# **Learning Amazon SageMaker**

### **Machine learning with Amazon SageMaker**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Martin] How can I quickly investigate a new dataset, train a predictive model, and deploy it all in one place? Can I set up an API to host a machine learning model without setting up a web server? How can I create an efficient model development workflow? These questions can be answered by understanding how Amazon SageMaker works. First we'll investigate some data in the cloud, and then we'll train a machine learning model on remote instance, and then we'll deploy it and make some predictions. Hi, I'm Martin Kemka. By walking through this course, I hope I can share some of the tips that I've found and give you a base to start experimenting, iterating, and building models using Amazon SageMaker.

### **What you should know**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] For this course it would be useful to have some experience with Python. There are many great courses on LinkedIn Learning that cover this. Any experience with statistics would be useful if you're interested in delving deeper into the model development aspects, or experience in web development if you're interested in the deployment side of the models. Experience using Amazon services is not required, but an account will be needed to use the platform.

### **What is Amazon SageMaker?**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In a nutshell, Amazon SageMaker is a collection of libraries and interfaces that facilitates the development and deployment of machine learning models. It's worth understanding as well that the SageMaker platform is a collection of different tools and services and they can be selected based on any of your requirements. There are also a number of different ways to interact with the platform, which might make this explanation seem a bit more complex, but at the end of the day it's actually a lot easier to combine code that you've written yourself, open source software, different elements that could be available on the net, and connect that all into one platform. And it makes it a lot easier at the end of the day and accessible to create and deploy machine learning solutions. Now just before we jump in to the Amazon SageMaker platform, it's worthwhile having a definition of what machine learning is here and many people will already be familiar with it, but in order to make the scope a bit more succinct and we'll work with the definition that machine learning is the creation of a function that generates inferences by learning that function from a dataset. So this means the weights or the function is not explicitly designed, but it's learned from looking at a dataset. Technically, any type of machine learning algorithm can be trained on Amazon SageMaker and in later sessions I'll talk about and describe how that's possible. What is consistent with this process is that there's always an element of data selection, modification, and training for each algorithm. The actual type of algorithm that's chosen and the steps to process the data will be distinct and separate for every project you take on. In terms of decisioning, one of the benefits of Amazon SageMaker is the ability to not only train models, but deploy them as a service on Amazon Web Services. This allows for the ability to call this function and get an inference for any query. This opens up the ability to easily try different types of decisions based on the output of many different types of algorithms. So as an example of this decisioning, it would be deciding what you would do if you knew there was a 70% chance of it would rain today? Perhaps the decision that you'd make would be to pack an umbrella. Although the explicit science of the decisioning is not in scope for this course, I'll give examples about how the tools provided by Amazon SageMaker make it possible to include this layout in your projects.

### **How does Amazon SageMaker work?**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] How does SageMaker work? Amazon SageMaker is not one software tool, but rather a collection of tools and services. All of these tools and services allow the training and deployment of machine learning models. Similar to a menu at a restaurant, you can choose and select different items, combinations of items based on your needs. You're also allowed to use other services and tools based on the problem that you're looking to solve. In the simplest of terms, the tools that are provided allowing the training and deployment of the model. Again, with the weather example that I spoke about before, perhaps you have a data set based on the weather for the last few days and you would like to predict if it would rain tomorrow. You can use that data set, import it into Amazon SageMaker, using their libraries, using your own code, and you can train and deploy a model. Amazon SageMaker provides an analytical environment as a service to parse the previous weather data. It also provides tools to manipulate this data and train a model. You can use your own tools to build the model if you wish. And those tools would usually be comprised of different Python modules or Python code that you've written. The platform provides tools to host the model as an API, but you can also set up your own API if you wish. You can even set up your own API in other cloud services. There are many ways to customize this process. And the approach might seem daunting, but once you run through a basic pattern, the opportunities for creation are pretty much endless.

### **Benefits of Amazon SageMaker**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] What are the benefits of SageMaker? There are a number of benefits to using Amazon SageMaker, but here are three areas that can be used as examples to show the strength of the platform. Those areas are accessibility, customization, and efficiency. The Amazon SageMaker model development and deployment tools can all be accessed through a web interface by the AWS console. This means that all you need to get started is a web connection and a web browser. The Dashboard on the AWS, uh, SageMaker console shows all the different areas and all the different services that can accessed through the web interface. T**he tools and services can also be managed from an external machine using the Amazon SageMaker CLI or command line interface.** There are a number of different wrappers, such as Python in this case, that can make accessing the SageMaker tools and services quite easy. There are a number of wrappers, such as the Python CLI, that make accessing the SageMaker tools and services quite easy. In this example, I'll use **AWS SageMaker and LS to** list all the services and you will see that all the services that are available through the web interface are also available here. So the things such as listing the algorithms, creating jobs, and even starting and stopping jobs. The Amazon SageMaker platform uses a number of freely available Python packages, many of which are open source, and they allow for customization of code and importation of many different libraries. In this example here, I've created a very simple function and it can be used and imported as part of the SageMaker workflow. So this means that many different tools that you build yourself or that you find on the net can be imported and used in the standard Python environment. These custom tools can be inserted as part of the data processing pipeline, the model training, or the API inference steps. The customization possibilities are infinite and libraries can easily be swapped in and out. Efficiency. The Amazon SageMaker machine learning libraries have been developed specifically to run optimally on AWS hardware and in AWS environments. Since the resources are usually priced based on the time of use, this means that for certain functions training times and training costs can be reduced. There are also efficiency gains based on the centralization of all the tools in a single location. Instead of having model hosting and training in separate distinctive environments all this information can be sourced locally from the SageMaker Dashboard and you can see that on the left hand side here. The Notebook Instances, Training, and Inference areas are all in one place.

