

**Course Outline**

Throughout this course, we’ll be focusing on deployment tools and the machine learning workflow; answering a few big questions along the way:

* How do you decide on the correct machine learning algorithm for a given task?
* How can we utilize cloud ML services in SageMaker to work with interesting datasets or improve our algorithms?

To approach these questions, we’ll go over a number of real-world **case studies**, and go from task and problem formulation to deploying models in SageMaker. We’ll also utilize a number of SageMaker’s built-in algorithms.

**Case Studies**

Case studies are in-depth examinations of specific, real-world tasks. In our case, we’ll focus on three different problems and look at how they can be solved using various machine learning strategies. The case studies are as follows:

***Case Study 1 - Population Segmentation using SageMaker***

You’ll look at a portion of [US census data](https://www.census.gov/data.html) and, using a combination of unsupervised learning methods, extract meaningful components from that data and group regions by similar census-recorded characteristics. This case study will be a deep dive into Principal Components Analysis (PCA) and K-Means clustering methods, and the end result will be groupings that are used to inform things like localized marketing campaigns and voter campaign strategies.

***Case Study 2 - Detecting Credit Card Fraud***

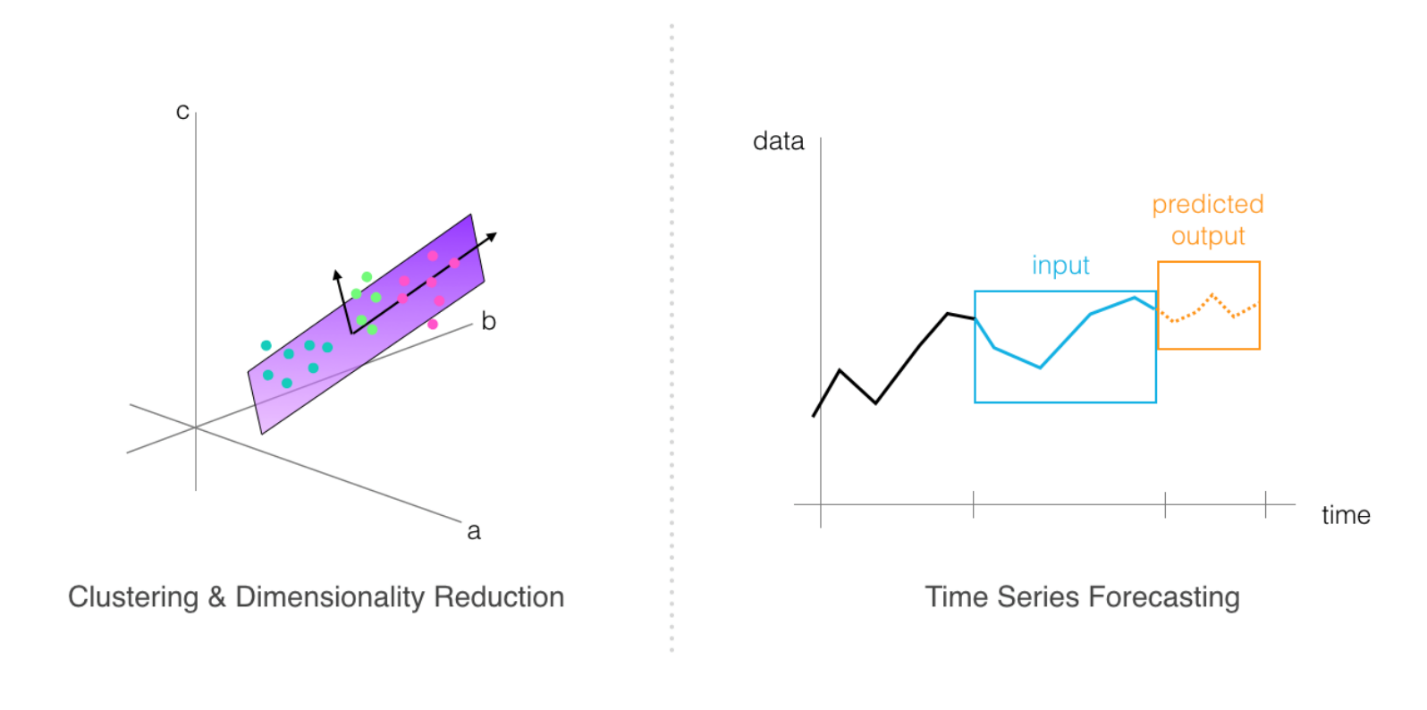
This case will demonstrate how to use supervised learning techniques, specifically SageMaker’s LinearLearner, for fraud detection. The payment transaction dataset we'll work with is unbalanced, with many more examples of valid transactions vs. fraudulent, and so you will investigate methods for compensating for this imbalance and tuning your model to improve its performance according to a specific product goal.

