# **Machine Learning with Scikit-Learn**

### **Effective machine learning with scikit-learn**

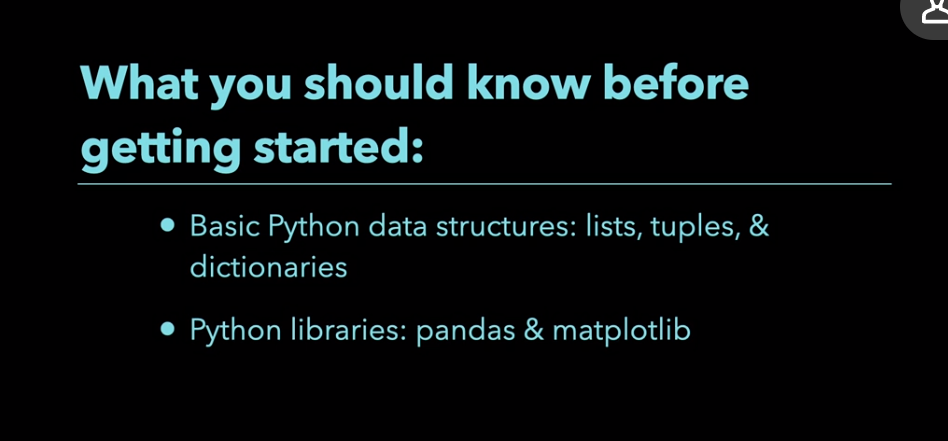
Selecting transcript lines in this section will navigate to timestamp in the video

- Machine learning is transforming industries and it's an exciting time to be in the field. A large amount of machine learning programs are written using open source Python library, Scikit-learn. Scikit-learn provides an easy to use streamlined API that provides efficient versions of a large number of common algorithms. And it makes it easy to train models. My name is Michael Galarnyk. I'm a data scientist, a machine learning instructor, and a blogger about all things data science. I'm also a big fan of Scikit-learn. In this course, I'll show you how to use several machine learning algorithms and when they're appropriate. I'll share with you how you can tune your models to better predict unseen data. So not only make your models better, but also help you understand the strengths and weaknesses of each algorithm. By the end of the course, you'll feel confident and ready to build your own powerful machine learning models using Scikit-learn. So if you're ready to dive in, then let's go.

### **What you should know before you start**

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

- [Instructor] There are a few things I'd like you to be aware of as we get started. First, you should understand basic Python data structures, such as lists, tuples, and dictionaries. Additionally, you should have knowledge of Python libraries such as pandas and matplotlib. If you'd like to know more about these libraries, you can check out my course, Python for Data Visualization. Don't worry if you feel your background knowledge could be better. Throughout the course, I'll offer a host of resources to fill in knowledge gaps. Regardless of your background, you'll still be able to follow the course and learn how to create machine learning algorithms with scikit-learn.



### **Using the exercise files**

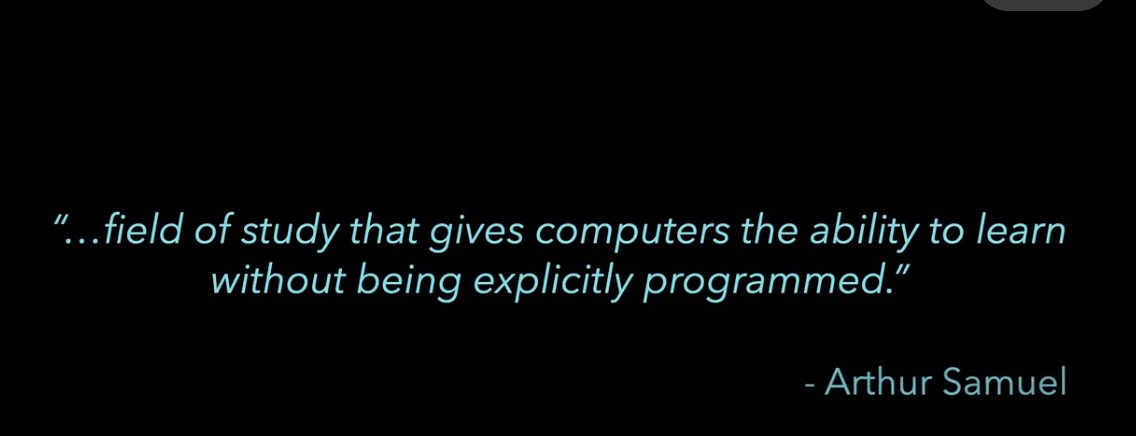
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] To learn the most from this course and to help you follow along, I've included a ZIP file of the contents of this course. You can find these included as the exercise files for this course. Once you've unzipped it, you'll see this folder here. If you click here, you can see the contents of a particular chapter of the course. If this sounds great to you, you can open the different notebooks and follow along.

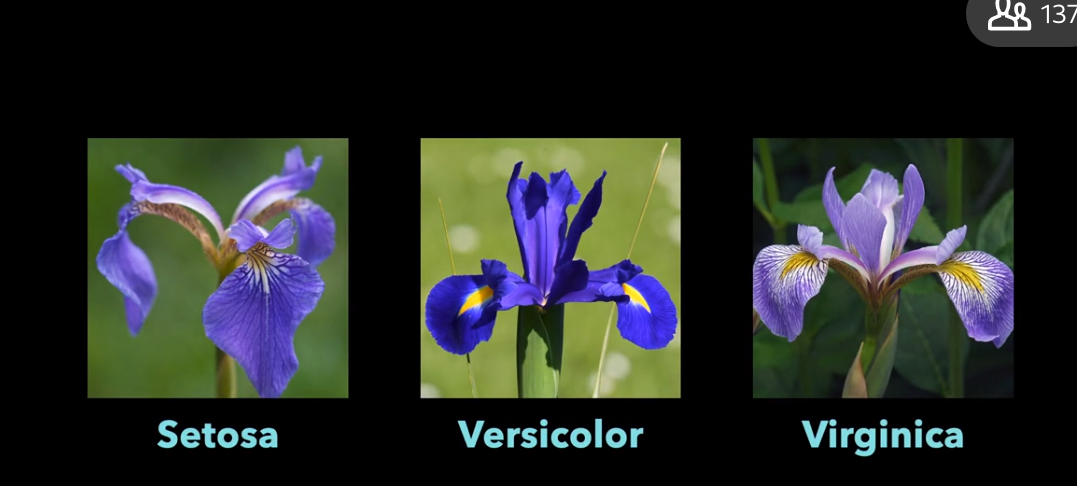
### **What is machine learning?**

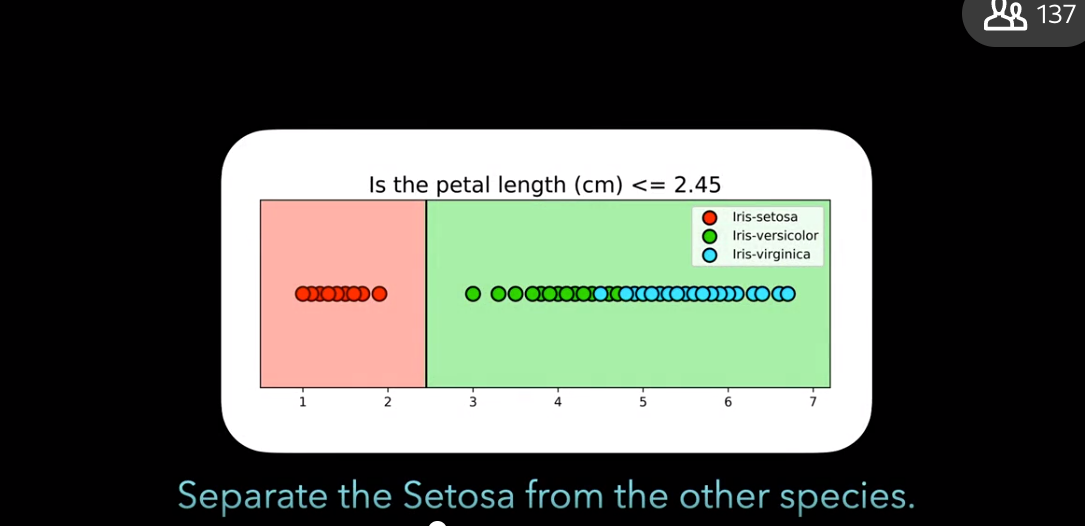
Selecting transcript lines in this section will navigate to timestamp in the video

