**Section1: Introduction and Housekeeping**

Compared

**Section2: SageMaker Housekeeping**

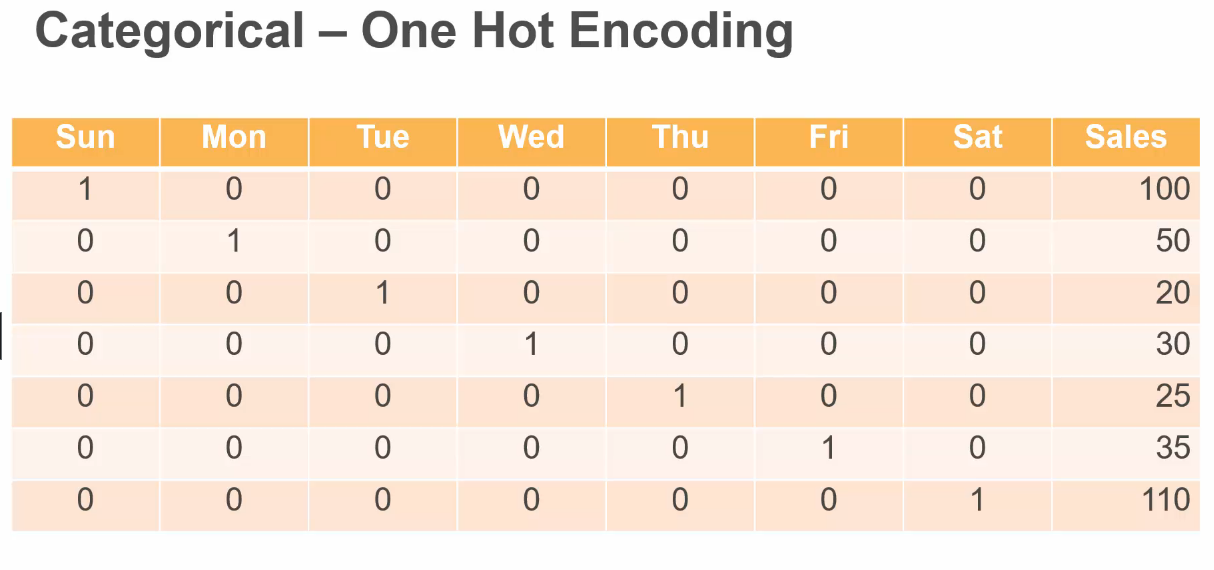
* For the notebook instances, if you want to open a terminal in the instance that is running go to:
  + new-- and “terminal” is down near the bottom
* You may need to go back and create a Kaggle account and do some manual data pulls from that site.

**Section3: Machine Learning Concepts**

* Unsupervised Learning:
  + only data with no defined target
  + group similar observations
  + detecting anomalies
* Unsupervised Algorithms:
  + clustering
  + dimensionality reduction – like PCA or principal-component-analysis
  + group words that are used in similar context or have similar meaning
* Reinforcement Learning
  + Decisions need to be made with uncertainty
    - e.g. autonomous driving, or games like Chess
  + this type of learning uses reward functions to reward correct decisions and punish incorrect decisions

Data Manipulation Examples:





There is also the possibility of combining two or more categorical features to form new features:

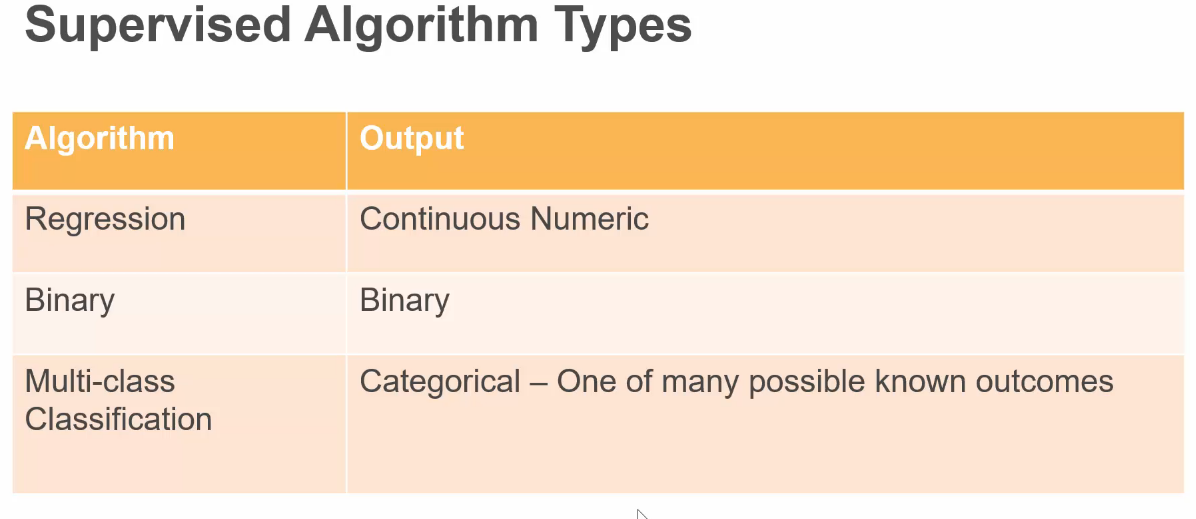


Underfitting: poor performance on both the training and the test set

* Correct by:
  + adding more complex features
  + adding more relevant features
  + expanding the size of the training set with more examples
  + optimize the hyper-parameters

Overfitting: Good performance on the training set, but poor performance on the test set. This can occur when the model memorizes the training data and cannot generalize for new/unseen data.