***Custom Models - Non-Linear Classification***

Adding on to what you have learned in the credit card fraud case study, you will learn how to manage cases where classes of data are not separable by a linear line. You'll train and deploy a custom, PyTorch neural network for classifying data.

***Case Study 3 - Time-Series Forecasting***

This case demonstrates how to train SageMaker's DeepAR model for forecasting predictions over time. Time-series forecasting is an active area of research because a good forecasting algorithm often takes in a number of different features and accounts for seasonal or repetitive patterns. In this study, you will learn a bit about creating features out of time series data and formatting it for training.



***Project: Plagiarism Detection***

You'll apply the skills that you've learned to a final project; building and deploying a plagiarism classification model. This project will test your ability to do [text] data processing and feature extraction, your ability to train and evaluate models according to an accuracy specification, and your ability to deploy a trained model to an endpoint.

By the end of this course, you should have all the skills you need to build, train and deploy models to solve tasks of your own design!

# K-Means Clustering

To perform population segmentation, one of our strategies will be to use k-means clustering to group data into similar clusters. To review, the k-means clustering algorithm can be broken down into a few steps; the following steps assume that you have n-dimensional data, which is to say, data with a discrete number of features associated with it. In the case of housing price data, these features include traits like house size, location, etc. **features** are just measurable components of a data point. K-means works as follows:

You select k, a predetermined number of clusters that you want to form. Then k points (centroids for k clusters) are selected at random locations in feature space. For each point in your training dataset:

1. You find the centroid that the point is closest to
2. And assign that point to that cluster
3. Then, for each cluster centroid, you move that point such that it is in the center of all the points that are were assigned to that cluster in step 2.
4. Repeat steps 2 and 3 until you’ve either reached convergence and points no longer change cluster membership \_or\_ until some specified number of iterations have been reached.

This algorithm can be applied to any kind of unlabelled data. You can watch a video explanation of the k-means algorithm, as applied to color image segmentation, below. In this case, the k-means algorithm looks at R, G, and B values as features, and uses those features to cluster individual pixels in an image!

## Color Image Segmentation

## Data Dimensionality

One thing to note is that it’s often easiest to form clusters when you have low-dimensional data. For example, it can be difficult, and often noisy, to get good clusters from data that has over 100 features. In high-dimensional cases, there is often a dimensionality reduction step that takes place before data is analyzed by a clustering algorithm. We’ll discuss PCA as a dimensionality reduction technique in the practical code example, later.

## The Github Repository

You can find a link to all of the exercise and project code for this course in the repository: <https://github.com/udacity/ML_SageMaker_Studies>. Copy and paste this repository into the Github clone option when you create your notebook instance!

**Note:** Once a notebook instance has been set up, by default, it will be **InService** which means that the notebook instance is running. This is important to know because the cost of a notebook instance is based on the length of time that it has been running. This means that once you are finished using a notebook instance you should **Stop** it so that you are no longer incurring a cost. Don't worry though, you won't lose any data provided you don't delete the instance. Just start the instance back up when you have time and all of your saved data will still be there.

NEXT

Create a SageMaker notebook instance for this course. This checklist is just meant to act as a check that you have set everything up correctly and are ready to move on!

Task List

* 

Open your [AWS console](https://console.aws.amazon.com/) and navigate to the SageMaker main page.

* 

Create a **new** notebook instance.

* 

Give that instance a descriptive name, and default role (no need to add access to any extra S3 buckets).

* 

Link that instance to the main [Github repository](https://github.com/udacity/ML_SageMaker_Studies) for this course!

* 

You may leave instance types and sizes as the default value.

## After: Clean up Resources

After you are done working with a notebook **make sure** that you stop the notebook from running, otherwise you may occur additional fees. You'll also have to **delete** all your endpoints, S3 resources, and notebooks at the end of this course, and will be given instructions to do so. **Do not skip the cleanup steps.**

# Notebook: Population Segmentation, Exercise

Now, you're ready to approach the task of population segmentation! As you follow along with this lesson, you are encouraged to open the referenced SageMaker notebooks. We will present a solution to you, but please try to work on a solution of your own, when prompted. Much of the value in this experience will come from experimenting with the code, **in your own way**.