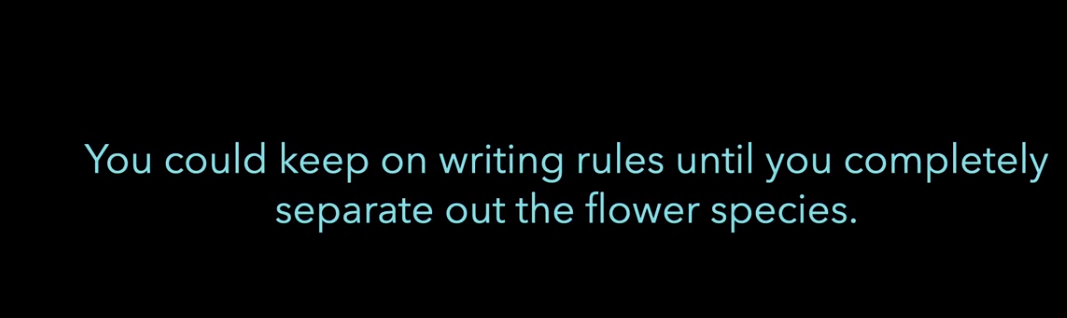
- [Narrator] What is machine learning and why is it useful? You can think of machine learning as a field of study that gives computers the ability to learn from data without being explicitly programmed. Here's an example of why machine learning is useful. Say you have 150 flowers. You're interested in determining which flowers are which flower species based on a feature like their petal length. There are three different flower species: Iris Setosa, Iris Versicolor, and Iris Virginica. From analyzing the data, you can mainly write rules to separate out the flowers from each other. The first rule could separate out the Setosa from the other species. From there, you can make another rule to try and separate out the Versicolor from the Virginica. You're going to keep on writing rules until you completely separate out the flower species from each other. For something more complicated with more features, your program could have a very long list of complex rules. This manual writing of rules might even seem feasible for some applications like flower classification. What if your data changes faster than you can update the rules? Can this approach work for more complicated problems like image or speech recognition? It turns out that these are the sort of problems where humans don't necessarily have a good understanding of how to solve the problem with rule-based approaches. This is where machine learning approaches shine. Machine learning can extract structure from data and solve problems that are normally too difficult or tedious for humans to solve. So that's it. Machine learning is an efficient means of building models from data.

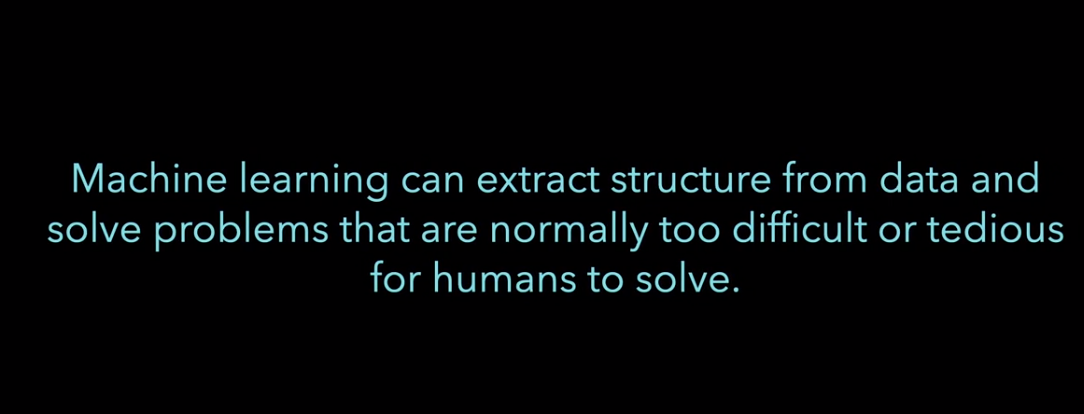








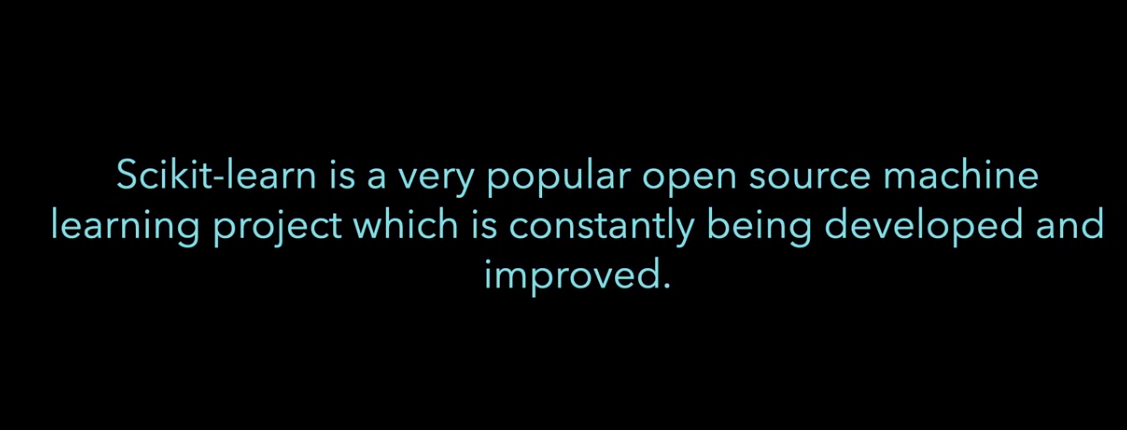




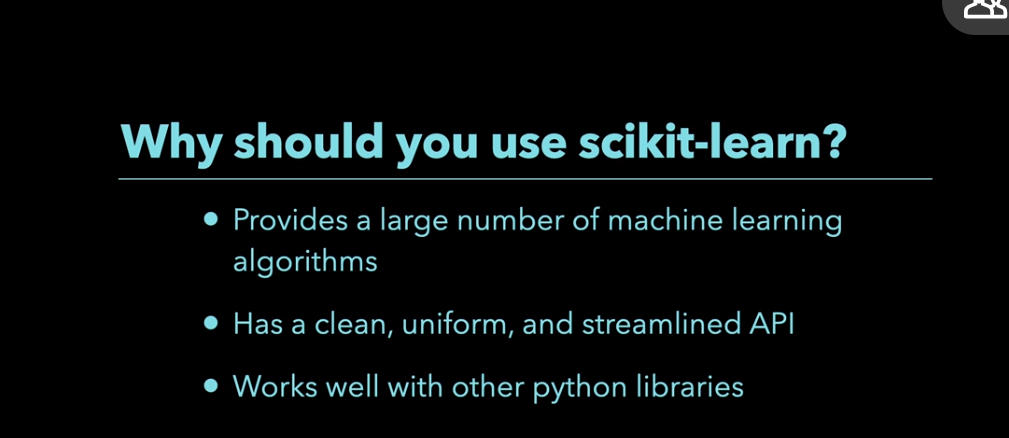
### **Why use scikit-learn for machine learning?**

