 1. And so 86 is quite good. The root mean square error in itself is less interesting. It's an absolute value. We could use this metric if we compared several models but for the time being, on its own, it doesn't mean a lot of things. There are other model diagnostics available in the sumary object for those who are familiar with these metrics, now we can use our model to perform some predictions on the test set so we could transform on the test dataset and the resulting data frame as a new column called Prediction. Prediction doesn't seem too good for the short distances, and it's a problem that we will visualize later To evaluate the linear regression model on the test data, we could use also a regression evaluator object. And the reason why we would do this is because an evaluator can be a step in a pipeline. So if we wanted to include in our pipeline an evaluation, we would use this object. So we import an object called regression evaluator. We instantiate this object to a context. We say that we want to use the R^2 metric. We can look at the list of other parameters. Different metrics are available, and if we call the evaluate method or evaluator instance, we get the same result as previously, and we could have also used the root mean square error, and we get

the same result as previously. Again, this gives you the same results, which is fortunate, but it gives you the added advantage that this could be included in a pipeline. So my recommendation would be to get used to using these objects. We can plot our model, and here we see the the problem we have on the short rides. The model is quite good for long rides, but for short rides, then there's a bias. In the next exercise, we are going to try and fix the problem by using a different machine learning algorithm called the isotonic regression to see if we can have a better fit for the short rides. This is to be expected in data science. The first iteration of your model is usually not the best one, and that is the end of this presentation.

In this video, I'm going to walk through the solution of the 13_regress notebook. In this previous notebook, we created a model with a linear regression model that was not very accurate for short rides. So we want to build a new model using a different algorithm that we suspect will be better for short rides. I read the data. It's the same steps as in the previous lesson. We only use distance and duration for this model. We do not need the cancelled lines. This is our data. So again, we use a sampling technique to be able to visualize using Matplotlib our data. We assemble our vector of features using a VectorAssembler. This gives us a new data frame with a vector features, and that vector only contains the distances. And to train our model, I'm going to split our dataset into a train dataframe and a test dataframe using the industry approved weights of 70% and 30%. So instead of doing a linear regression, I'm going to try an isotonic regression. So I need to implement the isotonic regression object to create a new instance of that object specific to my context with features and duration, I can look at the additional hyperparameters of this algorithm. But for the time being I'm going to just choose what is strictly needed. I train this model using the fit() method on the estimator. I get an object of type isotonic regression model. An isotonic model is a set of slope going in the same direction hence the name isotonic, and this is captured in the boundaries and predictions properties of the model. I use this model to make predictions on the train dataset. I need a regression evaluator. So I need to import the object and create an instance. Okay, so this time we are using the root mean square error, so I need to set it to root mean square error. This time we get a better result than with our linear regression the root mean square error going out for the linear regression with around 130. Now it's less. So it's going in the right direction. You can do the same thing for the test dataset. And we see that in the small increase of the root mean square error. But our model seems to generalize well, and we can look at the shape of our model. Indeed, we can check that the model performs better for short rides because of this of this specific algorithm of isotonic regression. For longer rides, The jury's out. I'm not sure this will generalize well for longer rides, but it performs much better from zero to, let's say, 40 kilometers. And that's it for this solution of this exercise.

In this video, we're going to walk through an example of a classification problem using Spark Machine Learning Library. What we want to achieve is to predict whether a given ride will get a five star rating. I will need Matplotlib as well

as a Spark session, and we will use our joint dataset. First, I need to remove the cancelled rides. We did this in a previous notebook manually. Now we are going to use the transformer to do the same thing in order to be able to include this stage in a future pipeline. The transformer that we use is a SQL transformer that takes as an input an SQL statement including a placeholder for the dataframe to which it applies. For this particular problem, we do not have the corresponding label that we are looking for, and we need to generate it. We need to generate a boolean that says this ride got a five star rating: true or false For the time being, this is the distribution we have to perform this transformation. We are going to use a binarizer, so that's something that we get from the machine learning library that applies to a double and uses a threshold. And above the threshold it will be 1 and below it will be 0. We set the threshold to 4.5, so that we get for a five star rating a 1 and below we get to 0. Next, we need to find what kind of features have an influence on our new label, high_rating. For this, we are going to use a helper function that will help us explore the features. This helper function could explore() will help us understand the influence of a given feature on our high_rating label. First, intuition is that when there's a review for a ride, it's probable that the rating of the ride was not positive. And that is based on the fact that people leave a review and they're not pleased with the ride. And using our explorer() function, we can see that this a negative influence on the presence of a review on the ride. Okay, so we can use this. Next, we suspect that the nicer the car will be, the better the ratings will be. But as is often the case in the real world, we don't have in the column something called quality of the car. What we have is a proxy to that, and it's the the year the car was released. So we can use this. And we see a positive association between the vehicle year and the average of high_ratings. So we have two good features now. And we want to add another one. And we are going to look at the vehicle color. That idea seems strange, and it is. The reason for this is not for the performance of the model, but is to showcase the features of the Spark machine learning library. So I apologize in advance; the result of our model will not be fantastic. But we will illustrate some concepts of a feature engineering such as Indexer and OneHotEncoder. Machine learning algorithms understand numbers. So we need to transform our string columns into something that the machine learning algorithm can use. The first idea is to create a kind of lookup table and use the key instead of the value for this. Again, we use something that is part of the machine learning library called the StringIndexer. So we instantiate a StringIndexer, and then we fit the indexer. And it uses this kind of a lookup table. And so we have an index vehicle color instead of having a vehicle column made of strings. This is one first step, but it's still not a good representation of the color of the vehicle. The reason for this is that the way it is represented may imply that there is an other relationship between the colors or that you could perform arithmetic operations between the colors. That is not the case. White plus gray doesn't equal silver, and blue is not more color than silver. So this is misleading. The correct representation for this information would be using OneHotEncoder. That's another utility that we import from the machine learning library. And now we get a new representation