To open this notebook:

* *Navigate to your SageMaker notebook instance, in the*[*SageMaker console*](https://console.aws.amazon.com/sagemaker/)*, which has been linked to the main*[*Github exercise repository*](https://github.com/udacity/ML_SageMaker_Studies)
* *Activate the notebook instance (if it is in a "Stopped" state), and open it via Jupyter*
* *Click on the exercise notebook in the Population\_Segmentation directory.*

You may also directly view the exercise and solution notebooks via the repository at the following links:

* [Exercise notebook](https://github.com/udacity/ML_SageMaker_Studies/blob/master/Population_Segmentation/Pop_Segmentation_Exercise.ipynb)
* [Solution notebook](https://github.com/udacity/ML_SageMaker_Studies/blob/master/Population_Segmentation/Pop_Segmentation_Solution.ipynb)

**The solution notebook is meant to be consulted if you are stuck or want to check your work.**

## Notebook Outline

We'll go over the following steps to complete the notebook.

* Load in and explore population data
* Perform dimensionality reduction with a deployed PCA model
* Cluster components with K-Means
* Visualize the results

## Later: Delete Resources

At the end of this exercise, and intermittently, you will be reminded to delete your endpoints and resources so that you do not incur any extra processing or storage fees!

## The Range of Values

One thing you may have noticed, especially when creating density plots and comparing feature values, is that that the values in each feature column are in quite a wide range; some are very large numbers or small, floating points.

Now, our end goal is to cluster this data and clustering relies on looking at the perceived similarities and differences between features! So, we want our model to be able to look at these columns and **consistently** measure the relationships between features.

*To make sure the feature measurements are consistent and comparable, you’ll scale all of the*numerical*features into a range between 0 and 1. This is a pretty typical****normalization****step.*

Below, is what the exercise looks like in the main notebook.

### EXERCISE: Normalize the data

You need to standardize the scale of the numerical columns in order to consistently compare the values of different features. You can use a MinMaxScaler to transform the numerical values so that they all fall between 0 and 1.

*# scale numerical features into a normalized range, 0-1*

*# store them in this dataframe*

counties\_scaled = **None**

**Try to complete this task on your own**, in the exercise notebook, and if you get stuck or want to consult a solution, I’ll go over my solution, next.

**PCA**

Principal Component Analysis (PCA) attempts to reduce the number of features within a dataset while retaining the “principal components”, which are defined as *weighted* combinations of existing features that:

1. Are uncorrelated with one another, so you can treat them as independent features, and
2. Account for the largest possible variability in the data!

So, depending on how many components we want to produce, the first one will be responsible for the largest variability on our data and the second component for the second-most variability, and so on. Which is exactly what we want to have for clustering purposes!

PCA is commonly used when you have data with many *many* features.

You can learn more about the details of the PCA algorithm in the video, below.

*Now, in our case, we have data that has 34-dimensions and we’ll want to use PCA to find combinations of features that produce the most variability in the population dataset.*

The idea is that components that cause a larger variance will help us to better differentiate between data points and (therefore) better separate data into clusters.

So, next, I’ll go over how to use **SageMaker’s built-in PCA model** to analyze our data.

## Creating a KMeans Estimator

Now that we’ve run the original data through PCA and moved from 34-dimensional data to 7-dimensional component data, we’re better prepared to actually cluster this data! So, now, I’ll ask you to use these components that we’ve gotten from our training data and cluster counties using k-means.