### **Interacting with Amazon SageMaker**

Selecting transcript lines in this section will navigate to timestamp in the video

- The web interface for Amazon SageMaker is the simplest introduction to the scope of the platform. There is a choice to interact with SageMaker through this GUI or to use a separate remote package, the CLI which we showed before. For this part of the course we can start walking through the web interface. First, go to aws.amazon.com. The next step is to go to My Account and then to AWS Management Console. From here log in with your password. If you have your own AWS account you'll be able to see this AWS Management Console. If you have a separate account or if someone else is the administrator you may need to ask them for access. In order to access SageMaker you can search for SageMaker under services or find it in the list below. Clicking here takes you to the Amazon SageMaker web interface where you can access all of the functions. So first of all the main area where a lot of the code is written and a lot of the data is manipulated with is in the Notebook section. So the Notebook section here will mainly have the instances which have been set up. So these are compute instances which are running the Jupyter Notebooks that can be interacted with and you can create these to be any size of compute power you need. And they will be used essentially to run different queries, run different functions to prepare the data and start the training process. The Training section of the interface allows for the management and review of different training jobs that are started from the notebook instances. This allows the training models to be monitored and that's under Training jobs, so you can check when you've scheduled models to start training, if they actually successfully trained, how long it took, were they using the right resources, et cetera, et cetera. And also allow for different settings to be altered by the Hyperparameter tuning. So you can set up specific jobs to test different settings for model training. The third main area that we'll focus on is the Inference. So the Inference section will allow for the management of different models that have been trained from the training job area and also the configurations of different endpoints so the APIs or the inference points. This allows both the API service to be managed and also batch transfer on jobs to be run against the models that have been created.

### **Data analysis tools**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Data analysis tools. Data analysis tools allow for the importing, reviewing, and querying of different datasets. In this example, I'll walk through creating a Jupyter Notebook instance in Amazon SageMaker, and creating all the settings in order to import, review, and query different datasets. To start with creating a notebook instance, we'll go over to the notebook instances area of the Amazon SageMaker web interface. You can see there are some that've already been set up, one that's currently in service, and one that has been stopped. But to create a new notebook instance, click on the button Create notebook instance. You can give your notebook instance a name. In this case I'll call it SagemakerExample as one word. You can choose the instance type, or the size, though in this case, because we're not doing any training or large amounts of data manipulation on this instance, we can keep it as the smallest instance type, so that, in this case, is medium. In terms of elastic inference, in terms of additional configurations, there are some other options available, but for the scope of this example, we'll just keep it quite simple and keep the default. In terms of permissions and encryption, again, for this example, we'll keep it quite simple, but we already do have a role that has been created, and we'll use that iAM role for this analysis. One customization that we will use is the use of a Git repository. So for this example, for this course, we'll be using the Amazon SageMaker examples that've been prepared. I essentially thought it would be quite useful to use an example that has been spoken about before previously in terms of the data challenge, and the dataset. So if you are looking to understand the data further, there'd be a lot of other online resources that could be used. But for this course specifically, I was using that as a base to go backwards and forwards and query the data, and use it specifically on SageMaker. So, in order to access that data, you can specify that repo here by clicking clone a public Git repository to this notebook instance only. Click on this area again, and then paste in the URL. The next step is to create the notebook instance. So, the request has been successful, but as you can see here on the Sagemaker example, the status is still pending. This might take a few minutes to update. By clicking on the name of the notebook instance, you can see the details of that instance, and also if it's been created yet. When the instance has been created, and when it's ready to be accessed, the status will be updated to InService. From here, we can open Jupyter, so, Jupyter is the Python notebook, the Python web interface that will allow you to access different datasets and perform different analysis functions. Clicking on that link opens up a new tab, and you'll be able to see all the folders that've been copied over from that Git function, so everything from the Amazon SageMaker example folder has already been prepared here on that new instance that's been created. If you can click on the folder introduction to applying machine learning, and then the xbgoost\_customer\_churn folder, there'll be an xgboost\_customer\_churn notebook that can be used as a base for this course. It's also possible in this folder on the right hand side to click new, and you can create your own Python notebook using a number of these configurations and kernel setups. In this case, we can just create a basic conda\_python3 notebook, which will use the Python3 environment. When this notebook was created, the kernel could not be found. So we need to specify which kernel to use. The kernel is essentially the Python executable that will be used to run all the code that is written in the cells. In this case, we will set it to the conda\_python3. As you can see, it started up, and in this interface, essentially how the Jupyter notebooks work is that code is written in each cell, and then that can be executed by either running the Run command, or holding Shift and pressing Return. In this case, I'll write a simple print statement to say hello world, and then I'll press the Run button, which will print it out straight into the notebook. I can also hold Shift and press Return, and it will execute there as well. For the analysis that we'll be performing as part of this course, we'll be using three main libraries outside of the SageMaker toolkit. Those are Pandas, NumPy, and Matplotlib.