* Correct by:
  + remove more complex data
  + optimize the hyper-parameters
    - e.g. by reducing the number of “passes” or “epochs” the training algorithm goes over the data



When working with Binary Classifiers:

* We need to clarify what outcomes are “positive”
* We may need to convert outcomes into binary class outcomes

**Recall =** TP / (TP +FN)

The True-Negative-Rate of a model:

The False-Positive-Rate is also called “**probability of false-alarm”**

The False-Negative Rate is also called “**misses**”

**Precision =** (True Positive) / (True positive + False positive)

**Accuracy** = (True Positive+ True Negative) / (All Outcomes)

**F1\_score =** 2\*Precision\*Recall / (Precision + Recall) --- harmonic mean between precision and recall

**Area under the Curve** (AUC) metric is the area under the curve defined by:

True Positive Rate vs. False Positive Rate (on opposing 2D axes)

Good Models have AUC near to 1

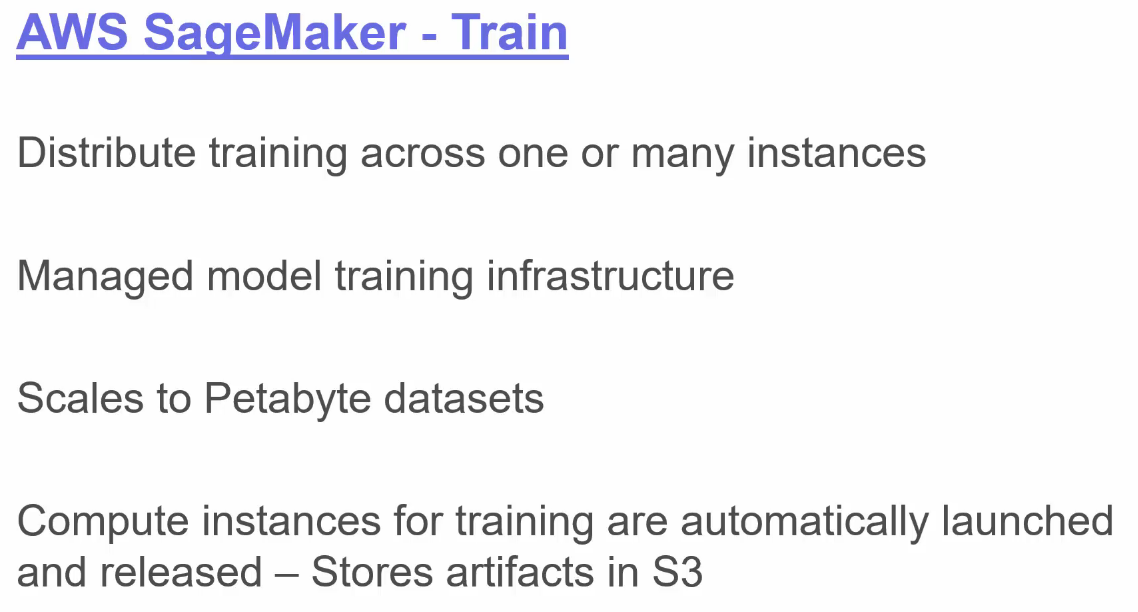
A value of AUC = 0.5 is considered a random guess

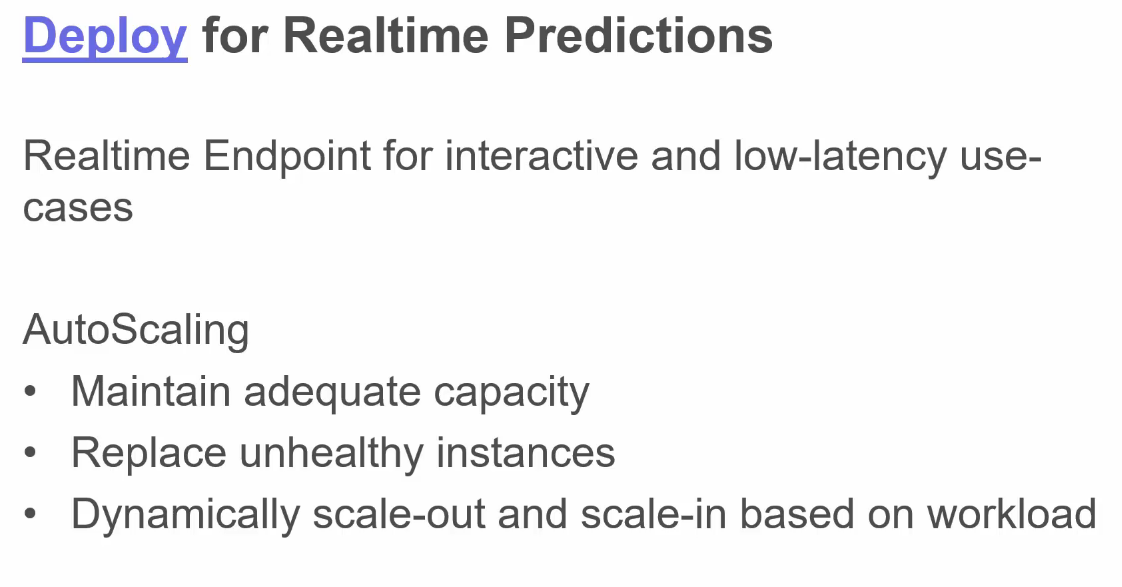
And a value of AUC near to 0 means the model is inverting true predictions

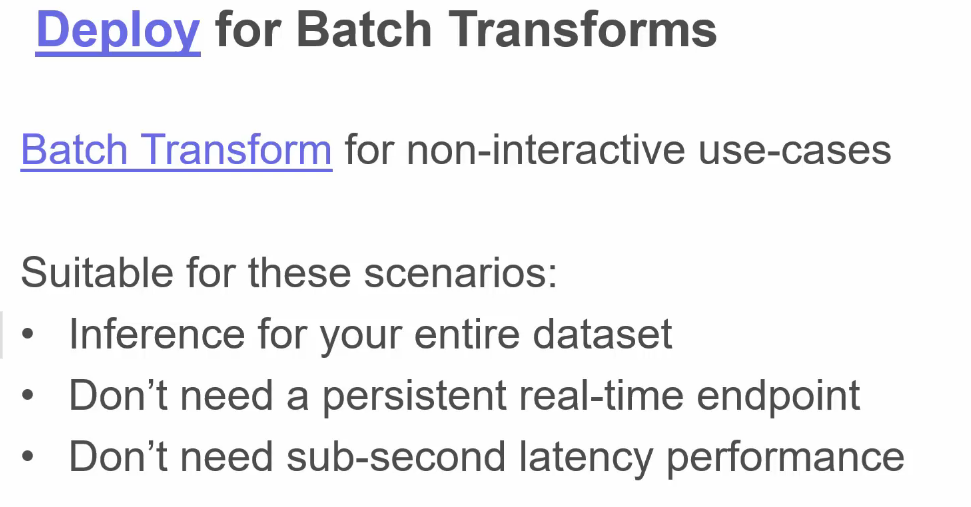
Key! is that this metric is for evaluating how well a model identifies positives and DOES NOT evaluate how well it does at evaluating negatives.