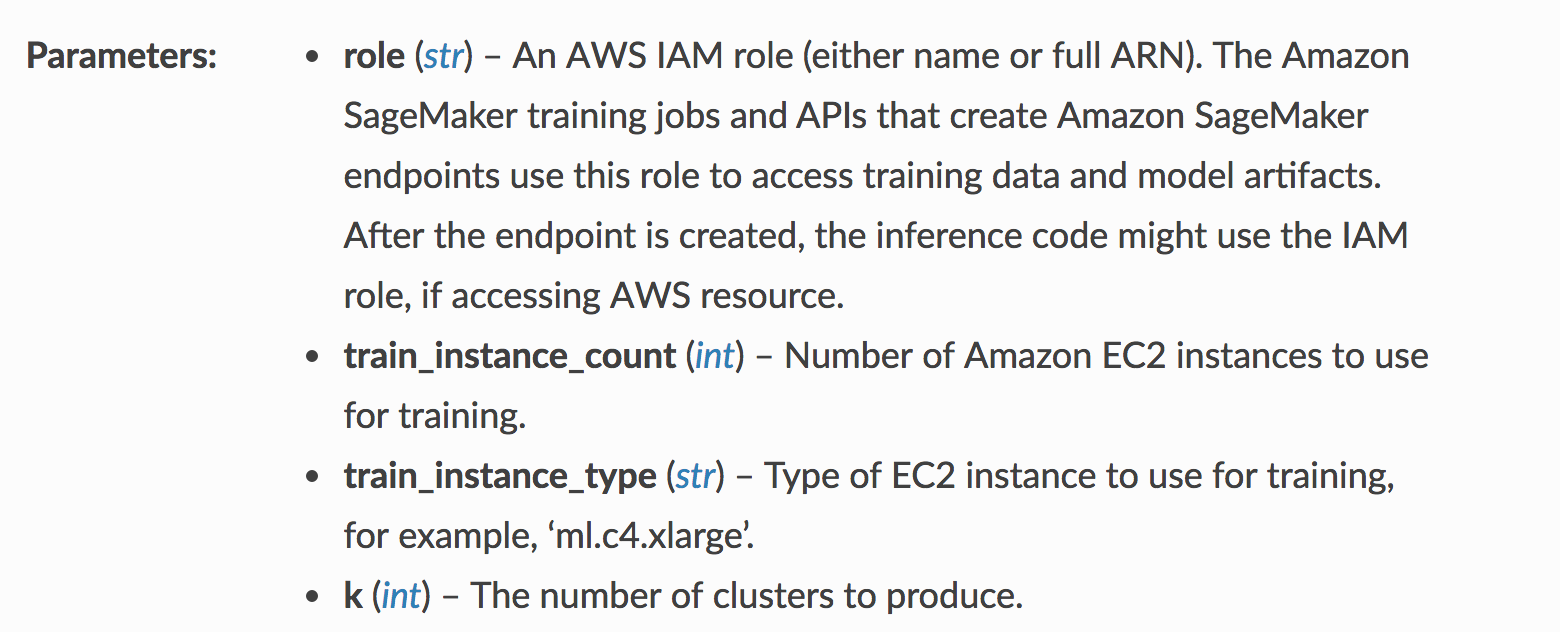
You'll instantiate a KMeans estimator, by specifying specific model arguments and passing them into a KMeans constructor ([documentation, here](https://sagemaker.readthedocs.io/en/stable/kmeans.html)). Knowing how to read documentation is an important skill for learning to create models on your own!

Here is what this exercise looks like in the main, exercise notebook:

### EXERCISE: Define a k-means model

Your task will be to instantiate a k-means model. A KMeans estimator requires a number of parameters to be instantiated, which allow us to specify the type of training instance to use, and the model hyperparameters.

*# define a KMeans estimator*



Some parameters in the KMeans documentation

## General Estimator Parameters

From the documentation, you can see that you'll need to specify the IAM role (which we defined when creating the notebook instance), and details about the instance type to use for training.

Most of SageMaker's built-in algorithms are based off of an [EstimatorBase object](https://sagemaker.readthedocs.io/en/stable/estimators.html#sagemaker.estimator.EstimatorBase), which allows you to specify additional parameters. It is good practice to be specific about two additional parameters:

* **output\_path (str)** – S3 location for saving the training result (model artifacts and output files). If not specified, results are stored to a default bucket.
* **sagemaker\_session (sagemaker.session.Session)** – Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed.

## Model-Specific Parameters

You'll also notice a parameter k for the number of clusters this model should produce as output. This parameter is specific to the k-means model, and for different models, you'll see different required model parameters.