of the vehicle color using Spark vectors. The way to read these Spark vectors is that it's a vector of 9 position for which the nominal value is in the second position because Spark vectors are zero indexed. So let's take that, you know, lookup table white should be the second in the second position, and it is. So Spark vectors are a way to represent vectors that is much more compact, and especially vectors coming from OneHotEncoding with only one nominal number. Okay, let's say we are pleased with our selection of features. We assemble them using a vector assembler. And this time the vector assembler creates a Spark vector. And since we have done some work on our features and for future iterations, we can save our dataframe to HDFS. Next, we need to create our train data so we do a random split using 70% and 30% and also adding a seed so that the training and testing data are random. But it's reproducible random. For the classification, we are going to use a logistic regression. We import the object from the library, and we instantiate the object to our context and we can have a look at all the hyperparameters available. Again, we used what is simply required for this first model. We trained a model with a fit() call to the machine learning algorithm. And we get a logistic regression model. We can check that by looking at the type. Some of the characteristics of the model are available at the top level of the object. So we get an intercept and a vector of coefficients. No, it's no longer one single line with a slope, it's a two vector in a multidimensional space. There is a summary also object embedded into the model, so we can ask to see how many iterations were performed and what was the history of the objective of those iterations. We can also plot those iterations. We see that there was no improvement, so iteration stopped after a while and then we can also retrieve this measure of the performance of the classification called areaUnderROC, and we get the 63%, which is not great because the range of this metric is between 0 and 1. It's an area, and this is typically not great. Again, we chose features for the pedagogical value more than for the business value. We can look at the ROC in the table, but we can plot it also. We can plot the ROC curve and it matches our expectation. It is not great. The area under the curve is the area under the blue line. And the closer it goes to 1, the better it is. The black line is just around the heads or tails. Our classification is slightly better than a random classification, which is not great. If you want to know more about those metrics, there is a dedicated chapter on those metrics around the end of this course. So next we look at the performance of our model on the test set, and the result is a binary logistic regression summary and we can perform the plot. Again, it's the same one. Alternatively, we could use an evaluator. So for this we generate the prediction from the test set. We import the binary classification evaluator. We instantiate the object to our context. We can have a look at the hyperparameters that are available, and then we perform the evaluation. Of course it gives us the same result, But again, using an object that we can insert into a pipeline. We could also have used the areaUnderPR. This is an alternative to the other one. It's also not great. This is the end of this notebook. In the following notebook, I will walk you through the solution of the exercise below.

In this video, I'm going to walk you through the solution of the exercise in the 15_classified notebook. The first question was about looking at the possibility of including vehicle_noir as a feature. For this we use our helper function called explore(), and we see that yes there is an influence on the high_rating from vehicle_noir. So we add this to our vector features. We performed the split again. We train our model again and perform some predictions. Then we evaluate the area under the curve, and we get these disappointing results of 0.62, which is similar to what we got before. Last, we were asked about this question: what cardinal sin did we commit? The cardinal sin we committed was to look at the full dataset to assess the potential of vehicle_noir. Well, by doing this, we leaked some information from the test dataset to the training phase, and that's a cardinal sin, and that's it for the solution of the exercise

In this video, I'm going to present the hypertuning features of the Spark Machine Learning Library. For this I will need the Spark session, and we will use the modeling data that we created in the previous notebook. I split the data in two datasets, one for training, one for testing For hyperparameter tuning, you need an estimator; that's the machine learning algorithm for which you try to find the best combination of hyperparameters. You need to build a grid of those hyper parameters. You need an evaluator to give you the performance of any given combination of hyperparameters and a validation method. So the estimator we will use is the linear regression that we used previously also. So I'm going to import the object from the library; instantiate object. Now, I have a linear regression algorithm. I can look at the hyperparameters and when we use this algorithm for the first time, we didn't touch any of those hyperparameters. Now, we're going to use some of them. And we are going to specify an hyperpara- meter grid. The hyperparameter we're going to try and tune is the regularization parameter: regParam The idea of regularization is that those of either the weights or the square of the weights of the model can improve the accuracy of the model by penalizing the large weights of the model. So the result of regularization gives you a smoother model with weights that are not so large. And usually that results in a model that generalizes better. We are going to study the impact of those values for organization. So this is the grid we have. We could have specified the the grid differently by specifying that that we are going to use L1 penalty, which means that we are going to use the weights themselves, not the square of the weights, which would be a L2 penalty. Now, we need to specify the evaluator, so we will use the regression evaluator as we did previously. And to validate the accuracy of our models, we are going to use a trained validation split object. This will allow us to test the accuracy of a model train on 75% of the training data using the remaining 25% for testing. Now that we have defined our train validation split, we can use fit() and now spark will test all the values for our regularization. So it's doing the computation we did once, it's doing it for all the points in the grid. So we have six points. We took a bit longer than usual. The result of this competition is a train validation split model, and we can look at the metrics that we obtained. So we are using the root mean square error and smaller is better, so we don't

see any improvement using regularization with the range we selected. We can visualize this by plotting those results. There's no local minimum, so it seems that no amount of regularization can improve the accuracy of our model. So the best model is obtained for regularization equals 0. And as a pointer to the best model inside the tvs_model, which we can use to look at the properties of this model. Okay, so this is not great. Maybe we can improve this by using more CPU. So that's always the tradeoff that we will find in machine learning. You trade more CPU for better accuracy. And what we do now, instead of using a train validation split, we are going to use a cross-validation. The idea of k-fold cross-validation is to iterate on the splitting of the training set to neutralize any bias that could result from the first splitting. So you do it several times and then you spend more CPU, but hopefully you get a better result. We can visualize our results, and we still don't get any improvement, no local minimum. We do have a best model, the one with zero regularization. And we can compute the performance of this the best model. That's it for this video on hyperparameter tuning. In the following one, I will walk you through the solution of the corresponding exercise.

In this video, I'm going to walk you through the exercise contained in the hyperparameter tuning notebook. In the previous range for regularization, we could not find any local minimum. So the intuition of regularization is that a small dose of regularization can improve the accuracy. So we are going to focus on the start of the range. And then we train a new model using this new range for regularization. And look at our results. Still no local minimum, and our best model is always the one with zero regularization. Next, we're going to create a grid with two dimensions where we mix L1 and L2 penalties to see whether that has an effect And that's already done. And let's see if we get better results. It's not obvious. And that's it for this exercise on hypertuning parameters.