**Section5: SageMaker Service Overview AND**

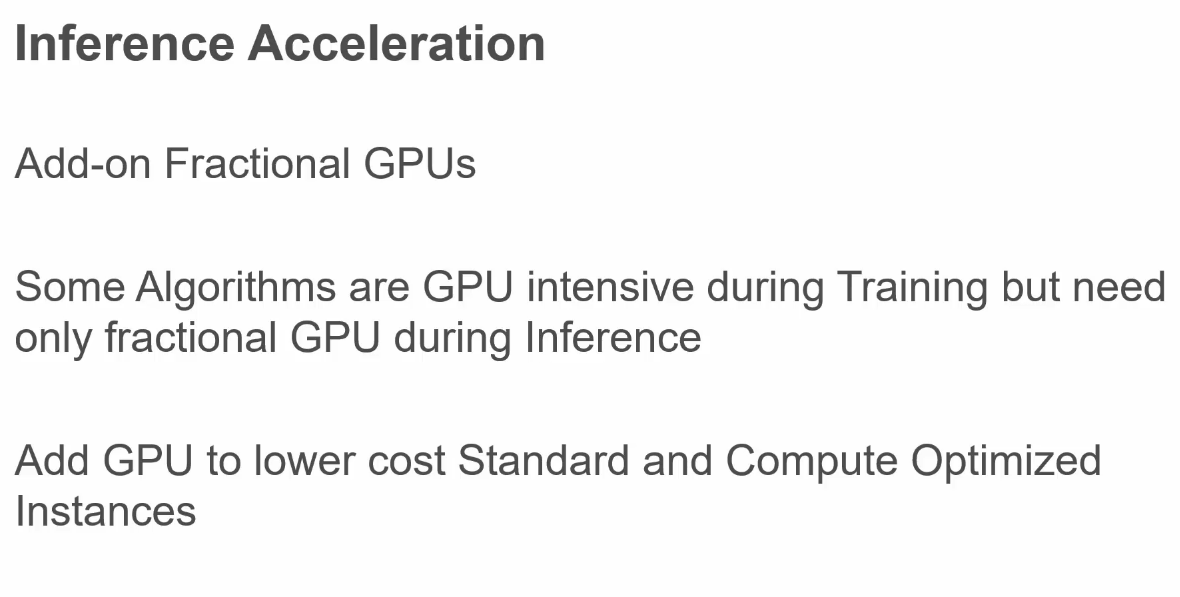
**Section6: SageMaker SDK**

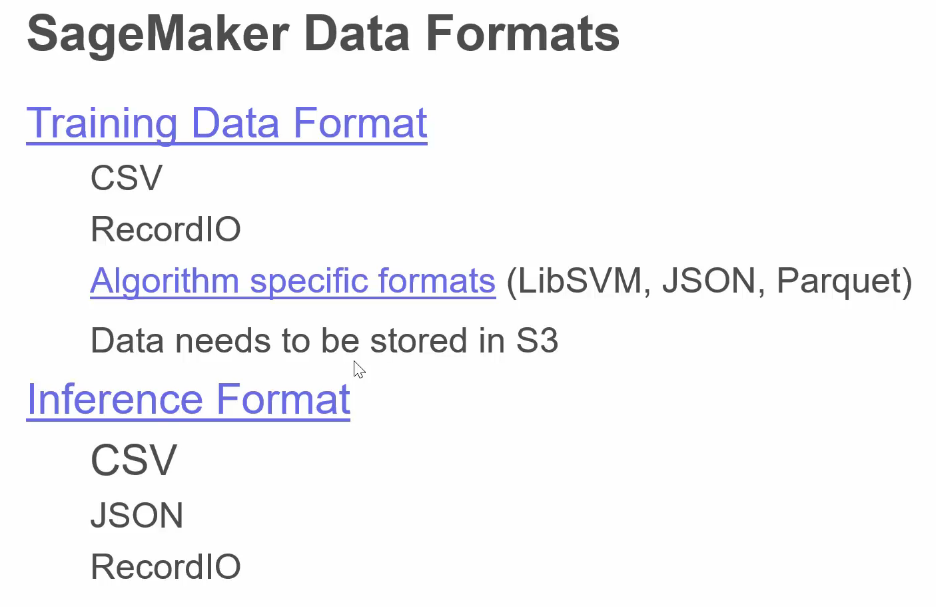


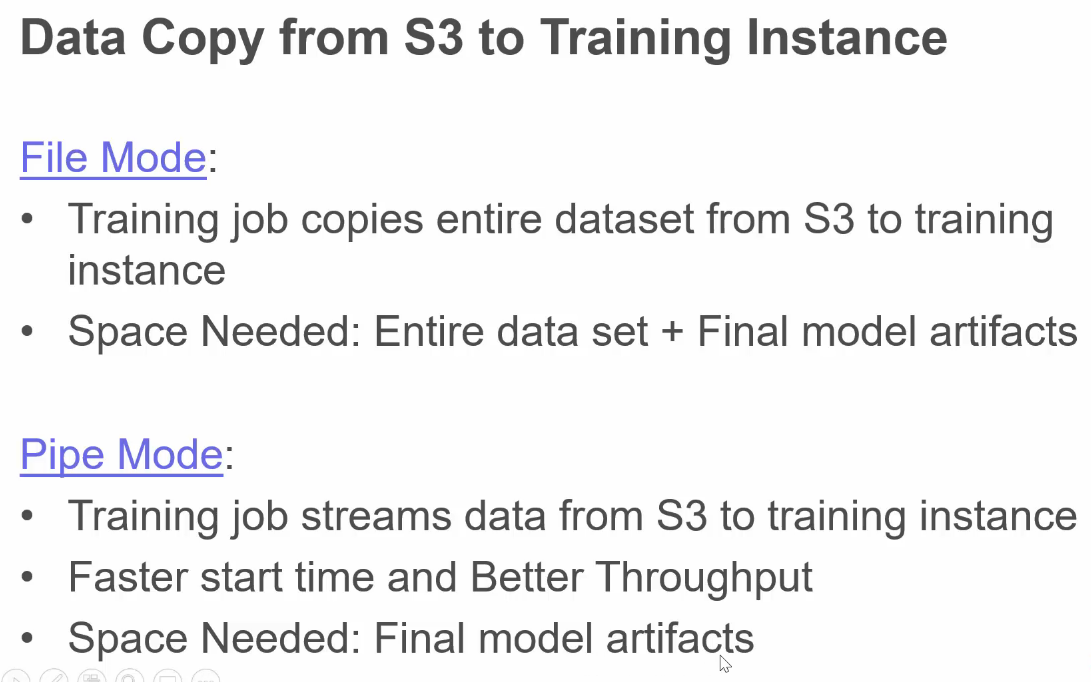


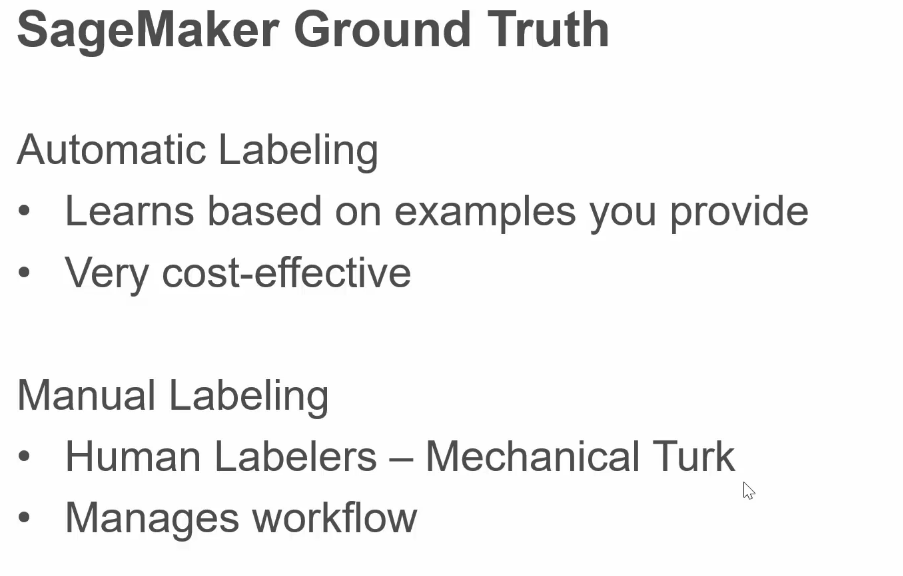