### **Download and import data**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Data analysis process. There are three main steps we'll walk through when looking at this churn dataset example. The first step will be downloading the data from the public web, the second step will be importing the data and we'll use the pandas Python library for that and the third step will be investigating the data. So, on the Notebook Instances screen, open up the Notebook instance that was created earlier by clicking on Open Jupyter Notebook and that will take you to the directory structure that was copied over from the GitHub repository earlier. If you could click on introduction\_to\_applying\_machine\_learning and then xgboost\_customer\_churn. From here open up the xgboost\_customer\_churn notebook. So, this is a tutorial that's been provided for Amazon SageMaker that goes through how do you train a model using XGBoost and then how to deploy that within the SageMaker environment. I thought this would be a good example to run through as we can do a bit of customization and I can talk through all the different steps to add a bit more color and a bit more flavor to what's happening. So, in terms of accessing the data, so, the data is actually available on the public web as I mentioned before but the code here actually copies it over to an S3 bucket that you have. So, S3 is a file hosting service that's part of AWS and if you have the right rights, you can create an S3 bucket to store the data for this analysis. In this case, I've already created a testsmdata S3 bucker and I'll use that to host this data. So, in the cell where the bucket is being assigned, you type in testsmdata or whatever name that you've called your S3 bucket. You can then hold Shift and press Return to execute the cell. But we'll also import boto3 which will allow us to interact with different AWS services, re for regular expressions and the SageMaker library to get the role, the execution role that we defined in an earlier step. We'll also import the bulk of the libraries that we'll use for the data analysis. The top three libraries, pandas is responsible for all the DataFrame or dataset manipulation. Numpy is the number of numerical functions and matplotlib will be for the plotting and the displays of different charts and graphs. In this case, the example has a number of functions that download the dataset, the churn dataset that we'll be looking at and the exclamation mark at the front of each of these cells just means that these steps will be executed in Bash. The wget command here will download the dataset from the Data Miner Consultant website and the unzip command will unzip it. As you can see here, all the datasets have been extracted. Now in the earlier cell, we imported the pandas library and gave it the alias pd, so this pd object will allow us to use all the pandas functions to import and manipulate datasets. One of those functions is read\_csv and we'll point it towards the churn dataset and assign that to the churn object. So, this read\_csv function will import a CSV file into a pandas DataFrame. We'll also set the pd option to display the max columns as 500, so we'll be able to see every single column.

### **Data visualization: Categories**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Data visualization tools. So, which tools? So, matplotlib is a powerful library to visualize different elements of a data set in Python. We'll be using matplotlib to look at some of these elements in the churn data set that we imported earlier in a more visual manner. These libraries, these functions, they're not confined to SageMaker in any way and there are many, many more online tutorials that go into a lot more detail and depth about how to use them but for these examples, we'll go through a few simple ways to look at the churn data set. From the previous example, we've imported the churn dataset and we now have it as a pandas DataFrame object. In the new cell, this object can be manipulated like shown before by running the head function. So, what we can also do is by looking at the state series here, the column with the state value, perhaps we'd like to not just see the number of unique states there are but what's the frequency of these different states. In order to look at the frequency, we will isolate the state series from the churn DataFrame and run the function value\_counts. Now this will count how many times a value occurs within that series which you can see here. It's showing a count of all the different state values. Now, from this function, we can then chain the plot function for pandas which will create a plot of these counts. Now straight away you can see that it's automatically chosen a line graph. You can see on the x-axis at the bottom that there are no labels to what these values are. You can on the y-axis that there is a count but this visualization doesn't really have any meaning because we haven't explicitly added any settings. In the plot function, we can specify the argument kind equals bar and instead of a line graph, we'll create a bar graph. Now, in this example, because the state series is a categorical value as in it's not a continuous number, it has a number of discreet groups, it's now automatically plotted the names on the x-axis and the frequency on the y-axis. Unfortunately there are too many groups here to actually extract any meaning from what's happening. So, what we can do in this case, if we were just looking to find out what are the top 10 states in a visualization, we can run the function head 10 and that will just show the top 10. If we were looking at the bottom 10 states that occur in this DataFrame, we can just copy and paste the same function but instead of head, we can write in tail.