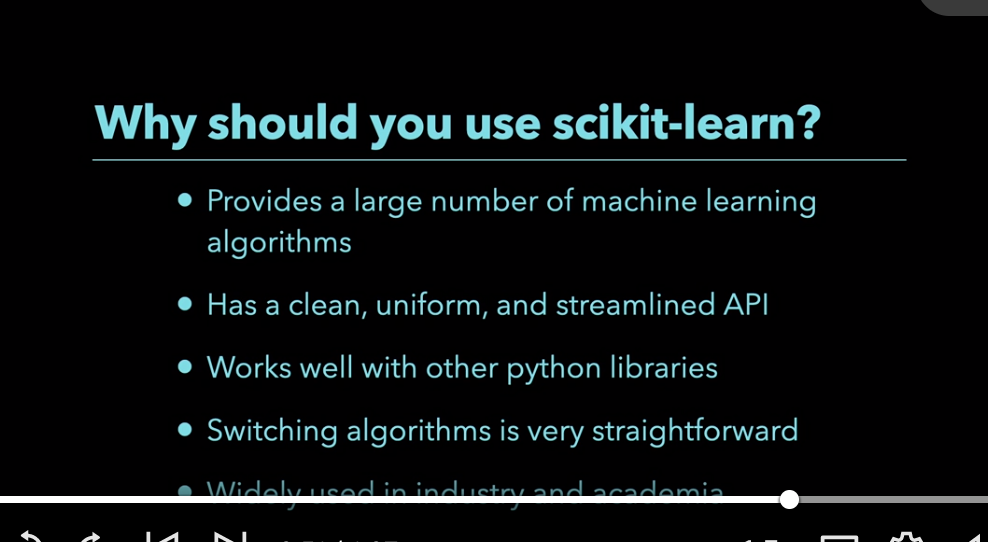
Selecting transcript lines in this section will navigate to timestamp in the video

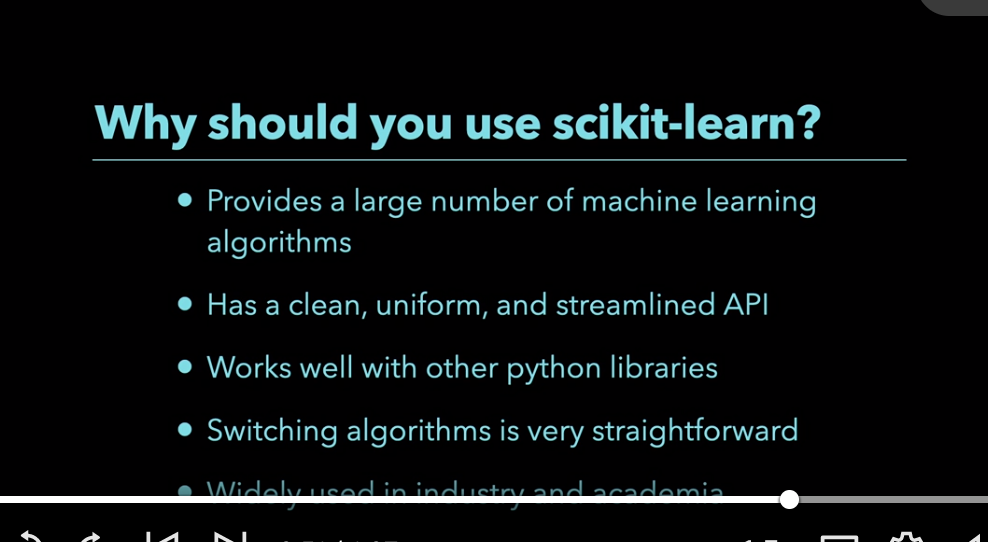
- [Instructor] With a host of machine learning tools and frameworks out there why should you scikit-learn? Scikit-learn is a very popular open source machine learning project which is constantly being developed and improved. In this video, I'll share with you some of the major advantages of scikit-learn. First scikit-learn provides a large number of machine learning algorithms. This is important as programming machine learning algorithms from scratch is not easy task. Most models in scikit-learn also have reasonable default values for hyper-parameters. This means that a machine learning algorithm might work well with little tuning. Next, scikit-learn has a clean uniform, and streamlined API. Scikit-learn also works well with other Python libraries, such as NumPy, Pandas, and Mathplotlib. Also, once you understand the basic use in syntax of scikit-learn for one model, switching to another is very straightforward. Finally, the advantage of the library being popular cannot be overlooked. It's widely used in academia and industry. There are always more scikit-learn tutorials being written. This also means it's easier to get your questions answered on stack overflow. So that's it. Due to its easy to use and popularity, scikit-learn is a great library for machine learning.