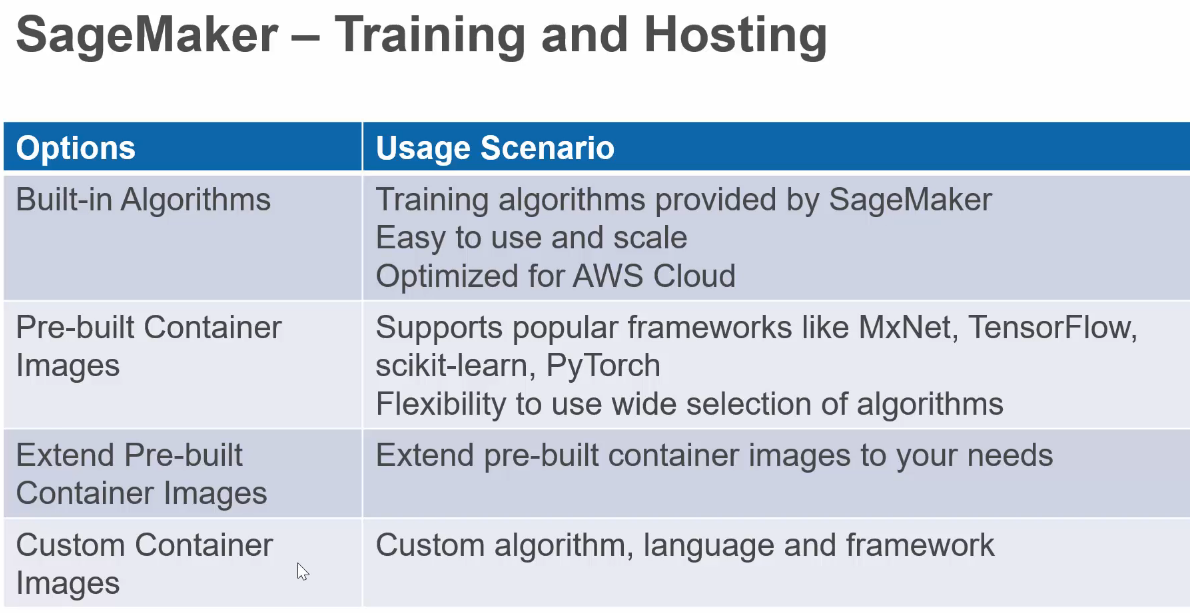


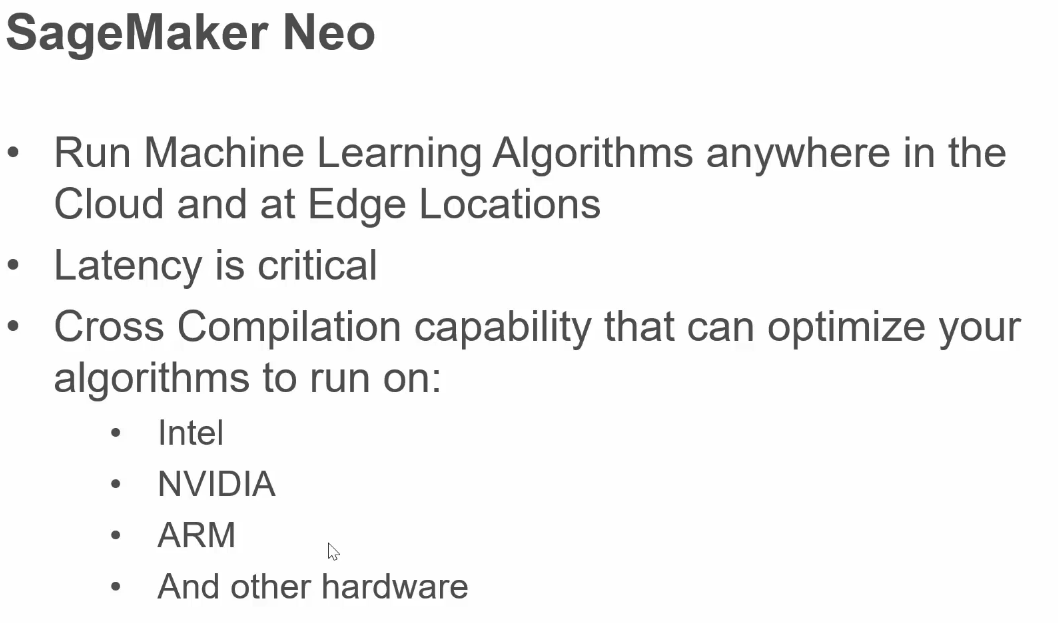
The kind of GPU during training only by using instances with GPUs added on sounds like it might be best for our applications.