### **Data visualization: Numerical**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Using bar plots are a great way to visualize categorical counts, so in the example before, so the number of states that are in the data frame. But for continuous variable, or numerical values, a better approach is to create a histogram. So, the way that's done is, earlier we imported a matplotlib object as plt, so we'll create a figure within the cell, we'll set the axis, we'll take a variable, or a series from the churn data frame call Day Calls, so this is the number of day calls, and we'll plot it as a histogram, with the argument hist. Then we'll just run this, and this will create a histogram based on the frequency. So, in this case we don't need to specify the value count's function, because this is not a discrete value, that's continuous, but the histogram function creates the distribution for you, and visualizes it here. Now, if we'd like to plot two distributions on the same chart, we can then add, we'll use the Night Calls Count, Night Calls series, and we will plot this as a histogram as well. And because we had this ax object, this, we will also set the ylabel as just the Frequency. So, running this, as you can see, the ylabel has been set to Frequency, and the two distributions have been plotted on the same chart, using the same axis. Again, this is a great approach when you're looking value by value, as in series by series, or variable by variable, in a Pandas data frame, but Pandas also provides a number of tools to plot all of the numeric values, all the categoric values, in one simple command. And this can be shown by, again, using the churn object, this time running the histogram function. We will need to add a few arguments, one of those being the number of bins. So, because there are all continuous variables, we're not using value counts to get a count of how many there are. Instead, using bins, we can specify, we'd like to group this into 30 groups, and then plot that from there. sharey will mean that it'll have the same y-axis, so everything will look consistent, and figsize takes in a set, 10, 10, just to be specific about how large the visualization is. Then running this. So, this will plot all the continuous variables in one go, instead of running them one by one, or creating a loop to visualize each of them.

### **Data summary tools**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In the previous example we walked through a number of simple ways to create visualizations in a Jupyter Notebook but you can also create a number of data summary outputs as well, so these are raw summaries about the pandas DataFrame. And the main two tools that are used is one is the describe function, so again using the churn DataFrame that was imported earlier, you can run the describe function straight from it and that will take all the numerical values and generate a count, create some basic statistics, the mean, standard deviation, min/max and interquartile ranges and you can review these and see how the data has been shaped. Sometimes I find depending on the number of columns that it can be quite difficult to see what's happening from this point of view, so after the describe function, if you run capital T, that will transpose, so you can start scrolling down, so the first check that I like to look at is are the counts all the same? Usually if the counts are different, that means that there's a null or other type of infinity value which may be present which might change the way you interpret the data. You can also see there were many zeros. You can have a quick check at the mean and the median or the 50% point to see how different they are, if they're quite similar, that means that the distribution is quite even but if they're quite different, that means they may be skewed but this is just a quick function to run to browse through, get an understanding about what the continuous variables look like in a pandas DataFrame. We haven't gone into detail about the meaning of the DataFrame at this point. We're really just applying a number of different functions as an introduction but the point is to predict if a customer does churn, so this is shown in the churn series as part of the DataFrame. And it's given a true or a false value. The pandas library has a crosstab function which allows you to create a count of two different variables that's in the pandas DataFrame. So, in this case, we can take the Churn output which is a true or a false and compare that to the VMail plan series which is if they have a voicemail plan or not. Running this function creates a nice little table that shows the counts of each of those values. This can be quite useful in describing the data without creating different visualizations, without going into too much detail. You can easily run quick little checks about assumptions that you have about the DataFrame.

### **Challenge: Describe a dataset**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat synthesizer music) - [Instructor] For this challenge, I'd like you to use Pandas again, but this time, import a new CSV file from the Data sets folder. If you could please import the cars.csv data file and then create a bar chart of the frequency of the year of cars from that imported DataFrame. And just a quick hint, as well, watch out for the column names because they can be a little bit misleading. Finally, compare the average values to the median values for all the numeric values in the cars dataset.