Give feedback



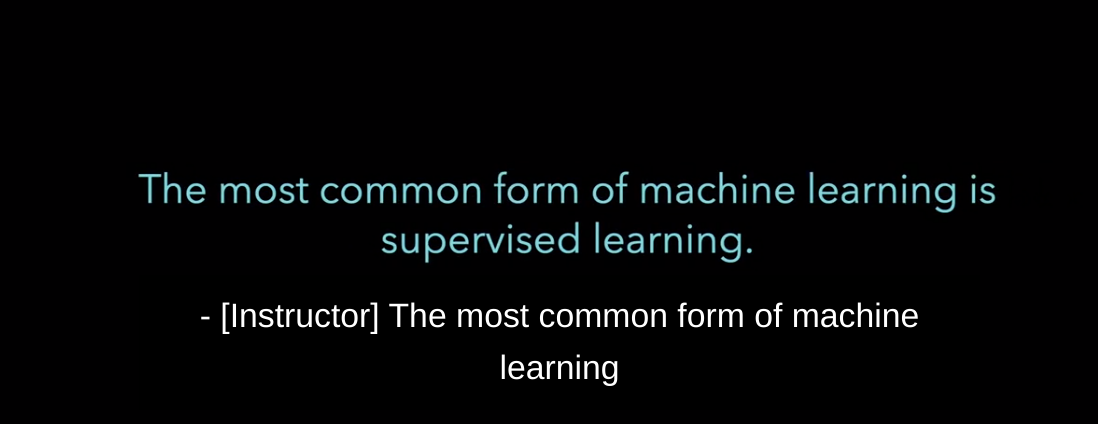


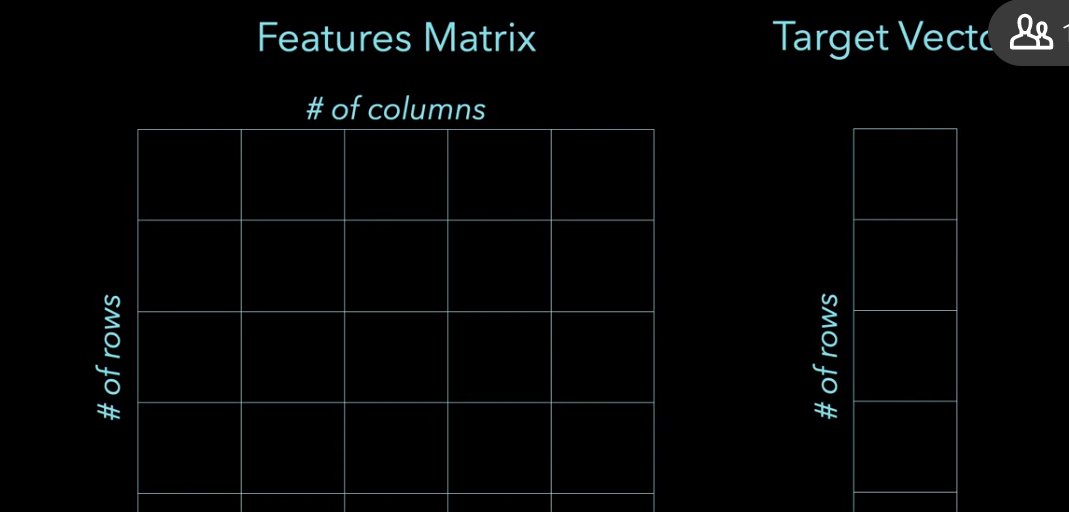


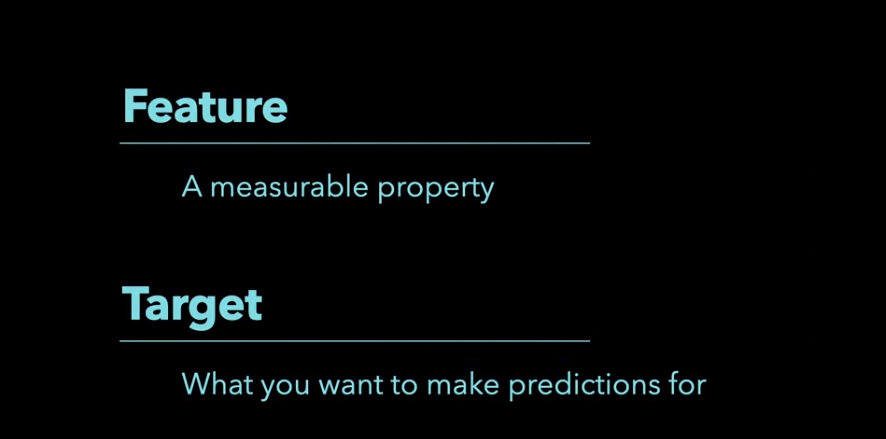
### **What is supervised learning?**

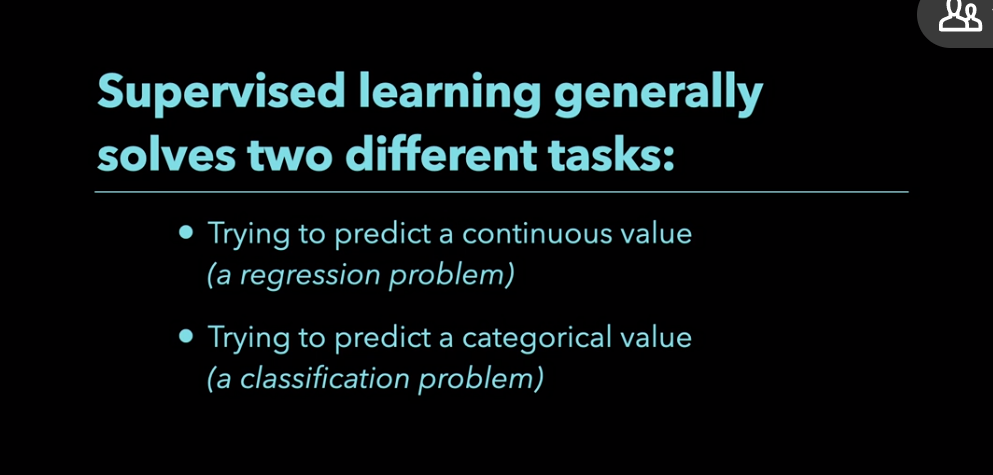
Selecting transcript lines in this section will navigate to timestamp in the video

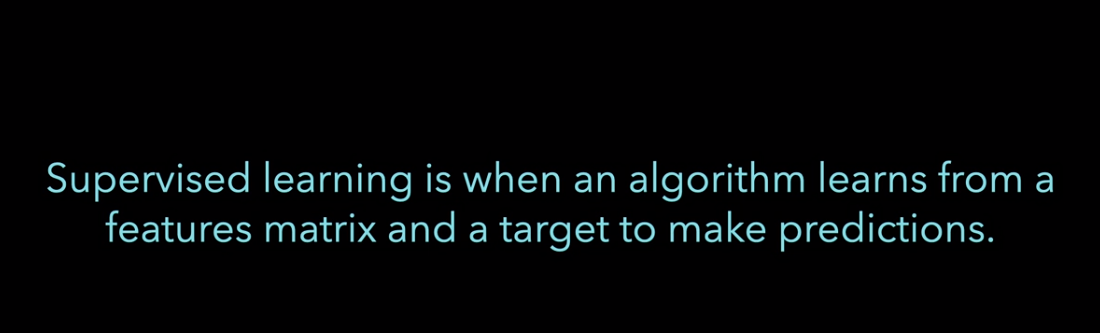
- [Instructor] The most common form of machine learning is supervised learning. In Scikit-Learn, a supervised learning algorithm learns a relationship between your features matrix and your target factor. A feature is a measurable property. A target is typically what you want to make predictions for. Once a model learns a relationship between a features matrix and a target factor, it can make predictions for unseen or future data. Supervised learning can generally be thought of to solve two different types of tasks. The first is when you try to predict a continuous value. This is considered a regression problem. This means that your target factor contains continuous qualities like home prices. The second is when you're trying to predict a categorical value. This is considered a classification problem. This means that your target factor contains categorical values like different flower species. So that's it. Supervised learning is when an algorithm learns from a features matrix and target factor to make predictions.











### **How to format data for scikit-learn**

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

- Scikit-learn is a great library for creating machine learning models from data. **Before you fit a model using scikit-learn, your data has to be in a recognizable format.** S**cikit-learn works well with numeric data that's stored in numpy arrays. Additionally, you can convert your data from objects like pandas dataframes to numpy arrays.** In this video, I'll show you how you can make your data a more acceptable input for scikit-learning. The first thing you have to understand is what scikit-learn expects for features matrices and target vectors**. In scikit-learn, a features matrix is a two dimensional grid of data where rows represent samples and columns represent features. A target vector is usually one dimensional and in the case of supervised learning, what you want to predict from the data.** Let's now see an example of this. The image is a pandas dataframe of the first five rows of the iris dataset. A single flower represents one row of the dataset and the flower measurements are the columns. In this dataset, the species column is what you're trying to predict. Let's now go over an example of how to make your data a more acceptable format. The first thing you have to do is import the libraries. In this case, you'll import matplotlib, pandas, as well as the Iris dataset. This code loads the Iris dataset into a pandas dataframe. From here, you can try to arrange your data into a features matrix and target vector. This is a multiple column panda's dataframe, which will then be converted into a numpy array. You can do this by using the values attribute. One important thing to do is to make sure your numpy array is two dimensional. This is the first dimension, and this is the second dimension. This piece of code is a panda series that you'll then convert to a numpy array. You can do this by using the values attribute. Notice that this is one dimensional. This is okay, as target vectors can be one dimensional. So that's it. Scikit-learn expects data in a particular format.