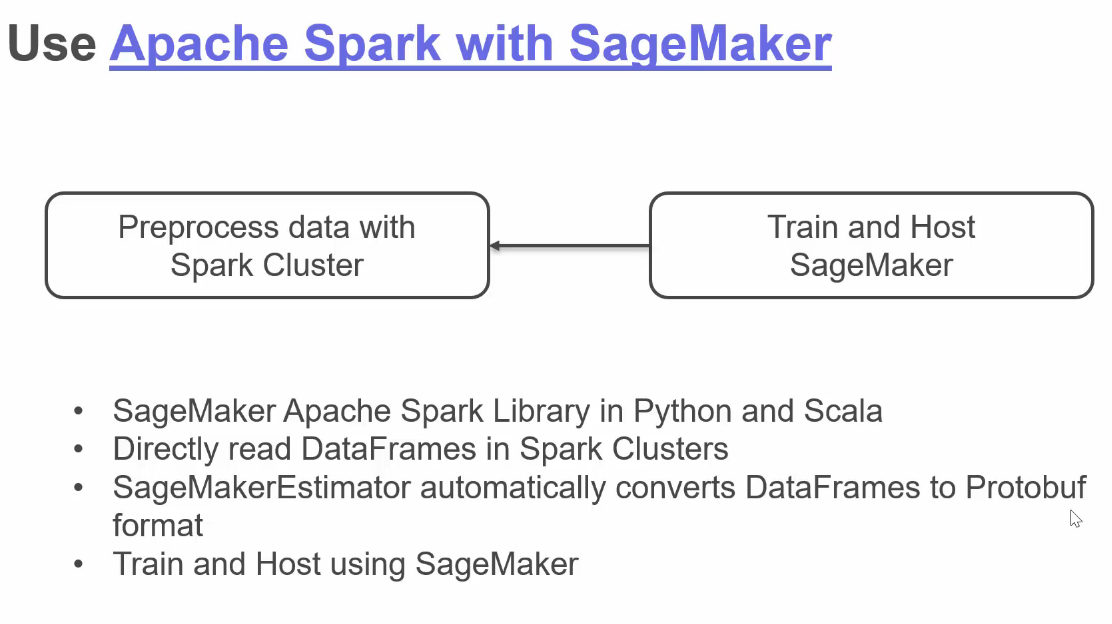


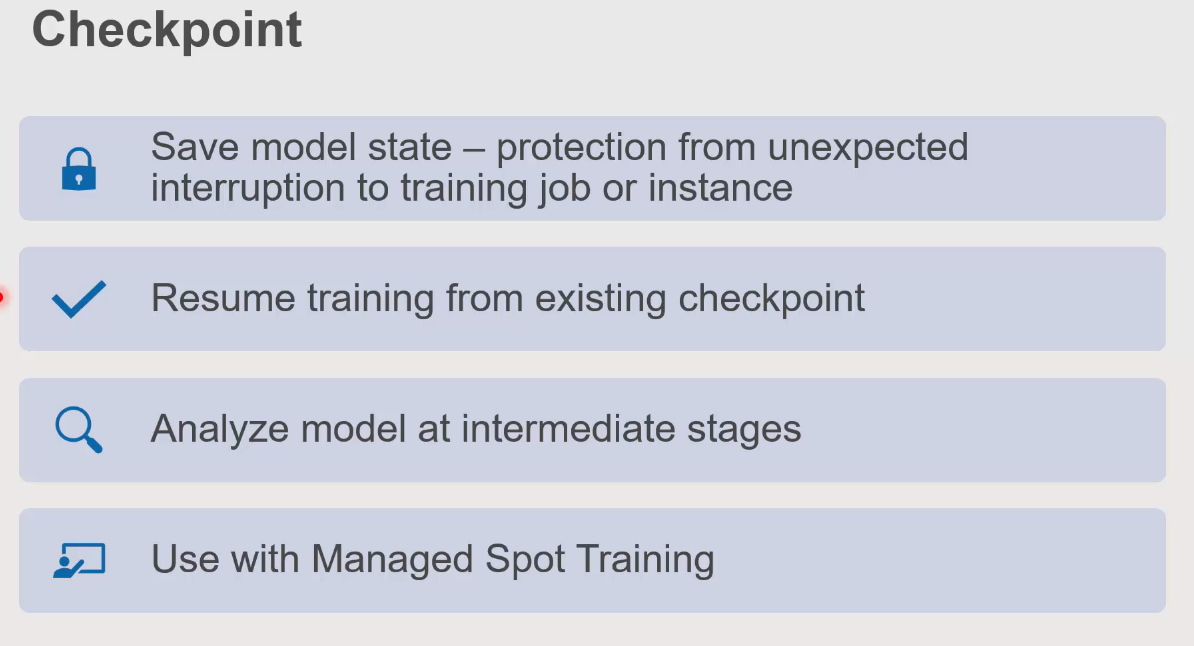






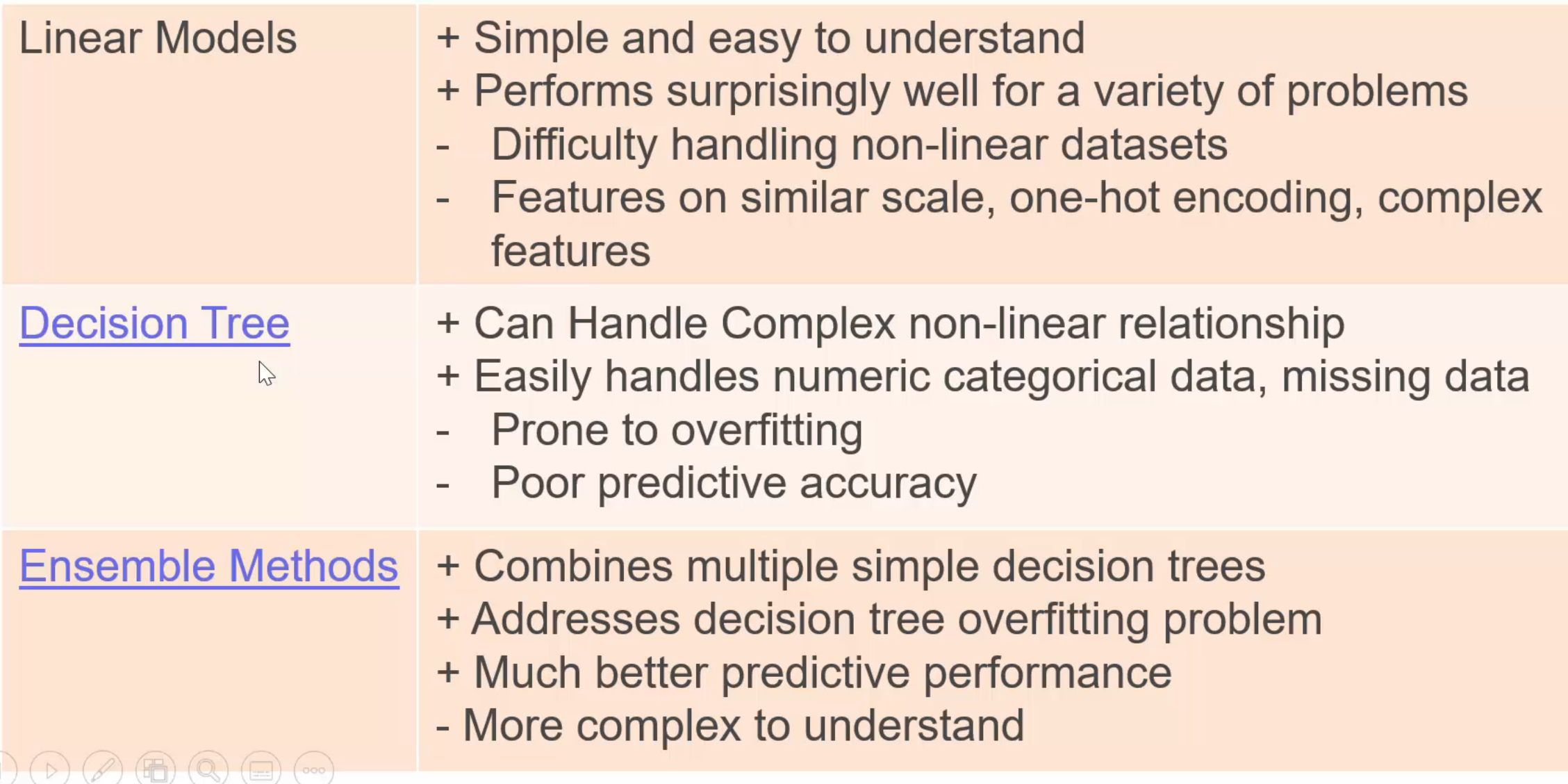






If you plan to be using the cheaper “spot” training instances to do your training jobs, then you MUST make sure you configure CHECKPOINT(S) appropriately for the usual interruptions that these cheaper kinds of jobs routinely endure.

**Section 7: Gradient Boosted Trees using Built-In XGBoost Algorithm:**

Decision trees easily overfit without artificially setting depth to some fixed depth. But then they can be prone to underfitting. Hence in practice a single decision tree is rarely used.

Decision trees are NOT good at extrapolating.