In this video, I'm going to present the clustering algorithm taken from the Spark Machine Learning Library. So this will be our first example of an unsupervised learning. There are fewer clustering algorithms than other kinds of algorithms. We're going to use the Gaussian mixture model to try and cluster the student riders by their own latitude and longitude. For this, we will use the usual suspects with the addition of folium, we'll create our Spark session and use the riders. Okay. We need to install folium. Okay. So that will now create my Spark session. And we'll work on the rider's data. To be more specific, we are only interested in the students. We can look at the home latitude and home longitude, but we can use folium to visualize our data. And that's the cool thing about visualization. Depending on the people, some people see two clusters. Some people see three clusters. So it depends on the resolution that we use. Now, I can make this one cluster. Okay, let's leave it at that. For the time being, the data that we need is home latitude and home longitude. And as usual, we'll use a vector assembler to create a vector of our features. We will import the Gaussian mixture object and instantiate this object, and we'll provide the features column and the hyperparameter for the number of clusters we want to see. We are going to set it to two. Additionally, we provide a seed to make our

experiment reproducible. We could fit on our model and using the assembled datasets because it's unsupervised learning, we don't need to split the data so we can use the full data. And the result of this is a Gaussian mixture model. And we can examine its properties, particularly its weights or the distribution. We can check if this model has a summary. So we can leverage the summary and look at the cluster sizes. And we can generate prediction for any given student belonging to one cluster or the other. Okay. We can evaluate our Gaussian mixture model by using the log-likelihood. But for clustering, there is an interesting metric called a silhouette that we can use as an evaluator. So we import the object, and we get this result. And the silhouette metric ranges from -1 to +1. So we get a very good score. And we can explain this by visualizing the results. So we have two clusters. That's what we asked for. We have the blue one and the yellow-orange one. And the clustering is particularly good because we are dealing with an American city with a very rectangular layout, and we are dealing with students that traditionally live on campuses. So we can see that we have the North Dakota State University here. And if we zoom a bit on the yellow one, there's the Minnesota State University of Moorhead. So that explains why the clustering is so successful. Let's look at the silhouette definition. What I wanted to show you is this the the two dimensions of the quality of a of clusters is cohesion and separation. Cohesion means that the cluster is dense and separation means that the clusters are well separated. So they were separated. And depending on the visualization, they can make them more or less dense. So we have two good clusters, and we can use those clusters to further analyze the cluster, so we can explore the cluster profiles and look at the distribution of genders using a groupBy().count(). But we can also use Seaborn to visualize the same information. Something that we haven't seen until now we can save our models to HDFS, and we can look at the all those models that are stored. So we have two subfolders: metadata and data. In data, we will find a parquet file. And in metadata, there will be a part file that we should be able to visualize (unintelligible). And indeed, the content of this file is a JSON definition of the model with all its hyperparameters set. So, together with the data file, you would be able to recreate the model. I can load the model back and I can use the model that I have just loaded to make predictions qgain. In the next video, I will walk you through the solution of the exercise.

In this video, I'm going to walk you through the solution to the exercise on clustering. So the first question was to create a Gaussian mixture model with three clusters, and we will call it gm3. So we create a new instance of the Gaussian mixture object with a hyperparameter k of value 3, and we call fit() on this object to create a new Gaussian mixture model. We can examine the weights and gaussianDF of this new model, but maybe more conveniently we can plot the clusters created by this new model. And we can see that we have three clusters now. One is centered around the Minnesota State University and the two others are close to this North Dakota State University. Visually, this clustering is not as good as the previous one. Specifically, the separation has diminished between the cluster orange and cluster blue. So we will check that

by going through the metric. We can create predictions, also. Let's measure the silhouette metric. You see the silhouette metric has dropped down significantly. We can print the distribution of a gender by cluster like previously and plot the distribution of gender by cluster also. That's it for this exercise on clustering.

In this notebook, we're going to look at a different use case involving text data and a new kind of cluster algorithm called latent dirichlet allocation. And for this notebook, we're going to leverage the reviews of the rides. Reviews are text data. And here we have a sample of the reviews: the first five. What we're going to try and achieve is to create clusters of those reviews using the latent utility allocation algorithm. This gives us the opportunity to touch on natural language processing. This area of machine learning as some unique tasks, and we are going to study some of them. The first one is to retrieve the tokens from the reviews. For theis we'll be using a tokenizer object that we instantiate and then apply to our reviews to get a tokenized dataframe. By doing this, we lose some information: the information that was stored implicitly in the structure of the sentence of the reviews and this representation of our text data is called the Bag of Words. The first thing that we notice on our tokenized dataframe is that it's not clean. We haven't removed the punctuation, so we need to use something more refined than the tokenizer. We're going to use a REGEX tokenizer. This will help us get rid of the punctuation. Now we have clean bags of word we can visualize our dataset using a world cloud and doing this we see that our data is still not clean. We need to remove what is referred to as the stop words. It's words that do not carry a lot of semantic value. And we need to remove them. There is also a stop words remover object in this Spark Machine Learning Library, so I can import the object, instantiate it and use it to transform our datasets. And now I have a clean bag of words representation, the frequency of different worlds Natural language processing has this characteristic that the data that is used for these algorithms has a lot of dimensions. Every single world can count as a dimension. So that's why it's important to study the frequency of words, to focus on the most frequent one, to reduce the dimensionality of the problem. For this, we're going to use another object from the library called the CountVectorizer, and we will limit our vocabulary size to 100. And because of the size of the vectors that we get, we're going to store the results as a Spark vector. Each review now is projected on a

and the victors in those space are represented by Spark vectors. Okay, we have done the feature engineering part of the job. Now we can move to machine learning. So we want to cluster our reviews and for the first iteration we'll use two clusters so we can build the LDA object and instantiate the object with our dataset. The number of clusters we want and seed to make the experiments reproducible. We can look at the list of hyperparameters that would be available. In this notebook, we are going to use the strict minimum. We train our model using the fit() method, and the result we get is a LDA model. This model can be evaluated using several metrics. We can look also at the most important words in each topic, but because of this representation, it's more convenient to visualize the results. And so we have two topics. So the topics have no

names, they are latent. One is about, let's say, the air, and the second one is about the driver. And we can apply this topic model to our data. We see that "Dale is extremely cordial." It seems to belomg more to topic 0. Very junky car belongs more to topic 1. The third review about stench is about the air, so it is topic 0. "No trouble of note" doesn't register with the system, possibly the vector resulting from the projection of the review on our space is null. The driver drove so well belongs to the second topic. So this algorithm is useful to get a feel for the set of texts and try to characterize these documents without reading documents. And it's a very popular use case. In the next video, I will go through the solution of the corresponding exercise.

In this video, I will go through the solution of the exercise on using latent dirichlet allocation on text. So, the first thing we were asked to do was to change the number of clusters. We set it to

Then we trained our model. And we can look at the topics we get. This time we have three topics. One is still about the air, one is still about the driver, and one is more about the car. We can perform predictions. And look at the results. We can see, for instance, that second review seems to belong more to the car cluster. Next we look at the another tool in natural language processing called ngrams. We import ngram and create an instance of an ngramer. So that would be bigrams and bigrams are the most frequent sequence of two words in a given input. And we can apply this transformation to our input data. And look at the result. Indeed, our input data has been transformed into bigrams. You have the review was "Dale was extremely cordial" becomes "Dale is," "is extremely," "extremely cordial." Ngrams can be used like the latent dirichlet allocation algothrim we just saw to get a feel for the content of a given document without reading the document. And that's it for the solution of this exercise.