### **Linear regression using scikit-learn**

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

- [Instructor] How do you create a complex model using scikit-learn? An easy solution is to start with a simple model like linear regression and go from there. In this image, we see a best fit line for a bunch of points. In this video, I'll show you how you can create a linear regression model using scikit-learn. So that more complex models will be easier to create. The first thing you have to do is import the libraries that you want to use. In this case Matplotlib, Pandas, train\_test\_split, as well as the model LinearRegression. From there, you need to load a dataset. This particular data set shows that scikit-learn requires data to be **free of missing values.** The goal of this dataset is to use the feature column X to predict the target column Y. Notice it looks like we have a missing value here. This is really important. As in scikit-learn, you can't have missing values input into a model. The next step is to remove or impute values. If you want to build models with your data, null values are almost never allowed. It's important to always see how many samples have missing values, and for which columns. Let's start by looking at the shape of the data frame. There are 102 rows and two columns. The next thing is to see how many missing values there are in each column. Notice that there's eight missing values for the column Y. You can either remove rows where there's a missing value, or you can fill in missing values. The option used in this notebook is to remove rows with missing values. To do this, you can use the dropna method. And look, there's no more missing values. Notice though, that the shape of the dataframe is different. Before we have 102 samples. And now we only have 94. From here, you can arrange your data into a features matrix and target vector. This code converts the X column to a Numpy array. Notice that this features matrix is two dimensional. You have dimension one, and dimension two. This is really important as to input something into scikit-learn, your features matrix needs to be two-dimensional. What this code is doing is, is it's create a target vector. Notice that this target vector is one-dimensional. Dimension one, and there's no dimension two. Let's now create a linear regression model using scikit-learn. The first step is to import the model that you want to use. In this case, it was already imported earlier in the notebook. So this is not necessary. From here, you can make an instance of the model. This is a place we tune the hyperparameters of a model. In the case of linear regression, you can set fit\_intercept to true or false. This is a really important concept. As more complex models have a lot more you can tune. In the case of linear regression, this is about it. On the left, you have an image of a model where fit\_intercept was equal to true. On the right, you have an image of a model where fit\_intercept was equal to false. Here's the code to make a instance of the model. Note that fit\_intercept in this case is equal to true. From here, you can use the fit method to train the model on the data. What's happening the model's learning the relationship between X and Y. From here, you can predict values of new data. The code here is predicting for one observation. The code here is predicting for multiple. With any model it's important to try to measure model performance. For regression tasks, one metric is R squared. Which is implemented by the score method. This model had a score of roughly .98. Which is actually pretty good. Scikit-learn also allows you to find the equation of the line. You can do this after you'd make an instance of a model, and then fit it on your data. The attribute coaf is essentially your slope. The intercept is your intercept. And here's the equation of the line. This next graph is plotting the best fit linear regression line. This next section is just showing how changing a single hyperparameter value can have a drastic impact on your model performance. By looking at these graphs, it's clear that fit\_intercept equal to true makes a much better model. As the R squared is 0.98. Whereas where fit\_intercept is equal to false, the R squared is 0.86. So that's it. I encourage you to create a linear regression model using scikit-learn. I hope you have a better understanding of how it works.

### **Train test split**

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

- [Instructor] The goal of machine learning is it build a model that performs well on new data. If you have new data, you can see how well your model performs on it. The problem is that you may not have new data, but you can simulate this experience with Scikit Learn's train test split. In this video, I'll show you how train test split works in Scikit Learn. The first thing that you need to know is what is train test split? Here's how that procedure works. The first step is to split your data into two pieces, a training set and a testing set. Typically, about 75% of the data goes to your training set and about 25% of your data goes to the test set. The second step of the process is to train the model on the training set. The final step is to test the model on the testing set and evaluate the performance. To do this in Scikit Learn, you first have to import libraries. The next step is to load a dataset. The dataset using this notebook is the Boston House Price dataset. The goal of this dataset is predict house prices based on features like number of rooms. The next step is to create a features matrix, as well as the target vector. From here, you can create a train test split. The colors in the image indicate which variable, X\_train, X\_test, Y\_train, and Y\_test, the data from the data frame derive from two for a particular train test split. Notice that roughly 75% of the data went to the training set and roughly 25% went to the test set. From here, you can utilize a machine learning model. In this case, it's linear regression. The final part is to measure model performance. By measuring model performance on the test set, you can estimate how well your model is likely to perform on new data. To do this, you can use a score method, just make sure that your inputs are your test set. So that's it. Train test split helps you simulate how well a model would perform on new data.

### **Logistic regression using scikit-learn**

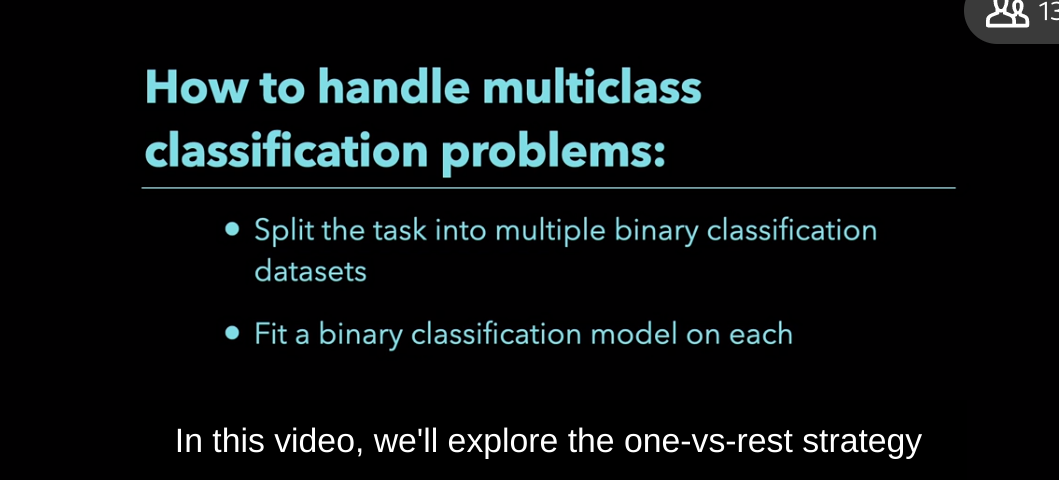
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] How do you create a logistic regression model using scikit-learn? The first thing that you need to know is that despite the name logistic regression contain the word regression, logistic regression is actually a model user classification. Classification models can be used for tasks like classifying flower species or image recognition. All of this of course depends on the availability and quality of your data. Logistic regression has some advantages, model training and predictions are relatively fast, additionally, no tuning is usually needed for the model. Finally, it can perform well with a small number of observations. In this video, I'll share with you how you can create a logistic regression model for binary classification. The first thing that you need to do is import the libraries that you want to use. In this notebook, it's Matplotlib, numpy, seaborn, pandas, as well as train\_test\_split, StandardScaler, and LogisticRegression. While it may seem like a lot of libraries, a lot of model building involves oftentimes data processing, splitting your data up into training and test sets, and actually applying the model itself. The code below loads, a modified version of the Iris dataset, which has two classes, a one in the dataset is a virginica flower and a zero is a versicolor flower. From here, you can split the data into training and test sets. A really important part of this process is to standardize your data. Logistic regression is affected by scale, so you need to scale your features onto unit scale for optimal performance. Unit scale means having a mean of zero and a variance of one for your features. You can utilize scikit-learn's standard scaler to accomplish this. Note that you fit on the training set and transform on the training and test sets. From here, you could import and use the model. Logistic regression was already imported at top of the notebook, so this step is commented out. from here, you can make an instance of the model. This is normally a place where you can tune the hyperparameters of a model. From here, You can train the model on the data. What this means is the models learn the relationship between your X, in other words, your sepal width, sepal height, et cetera, and your Y, which are the flower species. Finally, you can predict labels of new data. What this code is showing is a prediction for one flower sample. Their prediction was zero. The reason why it was zero and not one is by looking at the line below. The probability of a zero, according to the model was 0.52, the probability of one was 0.47. If it happened to be greater than or equal to 0.5, it would have predicted a one. If this is unclear, let's visualize how logistic regression makes predictions by looking at our test data. If you don't know what this code is doing, don't worry, it's all about the visualization. What you can see in the graph is the probability of virginica given different petal lengths. When the probability is greater than or equal to 0.5, everything in this area is classified as virginica even when it may not be. If you look at these two blue points here, even though they're classified as virginica, they're misclassified, they're actually versicolor flowers. Everything below this threshold is classified as versicolor. An important part of a machine learning model is measuring its model performance. The code here utilizes accuracy as a metric, which is simply the fraction of correct predictions. Accuracy is one metric, but it doesn't give much say into what went wrong. Let's take a look at a confusion matrix. There are a couple of things to notice in this confusion matrix, the first is that it correctly predicted versicolor 10 times when it was actually versicolor, additionally, it correctly predicted virginica when was actually virginica, however, it incorrectly predicted virginica, when it was actually versicolor, these points were misclassified. So that's it. I encourage you to create a logistic regression model.