* So in practice they employ “caps” on top and bottom, beyond which the prediction is just whatever the prediction would have been at the cap X-value.

Ensemble methods combining multiple decision trees are more commonly used.

“Bagging” and “Boosting” are two different approaches to generating different decision trees from the same training data.

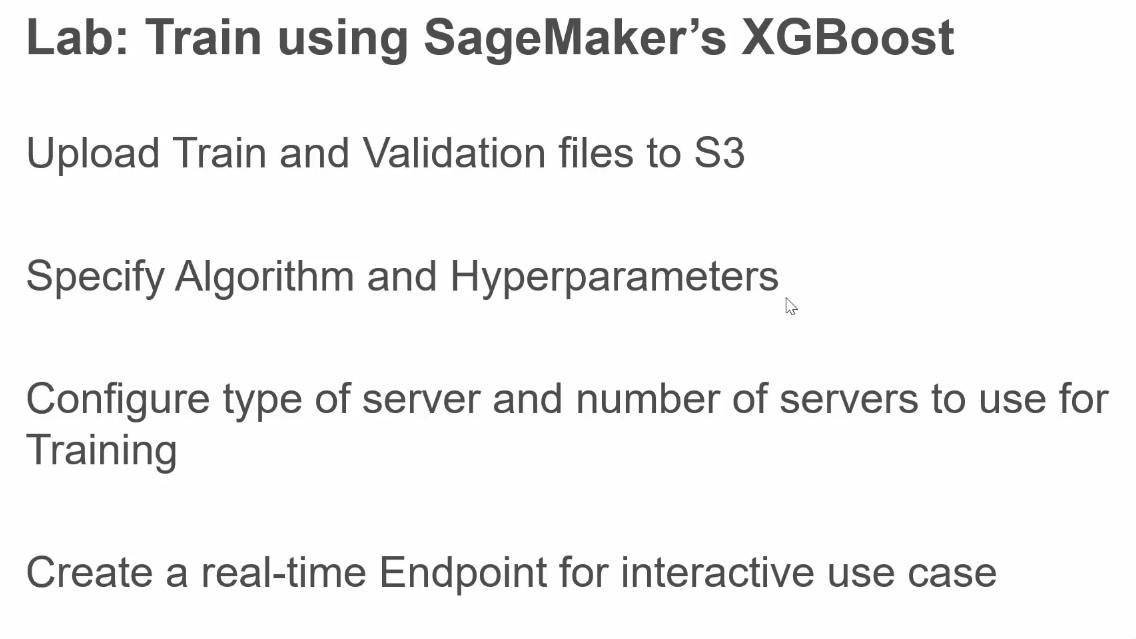
* Bagging: random sample of training data used at every step
* Boosting: start with a simple tree, then weight the incorrect guess of that simple tree more heavily and train a new tree, repeat until ensemble of produced trees generates negligibly better results.