In this video, we're going to look at another use case of machine learning, which is recommendation. Recommendation is used on many retail websites to try to increase the size of your basket, but it's also used in all the popular websites that push content on your phones like YouTube, TikTok, Netflix. What we're going to use here is a new dataset. The scenario here is that duocar has a partner called Earcloud that streams music in the cars, and we are going to leverage the weblogs from Earcloud to build a recommendation system for the riders. I create my Spark session and with the data that we will be using, it takes data; therefore, the schema for this is a simple schema with one column called Value of type string, and it's weblogs with a familiar structure. This gives us the opportunity to flex our regular expression skills, to transform this unstructured data into structured data. We use this pattern to extract from the logs what amounts to the canonical recommendation dataset featuring the user, the artist, and a proxy for the rating, which will be the breakdowns. It's usually difficult to have users from a service provide explicitly feedback. So in many cases, in machine learning we use implicit feedback. And play count is a very reasonable implicit feedback for "I like this music. I played several times." Unfortunately, it turns out that the data is not clean, and that is to be expected

with weblogs. There are some negative values in binary form, so we need to fix the playcount, and we do that by taking the absolute value and casting the value into our base 10 system. Okay. So now we have what is often referred to as a canonical recommendation dataset with the user, the artist, and and the proxy for the rating called the playcount. And because we did some work on this, we can save it to HDFS. We are going to go through the usual workflow. This time we use supervised learning. Okay, so we are going to split the data For recommendation. There's only one algorithm available in the machine learning library, and in general there are not a lot of algorithm. It's called alternative list square, or ALS for short. So I import the object from the library, instantiate the object, and look at the hyperparameters of this algorithm. There's one that we will use, and that is the problem with this approach. It's that there are no points available for a new song or for a new user, and therefore it's very difficult to for the algorithm to produce reasonable recommendations. So this is called the call stack problem. And for us, we'll just drop the corresponding record and now we can train our model. And what the algorithm does is try to find a diagonal matrix that explains the best, the points that are available, and fills the points that are missing with probable values. We can look at our model, and it's difficult to have an opinion on this. Some artists are not representing in the training data, and we can create a recommendation on our training data, and then perform an evaluation using a regression evaluator on our training data. And the metric that we use is the root mean square error. Okay, so did we upset the points that were already present in the matrix? So using this is difficult to say. We can perform predictions on the test data, and this will allow us to see whether we generalize. well, and that is the case. So we can use the matrix in two ways: we can look for users that would like an artist or offer the artist that one user would like. And that's it for this notebook on recommendation.

In this video, we are going to use the pipeline feature of Spark Machine Learning Library. For this notebook, notice that I'm using a slightly larger resource profile in anticipation of the memory required for the processing. I create my Spark session, and we will be using our joined dataset and we are going to use supervised learning. So I need to split my dataset in a train and test dataset. Next, we are going to specify the pipeline stages. So all the transformation we are going to perform on our data need to call an object that can fit into a pipeline. But first, we filter around the cancelled rides using SQL transformer. I believe we already saw this one. So it takes as input a statement with a placeholder for the data- frame to which it applies. We use the same transformer to generate the reviewed feature. We used a StringIndexer and OneHotEncoder to process the vehicle_color. And then we assemble our features with the VectorAssembler. This time we are going to use a RandomForestClassifier. So we need to import the object and instantiate it. RandomForest uses more CPU than the other classifiers that we have used. So we are again trading CPU for accuracy, but we are going to add to this grid and hyperparameter tuning. The most sensitive hyperparameters for RandomForest are the number of trees, the depth of the trees and the sub- samplingRate, so we are going to build a grid with those

three hyperparameters. And for the evaluator we will use a multi-classification evaluator. The problem we are trying to solve one more time is to predict the star_rating for a ride. Next, because we are doing hyperparameter tuning, we need a validator and we will use the train validation split that we used previously. Now, we can create our pipeline. The pipeline is made of those stages: filterer, extractor, indexer, encoder, assembler, and validator. We can call the fit() method on the pipeline to obtain a pipeline model. Okay, so this took more time than previously, because we had a lot of combinations of hyperparameter, and the algorithm itself takes a bit longer. We can inspect the pipeline model, so it's made of instances of the stages. And we can drill down on one stage and find all data for thendexer. We can find the labels of the colors that have being used. For the validator, we can look at the metrics. Embedded in the validator model is the best model. And we can ask to see the values of the hyperparameters for this best model. The best maximum depth is 5. The number of tree is 50, and the sampling is half of the population. Notice that we have used the underlying Java object to get to some of those properties because in previous version of Spark, the API was not exhaustive. But let's try to do this now and see the problem has been fixed. Yes, it has. We can plot the features importances, and we see that the fact that the ride has been reviewed and the vehicle year are important, and then the colors are more like noise. Again, as I said previously, the choice of this feature, vehicle color, was more to illustrate the features of the machine learning library, such as StringIndexer and OneHotEncoder, and to create a very efficient and accurate model. We can also save and load the pipeline model. So first we write it and it's going to use the same format that we saw previously. We can inspect the results. So we have a metadata folder. That folder contains a file with the JSON definition of the pipeline. The stages folder contains as many subfolders as there are stages, and for each stage we will find the same thing: a metadata folder containing the JSON definition of the stage. We can load the pipeline model back into a pipeline model loaded variable. I don't think we need to do this. We can use the model that we loaded to make some predictions. Last, we are going to evaluate this pipeline model to see whether the spend we did on CPU was worth it. We can build a confusion matrix. The confusion matrix is not clean. We can confirm this with an evaluator. I have to work around this bug of the UI to see the value from the evaluator. So it's not great. It's not very different from the previous attempt to do classification. If we compare this to the lit model that we will call prediction baseline that predicts that the the ride will always get a five star. We are not much better. Again, I have to work on the UI problem by forcing the print. Okay, so all these efforts, all this engineering, for this disappointing result. So at this stage, having used the best algorithm and trying to hypertune the algorithm, we should go back to our features and try to select better features. That's it for this note book on using pipelines with Spark Machine Learning Library. In the following video, I will go through the solution of the corresponding exercise.