### **Solution: Describe a dataset**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] Here's my solution to the problem. So, first of all, I'll create a few extra cells in the Jupyter Notebook just in order two write some code and keep it in line with the different problems. First of all, I'll import pandas as pd and by using that pd object, I can then start interacting with the pandas library. So, first of all, I'll track down what the actual file path is for the cars.csv file. As I mentioned earlier, you can use exclamation point to run commands in shell. And I can see that the folder name is Data sets. If there's a space in the folder name, a back slash, we just escape that and within that folder there's a cars.txt file. Now to import the cars.txt, I'll create a cars\_df object and I'll using the pandas library I'll write pd.read\_csv. Single quotes. Data sets/cars.txt. So, that will import that txt file into the cars\_df object and that will now be a pandas DataFrame, so if I run type around that, it will tell me it's a pandas DataFrame. For the next step, creating a bar chart, I'll look at what the names of the columns are in this DataFrame, so running cars\_df.columns will give me a list of the column names. Now I can see here that the year field is here but it actually has a space in front of it. And this sometimes happens when you're importing different, especially CSV files because there's no set typing and there's no set rules about what to name columns or different formats, so just be mindful that if you'd like to reference this without changing it, you need to include the space at the start. So, first thing I will do is I'll isolate that and I'll just do a bit of a check. Yeah, so that's showing different years, and I can see that's an integer. So, in order to create a bar chart, the quickest way to get frequency is to run value\_counts as shown before and then in order to create the bar chart running the plot function with the argument kind equals bar. Now you can see when I've run this it sets some output but it hasn't actually displayed the chart in line. As mentioned earlier, in order to that, if you write percentage like matplotlib inline, you'll create the chart right there. Now you only need to run matplotlib inline once for the whole Notebook and then it'll remember to display the results in line. Now in order to compare the average values to the median values for all the numeric values in the cars dataset, we'll use the describe function. So, first, cars\_df.describe. Now this would generate different statistics for all the numerical values in the cars dataset. Perhaps you'd also like to see it with the feature names on the left-hand side. In order to that, you run the same function and then transpose. You can scroll down and look at all the feature names and compare the mean value which is the average to the 50% percentile which is the median.

### **Cleaning up the data**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] For the next part of the course, I will look at preparing the data in order to be used for modeling, creating the modeling dataset and then starting the training of the model. Now I'm quite mindful that I haven't gone into much detail about this churn dataset and really haven't explained what we're trying to predict. Although the actual details of the data, how it was collected, its internal meaning is out of scope for this course, understanding data or having a firm grasp of what different data fields mean in any machine learning project is always in scope, in fact, it's probably the most important aspect. I'd probably say most of the time it's more important the actual modeling method itself. And the reason for that is, and as you'll see, it's quite simple to create a data pipeline and to train a model, but the model will never tell you the meaning behind what it's actually coming up with. It'll never really tell you how you need to use it, where you can use it, where you can't use it. It'll never give you that insight and that only comes from understanding the underlying data and the environment in which you're using the model in. All of that being said, these next steps will step up the model in order to create a dataset that can be used for training. Now the phone variable in the churn pandas DataFrame is actually an identifier rather than a feature or a variable of meaning. So, we'll actually remove that before we start creating the modeling dataset. The area code value, although it is numerical as well, its meaning is actually closer to a category than an actual continuous variable, so we'll run this function here to convert it to an object. This next loop here as shown in a previous presentation actually creates cross tabs for each of the categorical variables and displays how often the churn value is seen. So, again the churn is the outcome variable that we're looking to predict with this model and this crosstab function can give a simple data summary across all the categorical variables to give a bit of an understanding at which variables could be highly predictive. For each of the continuous variables, we'll actually plot a histogram as shown before. The next step that we'll run through is actually removing a number of the features that are highly correlated to each other. So, having correlated variables in a model can 'cause a few issues primarily because they are basically doing the same thing twice or multiple times depending on how often they appear in the DataFrame. So, this drop function will remove a number of the features or variables that are too correlated to each other.

### **Preparing the model training set**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that the data frame has been cleaned and we've been able to remove a number of correlated variables, it's time to prepare the data for modeling. So one way that we'll do that is by converting all the categorical variables into one-hot encoding or dummy variables. So the way this is done is using the pandas object, we run the get\_dummies function on the churn data frame and assign that to the model\_data object. When we run the model\_data.head function, the data frame, we can preview the first 10 rows. You will see that for the continuous variables such as Account Length and Day Mins, they are intact, there are no changes. But for variables such as State, that has been split out into a number of different columns for each categorical value. And now, instead of having the character value AK, it will have a binary yes or no if it equals that value. In this case, for State\_AL, you can see that this appears on the sixth row, so that has a value of AL. The reason why it's been split into this binary format is because of the type of tree algorithm we'll be using and the way that this information can be better understood by the model and cut points can be easily allocated. Now, this function here, it actually removes and brings back the churn outcome value. So something that I found that was quite interesting about this dataset is that the outcome variable that we're looking to use isn't actually a boolean value. It isn't true or false necessarily. It's actually a string value that says true or false. And so now, we run the pandas concat function to include the churn value which has been converted into a dummy and we remove the churn value that's false. So we actually only really need one outcome variable. And if we include the true and the false, they're basically the inverse of each other, so we only really need one of them. So having Churn?\_True is the one that we'll keep and we'll use for the target of the model. Now, what's quite interesting about this dataset, the churn dataset, I'll show you. If I take the churn value, just preview the top five, you'll see that it says false, false, false, false. Now, when I first looked at this dataset, I assumed this was a boolean value as in zero, one, true or false. But it's actually, it's an object which is a bit misleading. It's actually a text field. So it's false, full stop, or true, full stop. This means it could take nearly infinite values based on any characters. I can show you what that looks like here. So if I do churn like that, if I take the first instance, first value, if I asked does it equal to false, which is what I thought it did, it will say false. But if I say false, full stop, it will say true. So this is one of the irks of using Python. It's one of the concerns about properly typing fields. It's important to review, especially with the outcome variable. Usually, if it's not in the right format, you'll find a number of errors down the line during training and sometimes they can be quite vague, the feedback that you're getting. So it's always good to check through all the columns and that you have the right type set for each value. The next step is to split the model data object into the training validation and test data. And that's done here with this one step here. Using the numpy object that we created initially, we'll split a random number that we can assign, so the random seed, we can set here. We'll split it into those three datasets. Then we'll take each of them and using the to\_csv function, we'll save them as CSV files on the local drive. We'll remove the header and remove the index, and save them locally. And then using the boto3 object that we initialized at the beginning, we'll copy them up to the S3 bucket that we created, again at the beginning. The reason why we copy these up to S3 is so that when we create the model instance for training, we can access that data instead of having to connect to this data preparation instance.