His approach is to test algorithms/implementations in the SageMaker notebook and then proceed to full cloud implementation using SageMaker after a successful test.

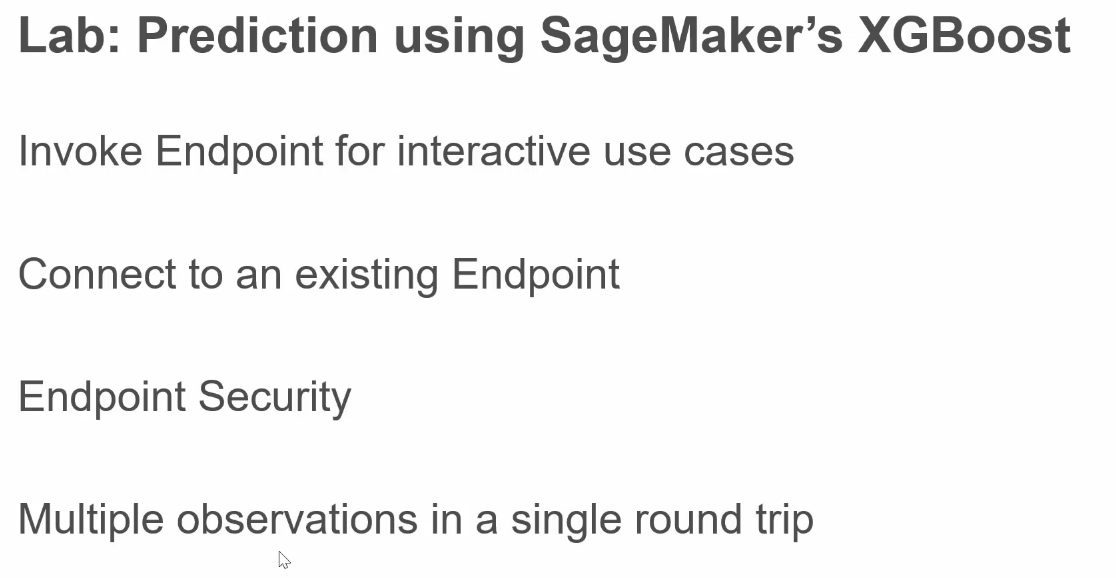
**Important:** For SageMaker built-in algorithms, the data is expected to be in the form:

* 1st column is expected to be the target outcomes
* other columns are expected to be the features

Ultimately, after looking at the labs which compared predictions using Regression and XGBoost, we recognize that XGBoost can equally utilize categorical and quantitative features equally well without extensive feature engineering. This is part of its power.



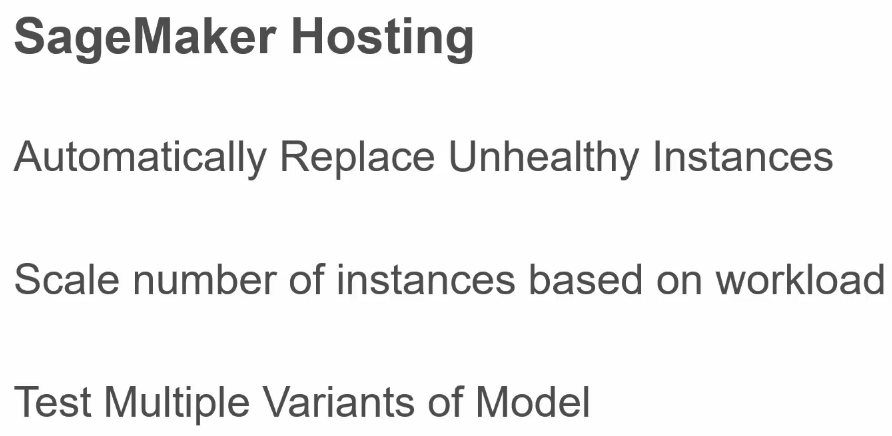
Once you have already trained a model and created an endpoint to host that model and access for predictions:

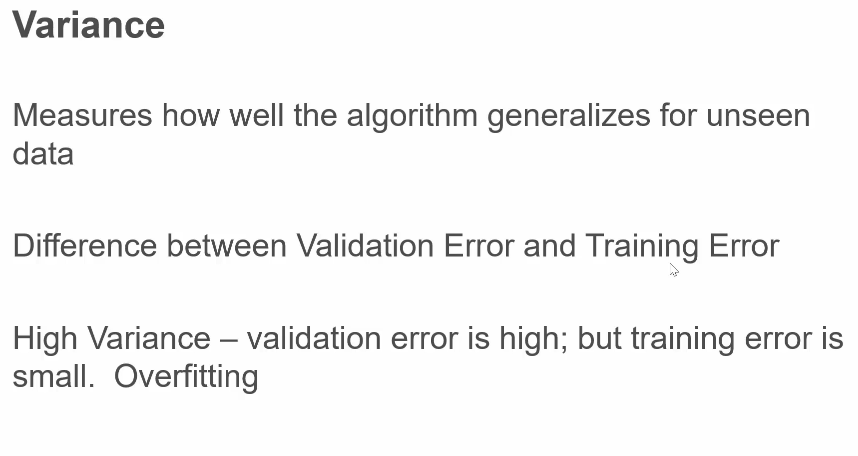


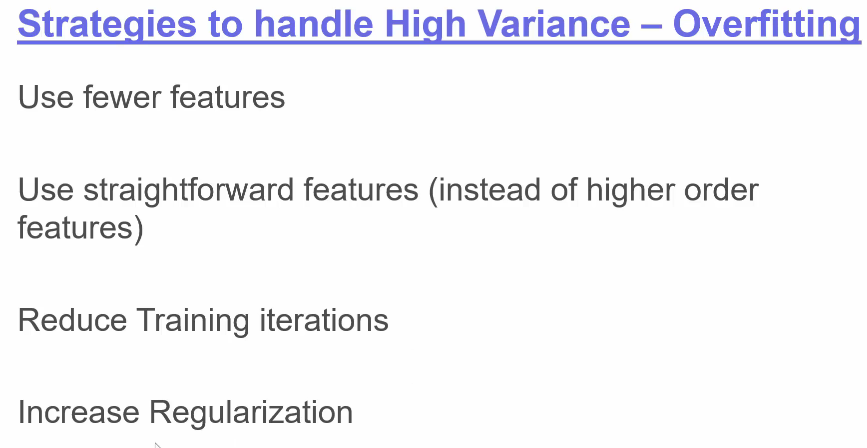
Security was loosely discussed here only in terms of the configured role we had already provided to notebook instances.