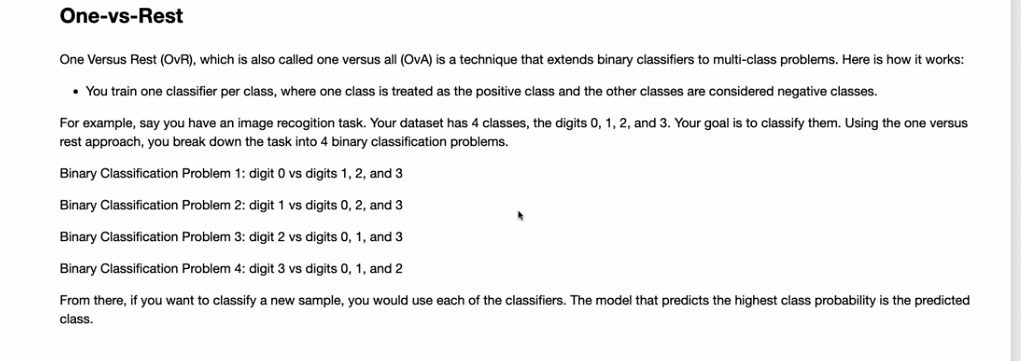
### **Logistic regression for multiclass classification**

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

- [Instructor] A lot of classification models like logistic regression were originally designed for binary classification. That's predicting whether something's one thing or another. For datasets with more than two classes, what do you do? For multi-class classification problems, one approach is to split the task into multiple binary classification datasets, and fit a binary classification model on each. In this video, we'll explore the **one-vs-rest strategy and how you can apply it to logistic regression using scikit-learn. One-vs-rest, which is also sometimes called one-vs-all is a technique that extends binary classifiers to multi-class problems.** Here's how it works. You train one classifier per class where one class is treated as the positive class. And the other classes are considered negative classes. For example, say you have an image recognition task. Your dataset has four classes the digits zero, one, two, and three. Your goal is to classify them. Using the one-vs-rest approach, you break down the task into four binary classification problems. Classification problem one is digit zero versus digits one, two, and three. Prom two is digit one versus digits, zero, two and three. And so on. From there, if you want to classify a new sample you'd use each of the classifiers. The model that predicts the highest class probability is the predicted class. Let's now see an example of this. The first thing you have to do is import the libraries that you want to use. In this case this notebook uses Matplotlib, pandas train-test-split, StandardScaler and of course, logistic regression. The dataset used in this notebook is a modified version of the digits dataset which is arranged into a CSV file for your convenience. The data consists of pixel intensity values for 720 images that are Eight by Eight pixels. Each image is labeled the number from zero to four. This code here loads into a pandas DataFrame. Before you create a machine learning model it's often a good idea to try to understand your data better. A great way to do this is to visualize your data. This code shows a sample of each individual digit as you can see, the images are rather low resolution. They're Eight by Eight pixels. From here, you can split your data into training and test sets. Logistic regressions are affected by scale. So you need to scale your features before using it. You can transform a data onto unit scale by utilizing scikit-learn's StandardScaler. This code creates an instance of multi-class logistic regression. Once you try to model, I encourage you to look at the attributes of a model. In particular for this model, I encourage you to look at the intercept and the shape. One thing to notice here is that we have four intercepts here. This is because we had a multi-class problem. We had binary classifier one, two, three, and four. The last thing I want you to look at is how the predictions are made. If you look at the predictive probabilities for this one simple digit, you notice that we have four different probabilities. We have the first one, the second, the third and the fourth. This is for class zero, class one, two and three. Notice that the predicted probability for class one is .98. What's going to happen here is it's going to predict class one because it has the highest probability. So that's it. I encourage you to use a logistic regression for multi-class classification.



<https://towardsdatascience.com/multi-class-classification-one-vs-all-one-vs-one-94daed32a87b>



### **Decision trees using scikit-learn**

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

- [Instructor ] One of the most important considerations when choosing a machine learning algorithm is how interpretable it is. The ability to explain how an algorithm makes predictions is useful not only to you, but also to potential stakeholders. A very interpretable machine learning algorithm is a decision tree. You can think of it as a series of questions, designed to assign a class or predict a continuous value depending on the task. Example image is a decision tree, designed for classification. So you have a flower with the following feature, petal length of 4.5 centimeters. The way decision trees work is you start at the top of the tree and ask questions until you reach these leafy green nodes. You first has the question is 4.5 centimeters less than or equal to 2.45? This is false, so going this other node. Is 4.5 centimeters less than or equal to 4.95? This is true, so you'll end up with a leaf node. Leaf nodes our predictions are assigned. In this leaf node, there were 38 versicolor and virginica. Their prediction for this leaf node is versicolor, as is the majority class. In this video, I'll share with you how you can create intuitive decision tree using Scikit-learn. The first thing you have to do is import the libraries that you're going to use. In this case, you'll import Matplotlib, pandas, the Iris data set, as well as train-test split, and decision tree classifier. This next piece of code loads the Iris data set. From here, you can split your data into training and test sets. What this image shows, is which variable the data from the day from df one, two for particular train test split. This is a really important step as oftentimes this decision trees overfit the training set. Train test split will help you avoid that. It's also important to note that another benefit of decision trees is that you don't have to standardize your features. This is different from other algorithms like logistic regression, and K nearest neighbors. From here, you can create a decision tree model. This model is already imported earlier in notebook so it's commented out. The next step is to make an instance of your model. This is normally a place where you can tune the hyper parameters of the model. The code below constraints the model to have at most a depth of two. Tree depth is a measure of how many splits it makes before coming to a prediction. It's important to note that max\_depth is not always equal to depth. Max\_depth is simply something that pre-prunes a tree to only grow at most discerned depth. From here, you can train your model. I can also make predictions. You can also measure your model performance. This notebook uses accuracy as the metric, which is a fraction of correct predictions. This section shows how to tune max\_depth. If you look at the graph, you'll see a couple of things. The first is that accuracy increases up to a certain max depth. There could be a couple reasons for this. One potential reason is that max\_depth is not necessarily equal to depth. It's possible that trees with max\_depth four and five have the same depth. It could also be that after a certain point, the models not getting any more useful information after a certain depth. So that's it, I encourage you to create a decision tree of your own.

### **How to visualize decision trees using Matplotlib**

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

- [Instructor] How do you understand how decision tree makes predictions? One of the strengths of decision trees, are they're relatively easy to interpret, as you can make a visualization based on your model. This is not only a powerful way to understand your model, but also to communicate how a model works as stakeholders. In this video, I'll show you how decision trees, can be plotted with Matplotlid . The first thing you have to do, is import libraries. Take note that you're also importing tree. This is what actually plots to the decision tree. The next step is a loaded dataset, in this case is the Iris dataset. From there, you can split your data into training and test sets. This is really important for decision trees, as they tend to be a high variance algorithm. What this means, is they tend to overfit on the training set. The next step is to create a decision tree model. Before you can make a visualization based on a decision tree, you need to make a decision tree first. So you make an instance of your model, you train the model on the data, you can also make predictions and measure model performance. After you fit a decision tree, you can make a visualization, based on the model, to do this, you can utilize tree.plot\_tree, with the instance of a fit model. This is not a perfect visualization, there're a couple of reasons why. The first, it seems a bit small, so let's try to fix that. This code makes the figure size a bit bigger, as well as the DPI. The problem with this image, is even though the visualization is bigger, it's still hard to understand what's going on. The next step is to make the tree more interpretable. There're a couple of ways to do this, what you can do, is utilize the feature name parameter, as well as the class name parameter. This tree is more interpretable, there're a couple of reasons why. First, it's easier to understand the leaf node predictions, this leaf node predicts the setosa, this one versicolor, and this one virginica, additionally, this visualization is more visually appealing, so that's it, you can visualize decision trees using Matplotlid.