In this video, I will go through the solution of the exercise on using pipelines with

Spark Machine Learning Library. So the first question was to import the RFormula object. These are familiar objects: allows data scientists to specify multivariable regression using the a popular R syntax such as this one, which reads predict star_ rating as a function of review, vehicle_year, and vehicle_color. Now we can create a pipeline made of this plus the filterer and extractor before. Fit this pipeline to obtain a pipeline model. Overwrite the previous one. Load it back into a new variable called pipeline_model_loaded and use this new model to make some predictions. That's it for the solution of the exercise in the pipeline notebook.

In this video, I'm going to present you a solution that allows you to serve a machine learning model train with scikit-learn using the UDF and Spark data frames. I need a Spark session as well as the data from the rides. The machine learning model that we will use is a nice isotonic regression, but this time train with scikit-learn. I'm going to prepare the regression data. We don't need the cancelled rides, and I'm going to use 10% of the data and put that in a Pandas dataframe. And I can use Matplotlib to see the correlation between the duration and distance. The features will be the distance, and the label will be the duration. I will use train_test_split object to create a train and test set. Notice are similar to the workflows are scikit-learn and Spark Machine Learning Library. We train our model using the fit() method. Evaluate our model using the root mean square error So, for this, we first create predictions on the training dataset and test dataset. And we get similar results that likw the ones we got with the Spark Machine Learning Library. We can plot the model we created. It's similar to what we had before, and since we're using Python, we can use the pickle package to store the model in a file. Now we're going to use the spark to serve the model in a UDF. We need to import the Pandas UDF object, and we define our Pandas UDF to be this to predict using the model we created before. And this automatically broadcasts the model to the executors. Now we can perform predictions using the UDF. This code will need pyarrow to be installed on all the worker nodes, because it's used for the pandas UDF. Spark took care of both casting the model. If you want to do it explicitly, you can too. I don't really see the point of doing extra work, but why not? Of course it gives us the same results. And we can also evaluate our model using a regression evaluator. And I need to use the print function to work around the UI bug and again, the same order of magnitude for the accuracy of the model using root mean square error. And that's applicable only if you have models that have been trained with scikit-learn, you can very well pickle them and then reload them and serve them. That's it for this notebook on deploying a scikit-learn model using a UDF.

In this video, I'm going to show you how to deploy your model in CML. I have this file that uses the model that we created in the previous notebook in

We created a model that predicts the duration of a ride as a function of the distance. We stored it in a pickle file. So what I'm doing in this file is that I'm loading the model from the pickle file, and then I created a predict_CDSW

function that takes as input the distance in JSON format and outputs the predicted duration in JSON format. I'm going to leverage this file to create a model. For this I go to the model's menu and click on Create New Model. I need to give it a unique name and then provide a description. Our environment is secured so I don't need to enable authentication. I will select the file that contains the predict function. Okay. Carefully, copy the function name. Provide a test input in JSON format as well as the corresponding output also in JSON format. So for this I need to fix the quotes. I could change the runtime profile, but I'm not going to. I don't need something very fancy and I could choose from 1 to 9 replicas of this model. I'm going to use one in the interest of time, and I will click on Deploy Model. So now the model is being built. So the way it is built is by executing a Docker file. That will create a container with all the project files and an end point which will allow external calls to the model. The model has been built and is now deploying. We can monitor all the logs, and now it is deployed. I can test the model. And it gives me the expected results. You can see here how to call the model in Shell script, but also in Python and in R. And that's it for this video, on deploying a model using CML.

In this video, I'm going to present autoscaling, performance and GPU settings in CML. So how does autoscaling work? Your CML workspace or cluster consist of several groups and for two of them, namely the CPU and GPU, you can set the range of the autoscaling. You could say this is the definition of a workspace, it's a set of instance types together with autoscale ranges. How does scaling up occur? It's automatic when the scheduler cannot find a node to schedule an engine pod, because of insufficient CPU or memory, then the engine pod will be in a pending state. The autoscaler will notice this situation and provision a new node in the cluster in the opposite direction. Scaling down occurs when the three conditions are met: the node does not have non-evictable pods, the node CPU utilization is less than 20%, and the number of active nodes in the auto-scaling group is more than the configured minimum capacity. The take-away from this mechanism is that for GPU workers, you want to set the minimum rage of the autoscaling to zero, so that you don't incur the cost of the GPUs when you do not use them.

So this is the panel that allows you to set your GPU settings. So you choose the instance types. Make sure that the instance type you select is available in your region. And then you setthe autoscale range and the wisdom is to set the minimum to 0 so that you don't pay for the GPU all the time, and you set the root volume size. For these settings, you need to have the MLAdmin role. Still, with the MLAdmin role, you can set quotas by users on GPUs, CPU, and memory. You should do that also to control the GPU usage. Then as a user, this time not as an admin but as a user, you can change the ranges of CPUs and GPUs, and you can add GPUs to your workspace. When you do that, you need to be aware of the financial implications of your choices. GPUs are still expensive. They are still more expensive than typical CPU counterparts, so be careful with your choices. And then once your workspace is GPU enabled, you can add GPUs to your sessions, jobs, and experiments. And for all of them

you will be using the same kind of settings. You have a specific dropdown box for GPUs, and this dropdown box must be set according to the library that you choose for the edition. So if you're using GPUs, you need to use the Nvidia GPU edition instead of the standard library. Otherwise, it will not work. Also, when you start using GPUs, you might want to write a code that is hardware aware, meaning that the code will adapt to CPU and GPUs. Depending on the package you use to code, you might have to do some extra work or none at all. This is in the example, it shows you what you need to do in with TensorFlow to use the MirroredStrategy class to distribute processing. If you were to use Spark and dataframes, you wouldn't have to do anything. So starting with Spark 3.0, Apache Spark can use GPUs to accelerate the end to end preparation and model training on the same Spark cluster. It does that by providing alternative library that endorse the calls to the dataframe API using GPUs. Another characteristic of GPUs is used to minimize the shuffle or optimize the console performance by bypassing the PCI bus. This acceleration can be used for Spark dataframes in the Spark SQL libraries. It's not available for datasets and RDDs, so it's focusing on the most popular data abstraction in Spark and the most popular library also. It's still a continuous and ongoing improvement. So this representation might be outdated. And recently there were news about some improvement made to the Machine Learning Library. And this is the new vision that is shared between Nvidia and Cloudera. The idea is to use GPUs for the full data pipeline, not only for the model training phase, but for the data preparation. What can you expect when you use a GPU in your code? Well, I've made three experiments using 0, 1, or 2 GPUs, and you can see that using one GPU reduces the duration of the job by a factor of more than ten. So it goes from 113 minutes to 10 minutes. Adding a second GPU still reduces the duration, but not so much. We go from 10 minutes to 8 minutes, so adding just one GPU already makes a massive difference.