It is a good idea to send observations “a few at a time” to the endpoint for prediction. For example he splits the ~ 6000 observations into 10 parts and sends these ten parts one-at-a-time.

Some scaling capabilities discussed were:







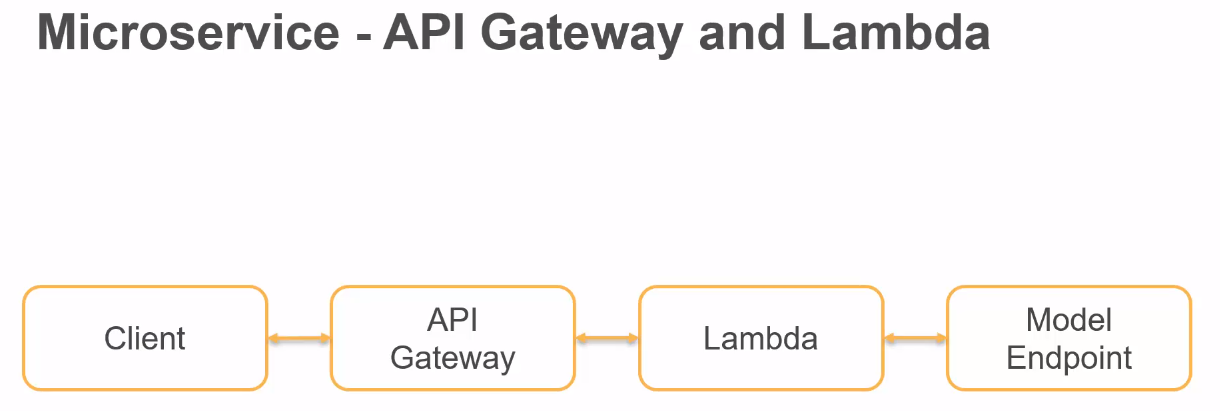
Regularization “tones down” a model’s dependence on specific features and tell it to learn on more features. XGBoost regularization hyperparameters are below:

* **L1 Regularization:** the algorithm is told to aggressively eliminate features that are not important.
  + Useful for a large dimension data-set with thousands of features
  + hyper-parameter controlling L1 is *alpha; default=0*
* **L2 Regularization:** the algorithm reduces the weight of individual features and allows other features to influence the outcome as well besides the main contributing ones the model has learned.
  + Often this is a good starting point
  + hyper-parameter controlling L2 is *lambda; default=1*

SageMaker also includes its own search utility for finding optimal hyperparameters in addition to the traditional grid and random searches. A specific part of the course will later cover this *Bayesian Search* in greater detail.

* Its a *Bayesian Search* which treats hyperparameter tuning as a ML problem in itself. And it often converges faster.

**Section 8: Invoke Model endpoints from external clients**

The link below is to a conversation about trying to revive an endpoint without having to retrain after having deleted it in order to save money

* <https://www.udemy.com/course/aws-machine-learning-a-complete-guide-with-python/>[learn/lecture/16506846#questions/14643154](https://www.udemy.com/course/aws-machine-learning-a-complete-guide-with-python/learn/lecture/16506846" \l "questions/14643154)
* Basically see the *Deploy\_Model* notebook he included in the outer directory

On your local machine you launched a jupyter notebook from the directory:

* /home/cole\_mcgee/Documents/Training\_Materials/Udemy/Sagemaker/AmazonSageMakerCourse\_Original
* Then you opened the notebook Integration\_Example/invoke\_using\_sagemaker\_sdk
* Remember to use aws cli command on your local machine you must run them with –profile like:
  + *aws s3 ls --profile ml\_user\_predict*
  + *aws --profile ml\_user\_predict sagemaker list-endpoints* 
    - **Important note:** I had to add the IAM role [AmazonSageMakerFullAccess](https://console.aws.amazon.com/iam/home" \l "/policies/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonSageMakerFullAccess) to ml\_user in order to allow the list-endpoint action

Next on my local machine we used the boto3 version of this similar thing:

* use the notebook: *Integration\_Examples/invoke\_using\_boto3*
* This didn’t seem any better or worse, just a different integration API

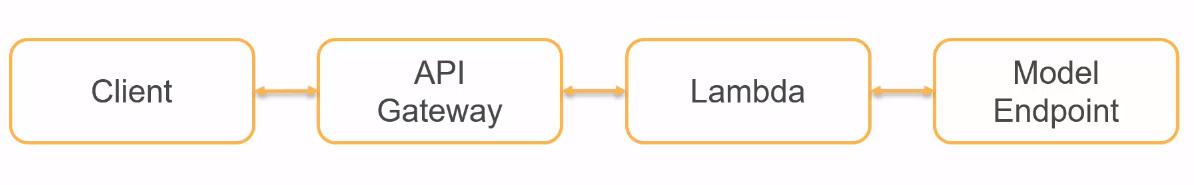
Next we moved on towards going more serverless and integration Lambda:

* use the notebook: *Integration\_Examples/invoke\_using\_boto3\_json*
* basically we just construct a JSON payload to send instead of the previous format of payload

Next we actually went to lambda:

* We setup a lambda function, along with an environment variable for our SageMaker endpoint (under the Configuration tab and Env Variables). We tested using his provided test payload.
  + The test didn’t work until I hit “deploy”

Next we deployed our lambda function as a REST API using the API Gateway service

* We first ran the test payload inside the API Gateway service
* Then we deployed the API Gateways
  + It yielded the following API Gateway url:
    - <https://l10ilf3zl4.execute-api.us-east-1.amazonaws.com/beta>
* Then we ran the test payload from the browser extension app called RESTer which is equivalent to POSTMAN from what I could tell
* Then we went back to the python notebook: *Integration\_Examples/invoke\_api\_gateway*