### **Bagged trees using scikit-learn**

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

- [Instructor] Each machine learning algorithm has strengths and weaknesses. A weakness of decision trees is that they're prone over fitting on the training set. A way to mitigate this problem, is to constraint how large a tree can grow. Bagged trees try to overcome this weakness by using bootstrapped data, to grow multiple deep decision trees. The idea is that matrix protect each other from individual weaknesses. What this image shows is that multiple decision trees come together to make a combined prediction. In this video, I'll share with you how you can build a Bagged Tree Model. The first step is to Import Libraries. The Dataset used in this notebook is a housing prices for King County. The code below loads the dataset. The goal of this dataset is to predict house prices based on features like number of bedrooms and bathrooms. This notebook only selects a small subset of the features for simplicity. However, if you have time I encourage you to play with this notebook, and add and subtract features. This code is just arranging the dataset into a features matrix, and target vector. From here you can Split the Data into Training and Test Sets. The next step is to build a Bagged Trees model. You import the model that you want to use. This is actually commented out because it was used earlier in the notebook. From here you can make an instance of the Model. Note that n\_estimators is how many decision trees are coming together to make a prediction. Next you can train the model on the data, as well as make predictions. You can also measure model performance. For a Bagged Trees Regressor. You can use a metric R squared. The code below tunes and estimators, which in this case is the number of decision trees. This code can take some time to run. As you have multiple estimators coming together to make a prediction. In the graph notice that their score starts improving after a certain number of estimators. There could be a couple reasons for this. One potential way to get a better score, would be to include more features in the features matrix. So that's it. I encourage you to try building up Bagged Trees model.

### **Random forests using scikit-learn**

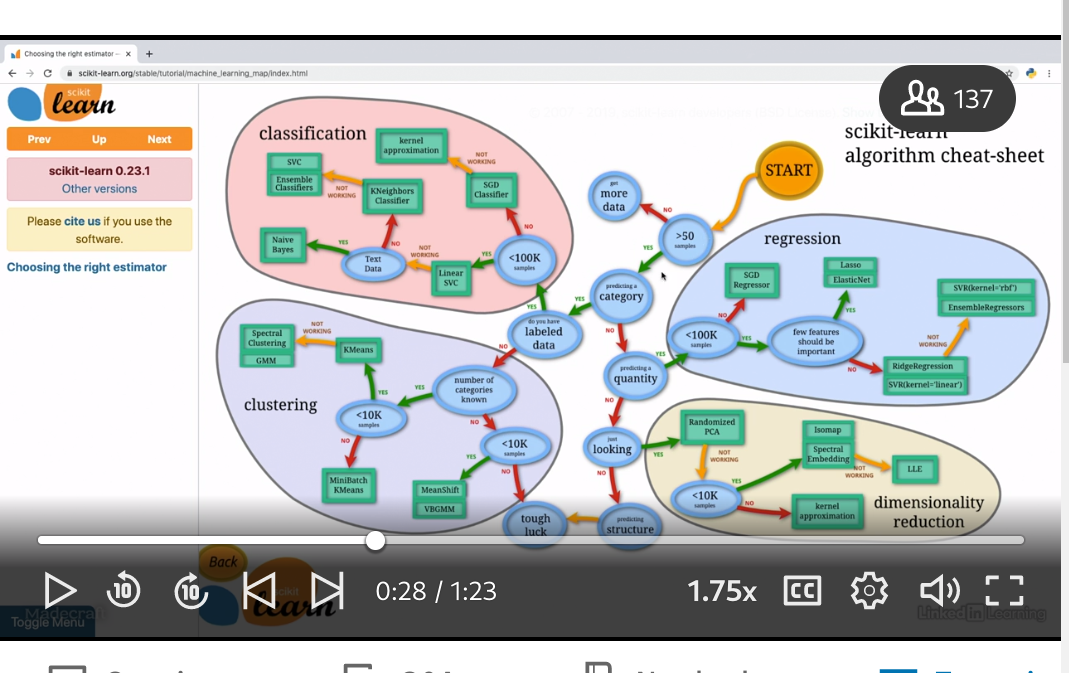
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Each machine learning algorithm has strengths and weaknesses. **Bagged tree models use many trees to protect individual decision trees from overfitting. However, bagged tree models are not without weaknesses. Suppose you have one very strong feature in a data set, most of the trees will use that feature as the top split. This will result in many similar trees. You can think of random forest as a variant of a bagged tree model.** The difference is that each time a split's considered, only a portion of the total number of features are split candidates. In short, random forests make the individual decision trees less correlated In this video, I'll share with you how you can build a random forest model using Scikit-Learn. The first step is to import libraries. The next step is to load a dataset. This dataset contains house sale prices for King County. The code below loads the dataset. The goal of this dataset is to predict house prices based on features like number of bedrooms and bathrooms. The code in this notebook only selects five features to make a prediction. This notebook only selects a couple of features for simplicity. However, I encourage you to play with adding and subtracting features. You can now create a features matrix and a target factor. The next step is to split your data into training and test sets. From here, you can build a random forest model. After importing the model, you can make an instance of the model. This is a place where you can tune hyperparameters. In the case of random forest, you have N\_estimators. What N\_estimators is, is how many decision trees you have coming together to make a prediction. From here, you can train your model and make predictions. It's important to note, that the training of a random forest model can take some time. In this case you have 100 estimators that need to be trained and come together to make a prediction. The next step is to measure your model's performance. In this case, the score method uses R squared as a metric. Since bagged tree and random forest models are ensemble models, you can visualize individual decision trees comprising those models. Here's the first decision tree of 100-estimator bagged tree model. You can also visualize individual decision trees for a random forest model. One of the benefits of using a random forest model is that they can give you a feature importance metric. Feature importance metrics can give you an idea of what features were important in making predictions for your model. In this example, the random forest is suggesting that pedal length was the most important feature. So that's it. I encourage you to build a random forest model.

### **Which machine learning model should you use?**

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

- [Instructor] With so many machine learning algorithms available from scikit-learn, which algorithm should you choose? Selecting a good enough model from among a large number of possible machine learning models is one of the hardest parts of machine learning. Some algorithms are better suited to different types of data and problems. Luckily, a quick answer to model selection with scikit-learn is, use the algorithm cheat sheet. It's meant to give you a rough guide in how to choose an algorithm. From the start point, you first ask, do you have more than 50 samples? From there, you keep on answering questions until you get an idea of what you should try. If you don't use the cheat sheet, here are a few things to consider when choosing a model. The first thing is a problem you're trying to solve. For example, if you have a supervised learning problem, figuring out if you're trying to predict a continuous or categorical value can be an important first step. Next, always consider the size, quality, and structure of your data. There's no machine learning without data. You should also consider the strengths and weaknesses of each algorithm you're considering. It's especially important, as some algorithms take longer to make predictions. Also, more complex models are often more difficult to maintain. Finally, consider the urgency of a task. Some models take longer to train and tune. Now, if you're feeling a little shaky, you're not quite sure which model to choose, don't worry, you've got this. I now encourage you to try a couple models and learn from the process.



### **What is unsupervised learning?**

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

- [Instructor] In machine learning, you aren't always tryna predict the value. Sometimes your goal is to find some structure in your dataset. Unsupervised learning is when you train an algorithm without giving it the answers for examples in your dataset. In the context of psychic learn, this means that you only provide a features matrix when you fit your algorithm. A features matrix is a two-dimensional grid of data where rows represent samples and columns represent features. Unlike supervised learning, there's no target factor. It's important to emphasize that unsupervised algorithms don't make predictions from the data. There are two common types of unsupervised learning algorithms. The first is clustering. Clustering is often used to discover natural groupings in a dataset, when common use is for market segmentation. Companies often have large amounts of customer information. By clustering customers into different segments, they can more efficiently sell or market to their customers. Another common type of unsupervised learning is dimensionality reduction. You can think of dimensionality reduction techniques as data compression algorithms. They can make your data take up less space on your computer. Having less features in your data can make visualizing your data easier as well speed up the fitting of your machine learning algorithms. So that's it. Unsupervised learning helps you discover structure in your data.