### **Model training**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that we've prepared the model data sets, the two CSV files, and uploaded them to S3, we'll look to start creating the training instance and start actually training the model. The way that we do that is initially using the SageMaker package, we import the get\_image\_uri function, and we use that to (mumbles) using boto 3, getting the XGBoost container. This container will contain all the code that's required for the XGBoost model to start training on a separate instance. Because we've already created the train and validation CSV data sets, we'll create objects that reference those using the sagemaker.s3\_input function, which we've done here. The next cell arguably contains the bulk of the training functions. These next four functions will create a SageMaker session, will create an estimator object, taking the container that we created before, the role that we created initially, and all the other values to actually create an XGB object which will allow us to start training the model. The third function will set all the hyperparameters for your XGBoost. These hyperparameters that have been chosen here as an example, there is a link to XGBoost's GitHub page which goes into far more detail. There's also a great original paper on XGBoost as well that can be found through archive. That will go through exactly the reasons why certain hyperparameters are chosen. In my experience, I always had a lot of trouble finding why certain hyperparameters were set or how to pick the best ones for the problems that I was facing. The best solution that I found was to look for different research, especially in the areas that I was approaching and trying to understand. There's not one size fits all. Based on what your needs are, XGBoost may not be a suitable modeling approach at all as well. I think it's important to look at research that's available and always adjust what you're doing. The last function here, the fit function, will take the two data sets that have been declared earlier and start the actual training of the model. When I run this cell, you will see that the SageMaker training job has started, it's given an ID based on the date, and the message Starting the training job entered there. Going to the SageMaker GUI, going to the Training Jobs section under Training, you can see that it's created this job here. The status is currently InProgress. By clicking on the link, you can get updates about what's happening and see the history of the training job.

### **Checking model training results**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In the previous step we created a container for xgboost, we assigned resources to start training a model, and we kicked off the training process with a training validation starter set. So now on the Amazon SageMaker web interface we've gone over to the training jobs, we've found the latest training job and we can see that it's been deleted and it's taken three minutes. By clicking on this we can see the duration, different details, and also the history about what happened during the training. We can see that the instance was prepared, data was downloaded, the training image, a container was downloaded, the training was started. Then the model was updated and the training was complete. This model has been now uploaded here to the inference section as you can see here. So the output in the Jupiter Notebook has also been updated based on all the training steps and this mirrors the history that's found in the web interface. The red text shown here is model updates from the xgboost container showing different updates to the training throughout the process. At the very end of the log it mirrors what's shown in the history page on the SageMaker web interface and also shows the billable seconds. So in this case it took 44 billable seconds to train this model.

### **Challenge: Train a basic model**

Selecting transcript lines in this section will navigate to timestamp in the video

(bright music) - [Instructor] For the next challenge, I'd like you to use the same process as above, but instead of using XGBoost, try training a logistic regression algorithm. So, this will require a little bit of research. It will require a new type of container, and that container may not necessarily be called logistic regression, but it is part of the SageMaker platform. If you can locate the container and then connect the data and train the new model, you'll be successful with this challenge.