In this video, I'm going to go through the autoscaling performance in GPU Settings Lab. First, let's look at our workspace. And view the details. And if we scroll down, we can see the definition of the workspace. There are four groups. For each group, there's an instance type and an autoscale range. For the CPU and GPU workers, those autoscale ranges can be configured by the users. For the time being, the GPU we are using is a g4dn.metal. It has eight GPUs. For the workers, we have a range of 1 to 10 I think the minimum for a worker is one. We ask for GPU, you can have zero and you should have zero to minimize costs. Let's create a new project. We will call this project Deep Learning with GPUs. So for this we use a GitHub repository. We'll use Python 3.7, and we need to add the GPU variant. Then we can create the project. Next, we are going to run a job with pur GPUs to make a baseline, and we'll call this with GPU. We don't need Spark. We use the standard edition and no GPUs. Okay, we have our session. So if we look at the environment_setup.py file, we need to run this to install the required libraries. And it takes about 7 minutes. Okay, setup has run. Now we are going to do some testing. And time the execution of our script with our GPUs. Okay, let's note those timings

somewhere. Now, guess what we are going to do? We are going to create a new session, but this time with GPU. select the correct library add one GPU and then start. I get a message from Kubernetes saying, oh, this is something I don't have, and hopefully Kubernetes is going to autoscale to provide me with the required GPU machine. Now, it's auto scaling. We finally got our GPU, and we are going to perform the same test. And then we can see that the GPU is being used for processing. And those are the results we got; going to copy them. We see a lot of improvement for the user. timing, and the experience was much quicker anyway for the user. So that's it for this lab on using GPUs.

In this video, I'm going to present common model metrics and monitoring features. You need to monitor your models because they are built using an assumption that is false, and this assumption is stationarity: the fact is that data changes and concepts evolve. So you need to monitor your models to see whether they drift or not. So there's such a thing as data drift. It means that the data that was used to test and validate the model has not changed. You have a couple of examples in the slide. For instance, the COVID-19 disrupted the consumers behaviors and the inventory management models that were built previously. Spam filters need to be constantly evolving to remain effective and so on and so forth. Concept drift happens when the patterns that the model learn no longer hold. This can happen when, for instance, when competitors launch new products. In the industry, it can be caused by mechanical wear of equipment. The fact that the process parameters slightly changed then can affect the quality of the predictions. So in an ideal world, you would use something like this called adaptive learning, where you would monitor the performance of your model and define a threshold that would trigger the readaptation of the model. The problem of adaptive learning is that it depends on labels being available. The example that is given in the blue box shows that if you wanted to train a system to detect hate speech from live tweets, you would need continuously

and that is just not sustainable.

So what are the common metrics that we can use for monitoring? So for regression, you have the metrics based on the error; so you have the mean error, the average of all the errors. That is seldom used because the negatives and the positives cancel each other out. To fix this problem, there's the mean absolute error. But the mean absolute error is an absolute value, and this is less useful than a percentage or a metric with a fixed range. So the mean absolute percentage error, it's useful because as a percentage twinges from 0 to 1, and you can quickly assess whether your performance is good or not. The mean square error is another way to fix the limitation of the mean error, which also gives more weight to larger differences and the root mean square error is measured in the same unit as the target variable. For classification, you have several tools: you have accuracy which is the division of the prediction by the total predictions. But this one can be dangerously misleading on skewed data. The confusion matrix gives you a full view of the performance of your model with true positives, false positives, false negatives, and true negatives. From this matrix you

can extract the precision and the goal, but if you want a metric that ranges from 0 to 1, then use the F1 metric, which is two times the precision multiplied by the recall, divided by the precision and recall. Another popular metric for classification is the area under the curve, and this one ranges from 0 to 1,

and 0.5 being a random classification. You can see in the diagram the area is in yellow is the area under the curve that is obtained by changing the threshold of your classification. For clustering, the popular metric is silhouette. The silhouette ranges from -1 to +1. The closer it is to 1, the better it is. And it measures at the same time two characteristics of clusters, the cohesion and the separation.

Now we are going to look at a Python package that allows you to implement continuous model monitoring. This package is called Evidently. It measures the data drift using the p-value. So in this example for house prices, it detected the drift for the zipcode, the view, the bedroom, and the bathroom. It's using the p-value to detect if the distribution has changed significantly. The p-value is the probability of obtaining test results, at least as extreme as the results actually observed under the assumption that the hypothesis is correct, which is statistical thing for saying nothing change. So if nothing changes, the new distribution is not very probable. You can also measure target diff to see whether the target as significantly changed using the same technique of p-value. And you can measure regression of performance using the metrics that we just discussed. To use Evidently it is quite straightforward. You need to install the library with pip, then import the object that you plan to use. Next, you prepare the data, and you can generate the dashboard using a call to the library. And you can save your dashboard to HTML if you want. So you can use this package to implement continuous model monitoring.

Alright, hi everyone. I'm Andrew. And in this video we're going to take at a newly released Applied Machine Learning prototype or AMP, as we like to call them, that demonstrates one way that we can achieve continuous model monitoring on CML. And in particular, we're going to take a look at how we can combine CML's model metrics feature along with evidently open source monitoring dashboards to monitor for data and concept drift. And so before we get started, we'll just take a look at a few slides here that demonstrate- that discuss what the importance of monitoring for concept drift is, and also a little bit of the the flow diagram for how the AMP actually works under the hood. So let's get started. All right. So as data scientists, we often pour a ton of our effort into training and fine tuning a model that performs really well on some desired metric. So we have a held-out test set and we train a model that gets really good performance, so it's more moderate for accuracy on that held-out test set. But once we put that model into production where it's actually making predictions on new incoming data, our model performance is naturally going to just decline over time. And this happens because what we have is a static model that was trained on a historical subset of the data that is trying to operate in a dynamic environment where things are going to be constantly