Give feedba

### **K-means clustering**

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

- [Instructor] Clustering algorithms have identified this thing groups of data. One example is who's clustering the group customers based on their behavior. There are so many clustering algorithms. But the most commonly used algorithm is K-Means. In this video, I'll show you how to use K-Means Clustering to find some underlying structure in your data. The first step is to import libraries. The next step is to load a dataset. This notebook uses the Iris data set. From there, you can arrange your data into a features matrix. It's important to note that K-Means is considered unsupervised learning algorithm. This means that you only need a features matrix. In the Iris data set, there are four features. In this notebook, the features matrix will only be two features, as it's easier to visualize clusters in two dimensions. It's important to mention that you do not eat a target factor, as this is an unsupervised learning algorithm. Like a lot of different algorithms, K-Means is sensitive to the scale of your data. To standardize your data, you can use Scikit-Learns StandardScaler to help standardize your features. Before you cluster your data, it's often a good idea to try to understand your data better. If your data is two or three dimensional, it's a good idea to at least try to visualize your data. Hopefully you can see if there's a natural looking clusters. From this graph, it looks like there's at least two clusters. You can now apply K-Means Clustering. In K-Means clustering, you can specify the number of clusters you want. In Scikit-Learn, this parameter is called N-Clusters. In the case of the code below, the number of clusters is at three, because most people who use this data set happen to know that there are three species. After you fit your model on the features matrix, you can get the cluster labels as well as the cluster centroids. Let's now visually evaluate the clusters. Here are your three clusters. You have cluster one, cluster two and cluster three. Let's now see how well instead, compared to the Iris data sets labels. There are a couple things you notice when you look at the two graphs. On the left, you have the K-Means clustering graph, on the right you have the flower species. They actually look pretty similar. It looks at K-Means picked up some flower differences with only two features and not the labels. The colors are different in the two graphs, simply because K-Means is on arbitrary cluster number, and the Iris data set said has an arbitrary number in the target column. So that's it. K-Means is a clustering algorithm that you can use to find some structure in your data.

### **Principal component analysis (PCA) for data visualization**

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

- [Narrator] Are all the features in our dataset needed? Say you have some flowers and you measure their petal length. If you have a column of that measurement in centimeters, and another column with measurement in inches do you need both columns? In that circumstance, you could probably drop either column without losing information. In other cases dropping a column could lead to issues. Principal Component Analysis better known as PCA. Is a technique that you can use to smartly reduce the dimensionality of your dataset while losing the least amount of information possible. One use of PCA, is for data visualization. In this video, I'll share with you how you can use PCA to help visualize your data. The first step is to import libraries. From there, you can load your dataset. The dataset used in this notebook is the hours dataset. The next step is to standardize your data PCA like a lot of different algorithms is affected by scale. You can transform your data onto unit scale by using scikit-learn standard scaler. Note, that PCA is an unsupervised learning algorithm. What this means is that when you use the fit step, you only fit it on your features matrix. From here, you can apply PCA the code below projects the original data which is four dimensional into two dimensions. Note that after you applied dimensionality reduction, there usually isn't a particular meaning assigned to each principal component. The new components are just the two main dimensions of variation. The next step creates a visualization of the first two principle components. One thing to note from the graph is that the Setosa class is well separated out from the other classes. One thing to look after PCA is the explained variance. What this shows is how much variance can be attributed to each of the principal components. This is important. As well, you can convert four dimensional space to two dimensional space. You lose some of the information when you do this. You can look at how much information is attributed to each of the principle components by using the explained variance ratio. These two principle components contain roughly 96% of the information in the dataset. The first component contains roughly 73% and the second one contains roughly 23%. So that's it.PCA can be used to help visualize your data.

### **PCA to speed up machine learning algorithms**

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

- Do you want to speed up the fitting of your machine learning algorithm? Second learn offers quite a few ways to do this. One way is to train your model in parallel using n\_jobs parameter, which exists for many psychic learn models. A really simple way is to reduce the number of columns or rows in your data. The problem with this approach is it's hard to know which rows and especially which columns to remove. Principle Component Analysis, commonly known as PCA is a technique that you can use to smartly reduce the dimensionality in your data while losing the least amount of information possible. In this video, I'll share with you the process of how you can use PCA to split the fitting of a logistic regression model. The first step is to import libraries. The next step is loaded dataset. The dataset is a modified version of the MNIST dataset that contains 2000 labeled images of each digit zero and one, the images are 28 pixels by 28 pixels. For your convenience, it's been arranged into a CSV file before you apply PCA or a machine learning model. It's often a good idea to try to visualize your data. The Coppola shows a sample image of each digit zero and one. The next step, it says, put your data into training and test sets before continuing it's important to standardize your data. PCA and logistic regression are sensitive to the scale of your features. You can standardize your data onto unit scale by using a psychic learns standard scaler. This next piece of code applies PCA and logistic regression. For the code here, note that n components equals 0.9. It means it's psychic learn we'll choose a minimum number of principal components such that 90% of the variance is retained From here, you fit PCA in the training set After that, you apply the mapping to both the train set and the test set. The final step is to create a logistic regression model. When you run this code, there's a couple of things to notice. First, the number of dimensions before PCA is 784. This is because the original images were 28 pixels by 28 pixels and 28 times 28 is 784. The number of dimensions after PCA was 104. Additionally, the classification accuracy with this model was very high. So even after removing a lot dimensions going from 784 dimensions to 104, the model still worked pretty well for this particular dataset. You can look at the relationship between cumulative explained variance and number of principal components. One thing to notice in this graph is that after roughly 150 to 200 dimensions, there wasn't a lot of explained variance in the remaining principal components. So that's it. PCA can be used to speed up the fitting of your algorithm.

### **scikit-learn pipelines**

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

- [Instructor] Machine learning is not always about applying a single machine learning algorithm. For a lot of machine learning applications, you'll need to apply various data processing steps, data transformations, and potentially multiple machine learning algorithms. This can lead to a lot of code. The question becomes, how do you keep your code organized and as bug free as possible? In this video, I'll share with you how you can use Pipelines in Scikit Learn to make your code cleaner and more resilient to bugs. To demonstrate the utility of Pipelines, this notebook shows how much less code you need to chain together PCA and logistic regression for image classification. Before getting to that though, you need to import the libraries that you're going to use. The dataset using this notebook is a modified version of the MNIST dataset that contains 2000 labeled images of each digit, zero and one. The images are 28 pixels by 28 pixels. For convenience, I arranged the data into a CSV file. This code loads the data into a panda's DataFrame. This code is a chaining together of PCA and logistic regression. It's quite a bit of code. The first step is a train\_test\_split. From there, there's a standardization step. Notice that there's a fit step here, as well as when you apply PCA and logistic regression. There's quite a few places where an error can occur as there's quite a bit of code. Let's now try to do this with Pipelines. The first step as before is a train\_test\_split. The next step is creating a Pipeline. You still data scalar, PCA and logistic regression. You can also name the steps, you have scalar, PCA and logistic. Note, you still have fit step, but unlike before you don't have three of them. Another advantage of using Pipelines is you can visualize your Pipeline. You can see all the steps your machine learning model took to get to the end. So you have a Standard Scalar, you have PCA and logistic regression. So that's it. Pipelines can make our code more organized and easier to understand.