### **Solution: Train a basic model**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] For the solution for this challenge, it does require a little bit of research to be able to find out exactly how to train a logistic regression model on Amazon SageMaker. Fortunately the documentation has heaps of accessible information and it's quite helpful in terms of understanding how to put it all together. So, the first step is to review how the container was created initially and this cell here contains the information where we're able to create a container from the xgboost image. So, by copying that here, I might rename the container to container\_logistic and now I need to locate exactly what the name is of the image for logistic regression. Now in the Amazon SageMaker documentation, linear logistic regression actually comes under the Linear Learner Algorithm. So, instead of using xgboost, we'll change that image name to linear-learner and run that command. So, the next step is to copy these commands from this cell here. First of all, this creates the SageMaker session. Instead of creating xgb object, let's call that logistic and instead of the container, let's change that to container\_logistic. The role will stay the same and all the other settings will stay the same. The next step is to assign that hyperparameters to the logistic function. Now the hyperparameters are quite different compared to the xgboost algorithm. In order to find out what they are, again we'll refer to the documentation under Linear Learner Hyperparameters. You'll notice straight away that a number of them are required. This means these arguments need to be passed in order to train the model. For logistic regression, since it's a binary outcome, the number of classes is important since we're only looking to predict a one or a zero. The three required hyperparameters are the feature dimensions, so the number of columns that are fed in, the number of classes or the predictor type, and the number of epochs or the number of cycles for training. Going back to the Notebook, we can set those with logistic.set\_hyperparameters. The first one being feature\_dim, so this is the number of columns that is in the modeled DataFrame. So, initially so this is 74. Then the predictor type. Since we're looking at a binary classifier, we specify binary\_classifier and the number of epochs. This can be whatever you choose, for something that's quite quick, 10 is acceptable but again with hyperparameters, it all depends on the dataset, it all depends on what you're looking to do. For this example, 10 will be done quite quickly. Now the next step to actually start training the model is to run the fit command. And the fit command is exactly the same as it was for xgboost. But obviously now instead of xgb, we'll execute that on the logistic object. And now the training job's started and you'll be able to follow updates both here in the Notebook and on the SageMaker platform.

### **Deploy trained model**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that a model has been successfully trained and saved to the Amazon SageMaker platform, I can show you how simple it is to start hosting that model. Now usually the hosting of a model is done by completely separate data science team, that uses different technologies, but in this case with Amazon SageMaker it's part of the same package, same library. So using the xgb object that we created earlier we'll run the deploy function. We'll deploy the model on one instance and we'll define that instance's size. So in this case an ml.m4.xlarge. So running this function you can see the log output creating the model, creating the endpoint. You can see that the progress bar is clicking along the bottom here. But you can also get updates by going to the Amazon SageMaker web interface. By clicking on endpoints you will see that the status is set to creating, and it is currently going through the process of creating that API endpoint. By clicking on the name you can get updates about what's happening.

### **Test deployed model for single record**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In the previous steps, we have trained a model, and we've also deployed that model. And that model is referenced by the xgb\_predictor object that we still have in the Jupyter Notebook here. So, in the next example, I'll run through how to use that as a tool to get an inference for one row out of our data set. So, first of all, we need to set the type of content, the serializer, so the content is text/csv, that it's expecting. Update the object to have those settings. I'll then reference the test\_data, Panda's dataframe that we created earlier on. So we created a train data and a validation data dataframe, which we use to train the model, but we'll use this for validation. So the shape of this model, so shape meaning the number of rows and the number of columns, is 334 by 70. What we'll look to do is we'll grab the first row using the head function, one. That will extract the first row from the dataframe. And then we'll convert it into a NumPy array. So the NumPy array will be purely just the values that are included, all the float values in an array, and it'll be shown like this. Now, I'll open up the test\_data head one example again. Now, this dataframe has the actual outcome included. It's the first column here. We don't actually need the outcome to infer the output of this row. We only need the features or the variables. So set as matrix. I'll specify one semicolon, so that will only include everything from 186 to the end, like this. Then, I'll take the xgb\_predictor object that's outlined up here. And I'll run the predict function on this array. Now the output here is 0.0155. So that's the probability that this row or this customer will churn. So this process is essentially, one by one, getting the probability or the inferred outcome of each of the data rows in the test\_data dataframe.

### **Test deployed model for multiple records**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that we've been able to use the xgb\_predictor object to create inferences on a single row from the test\_data DataFrame, we can write a function to loop through each of the rows and generate the inference. So, this predict function goes through each of the rows of the test\_data DataFrame and runs the xgb\_predictor predict function against each value. It then returns those into a list called predictions. So, when we run this, we'll have a predictions object that's been created. I can run a len function on predictions and see that there's 334 predictions or if I get the shape of the test\_data DataFrame, we will see that this also has 334, so essentially the predictions value is a list of all of the predictions for each row in the test\_data DataFrame. So, now what I can do is I can start comparing what was inferred to what was true, what really happened. In order to do that, we can use the pandas crosstab function that we used earlier and feed in the inferred and the actual, so the actual from the test\_data frame is Churn\_True. The prediction's value is that probability that they will churn. Now in the predictions object, it's a list of probabilities I can show here. So, I grab zero. It's a probability of 1.5%. If I use the NumPy library, and use the round function. This will round to zero or one. So, essentially if it is under 50% chance, we're looking to round to zero and if it's over 50%, we're looking to round to one and then we'll create a crosstab showing the zeros verus the zeros and the ones versus the ones. I'll show you how that works. So, here we type np.round again to the predictions list and run that. So, this is showing a crosstab or a confusion matrix this is actually called from the actual churn versus true versus the predicted. And what this table is saying is that there were 282 observations that were actually zero, that we predicted were zero and there were 39 that were actually one that we predicted were one. There four that were actually zero that we predicted were one, and there was nine that was actually one that we predicted was zero.