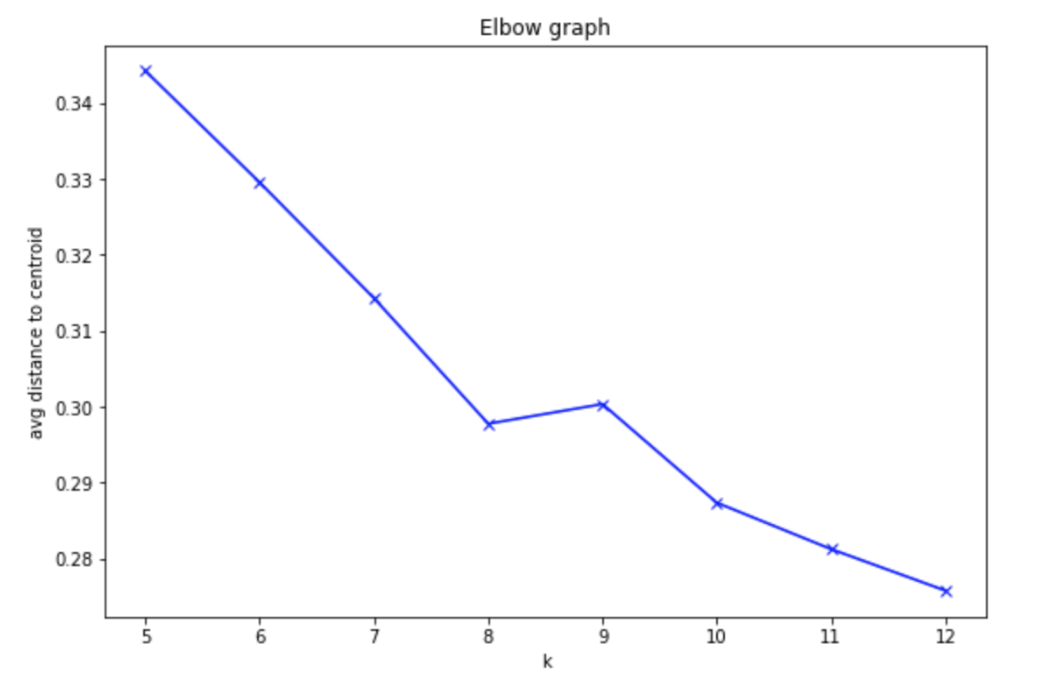
## Choosing a "Good" K

One method for choosing a "good" k, is to choose based on empirical data.

* A bad k would be one so high that only one or two very close data points are near it, and
* Another bad k would be one so low that data points are really far away from the centers.

You want to select a k such that data points in a single cluster are close together but that there are enough clusters to effectively separate the data. You can approximate this separation by measuring how close your data points are to each cluster center; the average centroid distance between cluster points and a centroid. After trying several values for k, the centroid distance typically reaches some "elbow"; it stops decreasing at a sharp rate and this indicates a good value of k.

The graph below indicates the average distance—between our component data and cluster centroids—for a value of k between 5 and 12.



## Training Job

After creating a KMeans estimator, I also want you to proceed with creating a training job. You'll have to format your data correctly for this job and make sure you are passing in the reduced-dimensionality training data. It may be helpful to reference the PCA training job code.

Here is what these exercises look like in the exercise notebook:

### EXERCISE: Create formatted, k-means training data

Just as before, you should convert the counties\_transformed df into a numpy array and then into a RecordSet. This is the required format for passing training data into a KMeans model.

*# convert the transformed dataframe into record\_set data*

### EXERCISE: Train the k-means model

Pass in the formatted training data and train the k-means model.

%%time

*# train kmeans*

After you are done with these steps, you can move on to model deployment!

## Getting Predicted Clusters

After you've trained your KMeans estimator, you can deploy it and apply it to our data to get resultant clusters.

### EXERCISE: Deploy the k-means model

Deploy the trained model to create a kmeans\_predictor.

%%time

*# deploy the model to create a predictor*

kmeans\_predictor = **None**

### EXERCISE: Pass in the training data and assign predicted cluster labels

After deploying the model, you can pass in the k-means training data, as a numpy array, and get resultant, predicted cluster labels for each data point.

*# get the predicted clusters for all the kmeans training data*

cluster\_info=**None**

If you finish this exercise, you should be able to proceed with some interesting visualizations that give you the ability to explore how counties are clustered and what that means as far as features that define the similarity between counties.

## Shutting Down the Endpoint

After you successfully make predictions and assign each county to a cluster, you can delete your KMenas endpoint.

# Model Attributes & Explainability

Explaining the result of the modeling is an important step in making use of our analysis. By combining PCA and k-means, and the information contained in the model attributes within a SageMaker trained model, you can learn about a population and remark on some patterns you've found, based on the data.

To access the k-means model attributes, you'll find the following guidance in the main, exercise notebook.

### EXERCISE: Access the k-means model attributes

Extract the k-means model attributes from where they are saved as a TAR file in an S3 bucket.

You'll need to access the model by the k-means training job name, and then unzip the file into model\_algo-1. Then you can load that file using MXNet, as before.