changing all the time. And so these changes in the properties of our data and the statistical relationships that our model had learned, this is considered concept drift. And again, it's naturally occurring phenomenon. It's just a part of production ML and if it's not accounted for, it will make our models obsolete. And so a common approach for dealing with this is to implement a an adaptive learning strategy. And so this is where we are continuously monitoring for changes in our data characteristics, and we're also monitoring for changes in our model's performance, and we're looking to detect if a concept has drifted. And if it has, because we've been monitoring it, that we can trigger a retraining pipeline, so our model can adapt and learn that new concept. And this type of adaptive learning strategy, it really requires that there's some fundamental infrastructure in place. And mainly what we need to have is some detailed logging set up, so the ability to log for projection and ground truth values. And also we need to have an intuitive set of visualizations that help us indicate or help us indicate to the modeling team when and where and how model performance has drifted. So you need a way to actually inspect and look at all all the different things that might be changing. And so that's what this AMP does. It shows how to accomplish these two pieces using CML model metrics feature, which is basically a managed PostgreSQL database and the set of easy to use APIs that allow users to log and query for performance metrics. And then the visual- for the visualization component, we show how you can how you save up metrics and populate these reporting and monitoring dashboards that are specifically tailored for data and target drift through Evidently AI. Okay. So before we get into actually looking at the AMP itself, we'll take a look at a flow chart that shows what's going on with the AMP in the background. And so what we have here in this AMP is actually a prototype that's intended to emulate some type of a production system, which naturally is a difficult thing to do. And so what we actually need to do here is, is the AMP is running a simulation that tries to mimic a production environment over time. And so the AMP actually starts by pre-processing, some data and training a model. And so in this case it's a pretty straightforward dataset. We're using housing prices. So housing sale prices from King County, USA, it's a Kaggle data set, and we use features like the number of bedrooms, square footage, and whatnot to predict what the price of a house will sell for. And so we have a script that trains a regression model using scikit-learn pipelines, and then we deploy that model to a hosted CML model endpoint. And in the prediction script that actually gets deployed, we include this functionality that you see on the screen here that is out of the model metrics API, and that is a decorator and the cdsw.track_metric method, which we'll take a look at later. But basically this allows us to automatically log anything we want. So features and predictions, every single time that endpoint is is hit for inference, which is a really useful feature to have. Okay. So that is the setup that is training the model on a subset of historical data. And then we run a simulation which is everything to the right here that runs in a in a monthly batched fashion. So in 3.a you see that for every new month that goes by, we need to query for all of the newly listed homes, and we want to make predictions about what we think they're going to sell for and then here

we're using a useful method called cdsw.call_model, which is a one liner to just make a prediction. Again, because we have the end point with wrapped with this model metrics feature, this call will automatically log out those prediction, the input metrics, the features, and the prediction value itself to that managed database. The next thing the simulation does is it looks up for this last month, were there any newly sold homes? And so to do that, we can use a combination of the read metrics and subsequently we then use the track delayed metrics. So for any houses that did actually sell in this last period, we can look up their ground truth. So the value that they actually sold for and join that into the database along with the input features and the predictive value that was initially predicted for it. And these delayed metrics are what this ground truth delayed metrics is, what's going to allow us to actually calculate performance in the model. And then finally, after we've got these these new ground truth values, we calculate our drift metrics and generate a report using Evidently AI's visualizations. And we deploy that dashboard as a hosted application in CML so that our end users can can use it to monitor for data drift or concept drift. So in a nutshell, that's what's actually happening here. And this type of cycle would actually run continuously in production. But since this is just a demonstration or prototype, it only runs for, I think, six cycles, six months worth of data. So now over to CML. We can see if you want to deploy the AMP for yourself in CML, you can come over here to the AMP catalog page, and you'll see at the bottom here we have the continuous model monitoring AMP, which we can configure and then deploy. And since I've already done this ahead of time, we'll jump ahead. After the model- after the AMP has been deployed, you'll see a series of completed steps on the screen here. And if we go to the project itself, we see that project creation was successful and that there has been a model that's deployed called Price Reggressor. If we really quickly just look at the script that this model endpoint is running, we can take a look at it here scripts, and then we go to predict we will see that in here we do have this CDSW model metrics decorator around our predict() function, and you also see that in here we're using CDSW track metrics. So we're tracking all of our input features and saving them as input features. Similarly, we're tracking the predicted result from every single call to this function. Is that there is an application up and running? That is our price regression monitoring dashboard. And so if we take a look here, we'll see. It's a pretty lightweight dashboard that hosts, we scroll down, we have three different types of reports. This first one is just monitoring for data drift so you can drill in by feature and look at data distribution changes from the reference to the current timeframe of data that we're comparing against. We can also have a report in here for a numerical target drift. So this is looking at is there a significant drift in our target variable itself, which in this case is price. We have a whole bunch of visualizations that help you inspect if that price target variable is actually drifting or not, and how it compares to historical performance values. And similarly, then we have a regression performance dashboard that includes the ground truth value. So actually calculating error and different error metrics across our different sets of data, what the model was trained on versus the most recent time frame of

batch of data that comes in. And we have lots of different visualizations that we can use to understand error. And if our error distributions are changing. Then finally you'll see that we can filter or browse by different batches of data, so different time periods. And this is a way that as a model maintainer, I can get a good idea of how my model is performing over time and how it may be degrading and when it may need to be retrained. All right. So that is a wrap for this video demonstration of our new AMP that walks you through how one way that you can do continuous model monitoring on CML. I hope you enjoyed this run through and that it was informational. Please do go ahead and deploy this AMP. Run it yourself and reach out with any questions. Thanks.

In this video, we are going to study continuous model monitoring with Evidently. We are going to use an AMP for this, which we need to configure. So, we are going to use the dev mode to work with a subset of all the data. We don't need Spark. We are working with Python 3.9, and I'm going to start the project. This will take time. Okay. For me, the process took a little over 12 minutes, so it went through several steps. First, install the requirements, then it built a model, and deployed the model, and deployed also an application which we can look at. This app allows you to time travel and for all the different dates, you can visualize the data drift, target drift, and the regression performance. So for the data drit, we can see all the nine features involved in the in the model for the pricing of houses. And we can see that at this time three out of nine features had drifted significantly from the reference distribution. The significance is measured by the p-value, which tells us how likely our otherwise the current distribution is given the reference distribution. Based on this dashboard, we can decide whether or not to retrain the model with new data. We can do the same thing for the target drift, so there's no drift detected in the target. So the price of houses seems to be quite stable and you have several dashboards to visualize this. Kind of like this one. So no drift in the target. And you can also look at the regression performance. We have several metrics or several graphics. And what is interesting it shows you whether or not your model is underestimating or overestimating, and it seems to be too good for that. There's no bias detected that would lead to systematic underestimation or systematic overestimation. This app uses a model that has been deployed during step 6. Let's check this model. So we are going to go to model deployments. We see the deployed model, and we can drill down. We can test the model with the test button. Gives us a prediction on the price of the sample house. And we can look at the also how you can call the model using Shell, Python, or R, and we can see that the model is using this function in this script. So let's go and check this script. This is the predict function. We can see that a call to CDSW .track_metric is made to store the prediction metrics in the CML PostgreSQL data matrix database. Remember that in order to use this feature, the CML workspace must have the enabled model matrix option selected when it is provisioned. Next, we are going to look at the simulation. This simulation mimics the production monitoring use case. It assumes the price regression model has already been deployed and accepts the model name as input. The run simulation method