### **Challenge: Transfer model to server**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat electronic music) - [Instructor] For the next challenge, I'd like you to use the Amazon Sagemaker Batch Transform Function. Now, previously, we created a model, and we were able to use that to make single record inferences and using a loop, were able to generate inferences on multiple records. But for this challenge, I'd like to see you use the actual Batch Transform Function, which it built to make inferences on millions of records on the train.csv dataset. This will require a bit of searching through the Sagemaker documentation to understand exactly how to create a Batch Transform job, but the progress of the job and the results can all be tracked in the Sagemaker dashboard.

### **Solution: Transfer model to server**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] For the solution for this challenge, you will need to look through the Amazon SageMaker documentation to find out how to execute the batch transform function. Fortunately it's quite easy to read through and there's lots of useful code examples. Under the Deployment section of the documentation you can find information about how to interact with the Batch Transform API. You can also see that this is a code example that uses the Batch Transform transformer object to be able to get inferences on a CSV file. In our Notebook, we've already set a number of these variables, so we only just need the information around the transformer object and the transform function. I'll copy that section here and then I'll paste it back in our Notebook. We've already imported SageMaker, we've already defined a model and we've already defined an input, but we haven't defined an output path yet. So, I'll set up those variables one by one. The first variable that we need is the model name. This can be located on the Amazon SageMaker dashboard. I'll highlight and copy the latest name of the XGBoost algorithm that we trained and I'll paste that back in the Notebook. The next variable that we need is an output location. So, we haven't created this yet but we can use the same S3 bucket that we created earlier. The S3 bucket location is still stored in the bucket object that we created, so by typing bucket, I'll be able to get the bucket name, I can copy that name and just create an extra key called batchoutput. The final variable that we need is the S3 location of the training.csv file that we used to train the model. That information is still located under the s3\_input\_train variable. Fine, get the config element. I can see that the URL is located here. By copying that line, I can change the input location to that value. And then by holding Shift and Return, it will start the batch transform job. This batch transform job will also update on the SageMaker dashboard.

### **Review the model for accuracy**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So previously we were able to create a confusion matrix showing how many of the rows we were able to predict correctly, and how many we were able to predict incorrectly. Ideally, we're looking for one single matrix to understand how well the model is working on this test starter set, and we can get that with an accuracy measure. Now accuracy's calculated by the number of classifications that were correct, divided by the total number of classifications that were attempted. So what I'll do here is I'll take this panda's data frame, I'll extract the values of the four values that are shown here, and I'll perform that calculation. So first of all, I'll copy this command and I'll save it as a separate data frame. I'll then add the values function in order to just extract the numbers, excluding the index, excluding the column names. This is what it will look like. Now with the results values, I'll isolate the correct classifications and divide that by the total. I'll use the reshape function to convert all the values into one list and then sum that list. So this will show that 96% of the rows were properly classified. Now this accuracy metric is useful, but it's not the end of the story. So there's so many other moving parts, there's so many other options that we could have, or decisions that we could have made to change how this model was trained and how it was deployed. Usually what we find is this is just getting to a point where we can now see the whole process and we can start reviewing it. And one of the strengths of the Amazon Sagemaker platform is the ability to do that really quickly, so you can see that you have all these different Python libraries at your disposal within the notebook. You can see that it's quite easy to create multiple versions of models using different model types, and having a list, a full history of everything that's been trained, and it's really simple to deploy not only single APIs but also batch transform jobs. So you can consistently iterate, you can constantly improve what you're doing, and review. And I think that's really one of the most powerful benefits of this platform, because with any machine learning project, it's never a straight line. It's never from A to B. There's always a lot of back and forth, and this makes it a really useful platform to be able to work on those projects.

### **Next steps**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Martin] As you can see there is not one model that is the right answer and there is no clear linear path for any solution. What's important is having an environment where alteration and change is really simple and also it's important to have confidence, that with some research, new solutions can be presented. From here I'd recommend getting comfortable with the Notebook workflow and also look at changing things up so it suits you. Having a look at different model containers on SageMaker can be really useful, and even creating your own from new papers and new research that comes out. These functions and workflows can help you bring the technical side of the model development process to life, but always keep asking yourself, what is the purpose of this prediction, why am I building a model in the first place, how is the data structured under it and is that data consistent, is it high quality, can I make proper decisions with this. These questions will help improve the decisioning process and ensure that your models are used successfully.