operates. the main logic of this class. It calls all training data against the deployed model so that we can query matrix from the model matrix database for evaluation. Initializes a simulation clock, which is just a list of date ranges from the prod_df to iterate over. These batches mimic the cadence upon which new data arrives in a production setting. For each simulation clock date range, we query the prod_df for newly listed records and score them using deployed model, query the prod_df for newly sold records, and add ground truth to metrics store Query the metrics store for those newly sold records and generate new Evidently reports. We deployed the (unintelligible) application to surface to the new monitoring report. And here we can see the call to the query model metric to read the metrics stored in the CML model metric database. Then we have the build Evidently on those score reports. It builds the reports and saved them as HTML to be served by the Price Regressor when monitoring dashboard application. And here we can see all the simulation deploys the updated price regressor monitoring dashboard application upon simulation completion. Next, we are going to build our own evidently report. Using another project and we'll use a git repo for this from an AMP. We're going to use Python's 3.9. We don't need Spark. The editor will be Jupyter Lab, no GPUs, so we use Standard. Okay, I'm going to launch the project, and it's going to install the requirements. The installation is completed. Let's start the session. I'm going to call it Evidently. We'll be using Jupyter lab and this version of Python kernel. Okay, there's a Jupyter notebook in the code folder, and we can run it in this notebook. We try different dashboards with different levels of verbose. So, here we look at the target drift dashboard. We have a short version. Next, we look at the data drift and again with a short version, we have the combination of the two. That's it for this exercise on continuous model monitoring with Evidently.

In this video, I will present the workflow to provision a workspace. So the Provision Workspace button is available from the machine learning workspace landing page. Be aware that provisioning can take up to an hour, and that domain name is randomly generated and cannot be changed. So you need to specify your unique name in the workspace. Point to the environment where the ML workspace must be provisioned. To access the advanced option, you need to toggle the switch. Here you can access the CPU settings, choose the instance type, the autoscale range, and the root volume size. This is the default size of the root volume disk for the nodes in the group. Then you have symmetric GPU settings. On top of that you have a toggle switch to switch GPU instances on or off, but you go through the same settings, so choose the instance type, set the autoscale range, and choose the root volume size. Next, we have network settings. So you can optionally select one or more subnets to use for Kubernetes worker nodes that's only available for AWS. You can optionally select one or more subnets to use for load balancer. Azure only you can enter a CIDR range of IP addresses allowed to access the cluster, and you need to switch the enable public IP address for load balancer to enable access to your workspace. Last, you can choose to enable governance to integrate with Atlas. You can also enable model metrics to track and analyze and store metrics. Enable TLS,

enable monitoring, and define tags that will be propagated to your cloud service provider account and provide a custom name for the workspace endpoint and URLs of models, applications, and experiments. That's it for this short overview of the workflow to provision a workspace with CML.

In this video, I will give you a short presentation of an an LLMs and generative AI. ChatGPT is a very popular system, and it's a large language model. And those systems consist of neural networks that have been trained on a large amount of text data. What they do given a text input, they predict what comes next. Those systems are already at work in your phones to predict what is the next word you're going to type in your text message. LLMs are foundation models, which means that they can be adaptive to perform a variety of tasks. This is part of the success of these systems, and this is what justifies the investment in their training. But they are not without some issues. The first one is about public hosted LLMs. If you want to get specific answers involving some knowledge about your organization, you have to leak that information to the system and that can create problems. If you are tempted to work around this limit and build your own LLM, then you have a financial problem. And the size of this problem is in millions of dollars, sometimes tens of millions of dollars. What you can do is use locally hosted pre-trained LLMs to avoid the two first issues, but then those pre-trained have no knowledge about your specific context and questions related to that context. You will get some very generic or hallucinated answers. Hallucinations means the LLMs generates a sequence of words that makes no sense and factual but out of context answers mean that the answer that the LLM provide is correct, but very generic and not specifically relevant to your context. This diagram shows you the actual boom of those systems since 2021. Also, you can see the prevalence of Google on this technology. So what can you do with those pre-trained LLMs? Well, you could be tempted to fine tune them, so retrain them, but that would be expensive, although there are some new technologies that are promising and could bring the costs of retraining down. But the current popular solution is to use a mix of prompt tuning and knowledge retrieval. This is what is being done in the AMP that is available in CML. The AMP, like any AMP, is very straightforward to install and, this is the architecture of the system. So the system relies on this black box, which is an open source instruction-tuned LLM provided by a (unintelligible). But we supplement the context by ingesting specific text finds in a vector database that is queried at the time of prompt and used to enhance the prompt before querying the LLM. And this leads to factual responses instead of hallucinated responses. So this slide has been made for our friends at IBM, and you see that the context is critical. And answers can be very different with or without context. For us, this new technology can lead the way to a lot of interesting use cases. For instance, to improve support or to generate SQL queries from natural language. The AMP we provide provides a blueprint for our customers to implement their own LLM applications by leveraging the enterprise knowledge base without making any external calls to open AI or AI services.

Let's go now to CML and look at the demo of this AMP. I have successfully

installed the AMP, so I should have the application ready. I can test the application. This is using the context we provide in the vector database. So we see that within the context, the answer of the LLM is relevant. We also can see that this system relies on an open source model that's already trained and has notions about other things than Cloudera. Okay, let's look at the files we ingested in our vector database. We have something about Iceberg. Let's asks about Iceberg. I'd like to know So you see that without the context, the iceberg is a large piece of ice. And in the context of Cloudera, iceberg is a data warehouse system. Mm.. not the most accurate answer in my opinion; it's a table format, not a data warehouse system for me, but if I asked this specific question to the system, it gives me a reasonable answer with with context. What we are going to do now is ingest in the vector database relevant information that will improve the accuracy of the response from the system. So to do this, we are going to upload some data, and we are going to perform the ingestion job again. And in the file that we are ingesting in the vector database that is very specific answer to the question that we are asking. Here it is. So we are going to try and retrieve this information from the system. I have to restart the application. Let's ask the same question. And we get the correct response from the system, leveraging the vector database and the additional data we ingested.