*# download and unzip the kmeans model file*

*# use the name model\_algo-1*

*# get the trained kmeans params using mxnet*

kmeans\_model\_params = **None**

print(kmeans\_model\_params)

Save the model attributes as kmeans\_model\_params; you should see that there is only 1 set of model parameters contained within the k-means model: the **cluster centroid locations** in PCA-transformed, component space.

### Cluster Centroids

You know that each of the counties in our US county data, is assigned to a cluster and that indicates something about groupings found by k-means and similarities between counties in the same cluster. But, what exactly are the features that these clusters have in common?

*For example, how is cluster 1 any different than cluster 2?*

This is what we aim to define by looking at the location of cluster centroids in component space. Since each cluster is defined in component space, we can look at how eighty each component is in defining a certain cluster. Then, we can go one step further and map the components back to the original data features. I encourage you to look at the visualization code in the exercise notebook and watch the next video to see some complete code.

# Clean up Resources

It is good practice to always clean up and delete any resources that you are no longer using. That is, after you complete an exercise, and you are done with predictions and data analysis, you should get rid of any:

* Data source in S3 that you are no longer using
* Endpoint configuration files that you no longer need
* Endpoints that you will no longer use
* CloudWatch logs that are no longer useful

## Deleting Endpoints

In the notebook, we have usually included code to delete your endpoints after creating some predictions, for example:

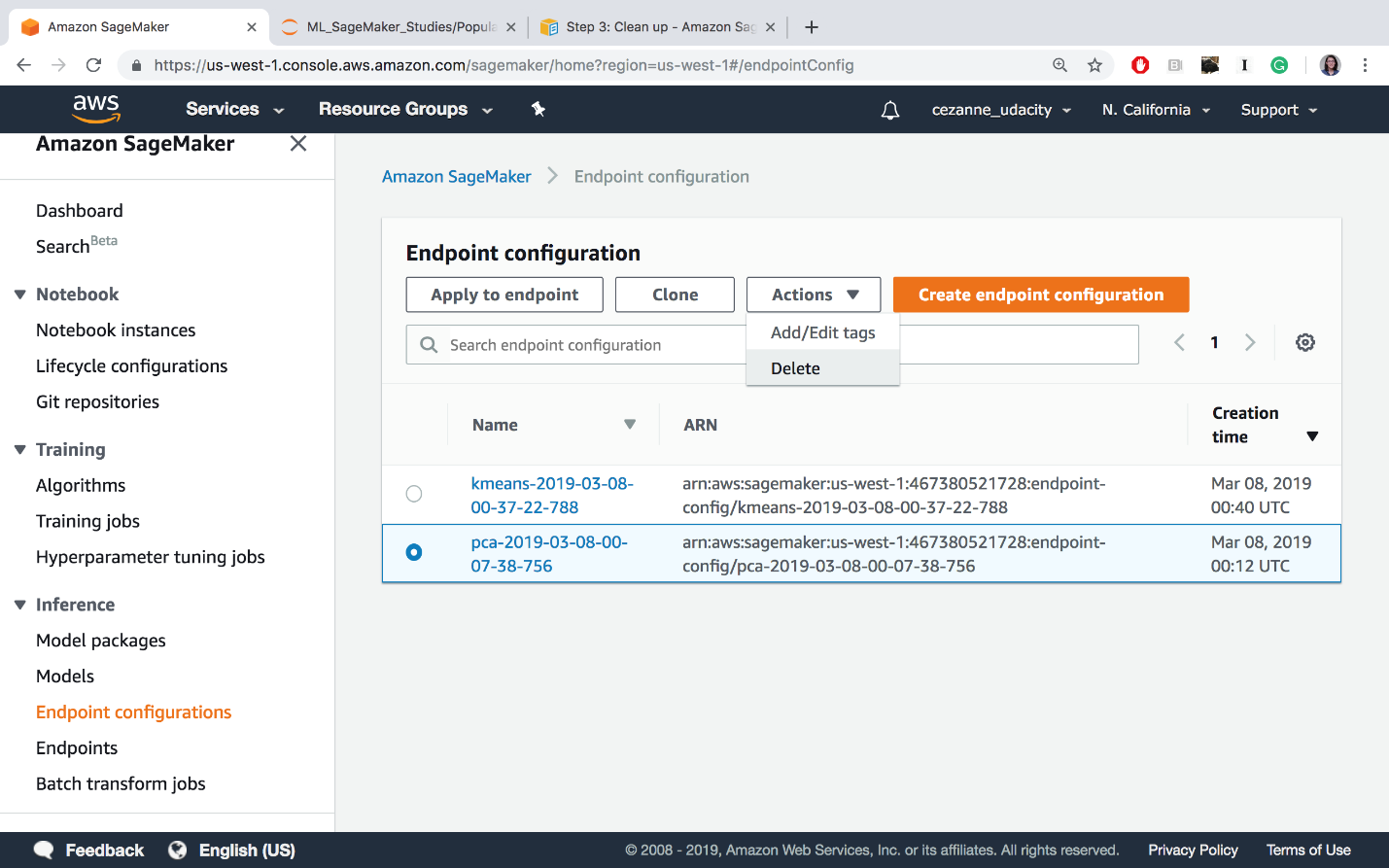
*# delete predictor endpoint*

session.delete\_endpoint(predictor.endpoint)

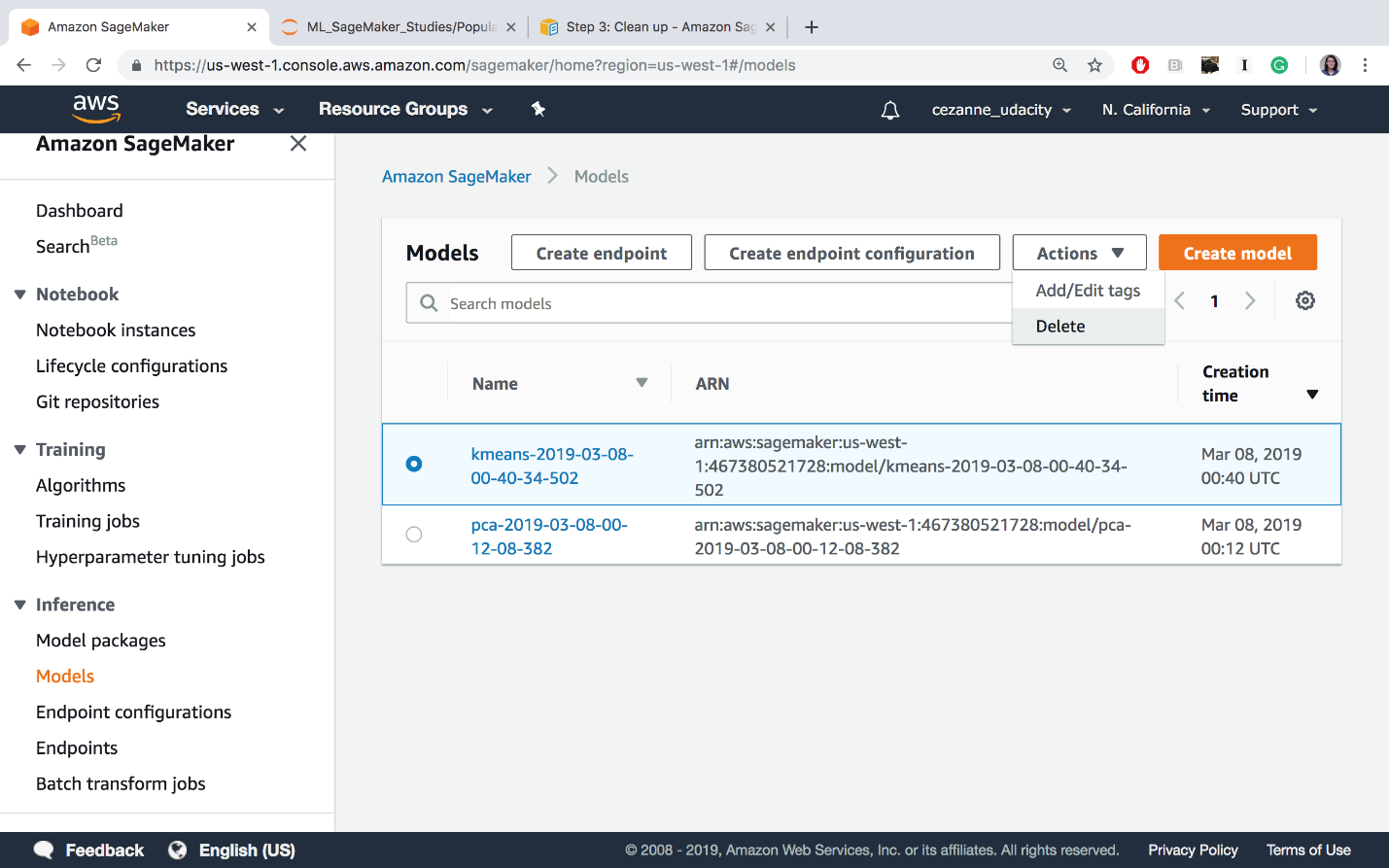
## Thorough Clean up

You can find a link for instructions on cleaning up all your resources, [in this documentation](https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-cleanup.html) and I will go over some of these details, next.

* Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/> and delete the following resources:
* The endpoint configuration.
* The model.

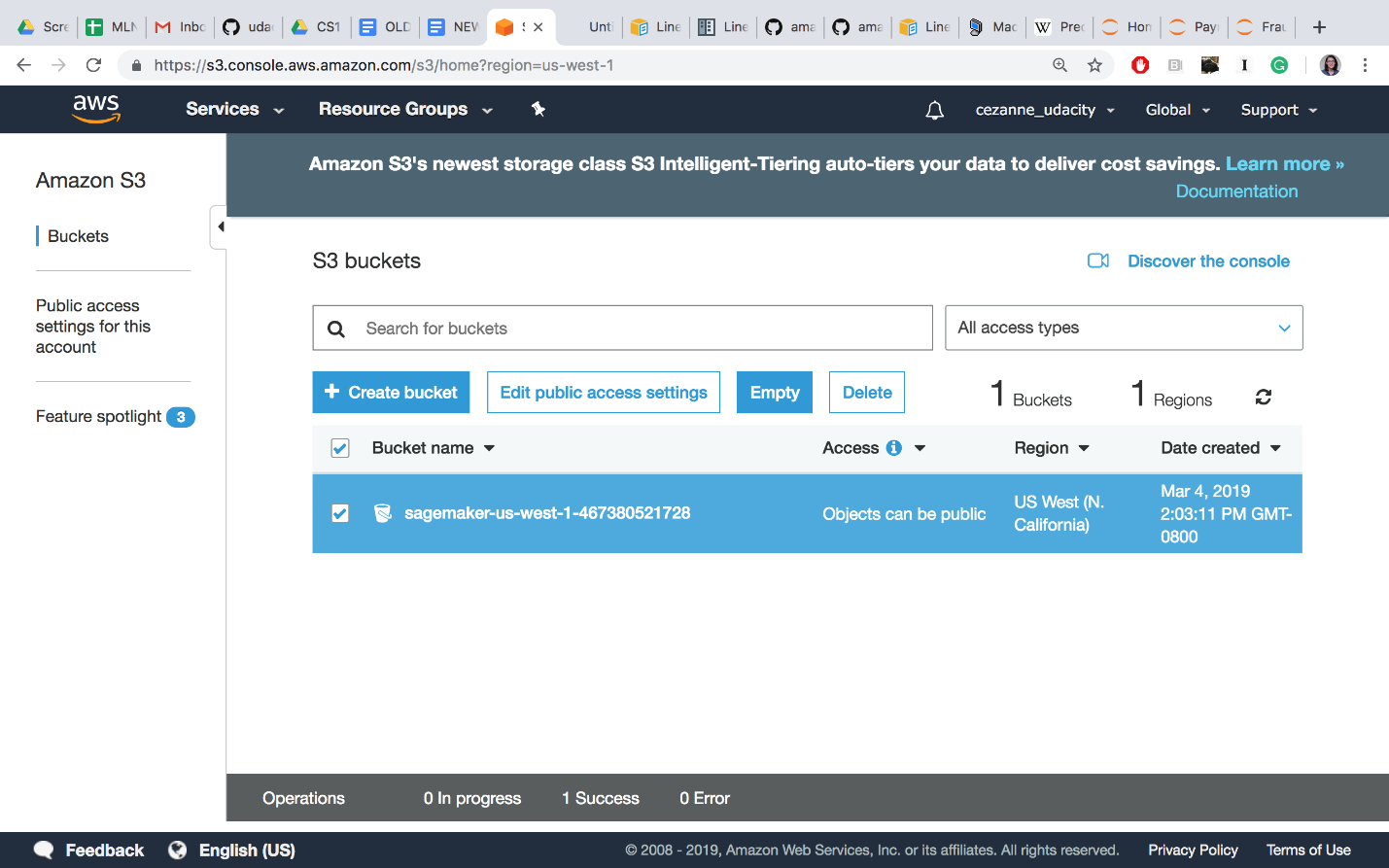


Delete endpoint config files.



Deleting models

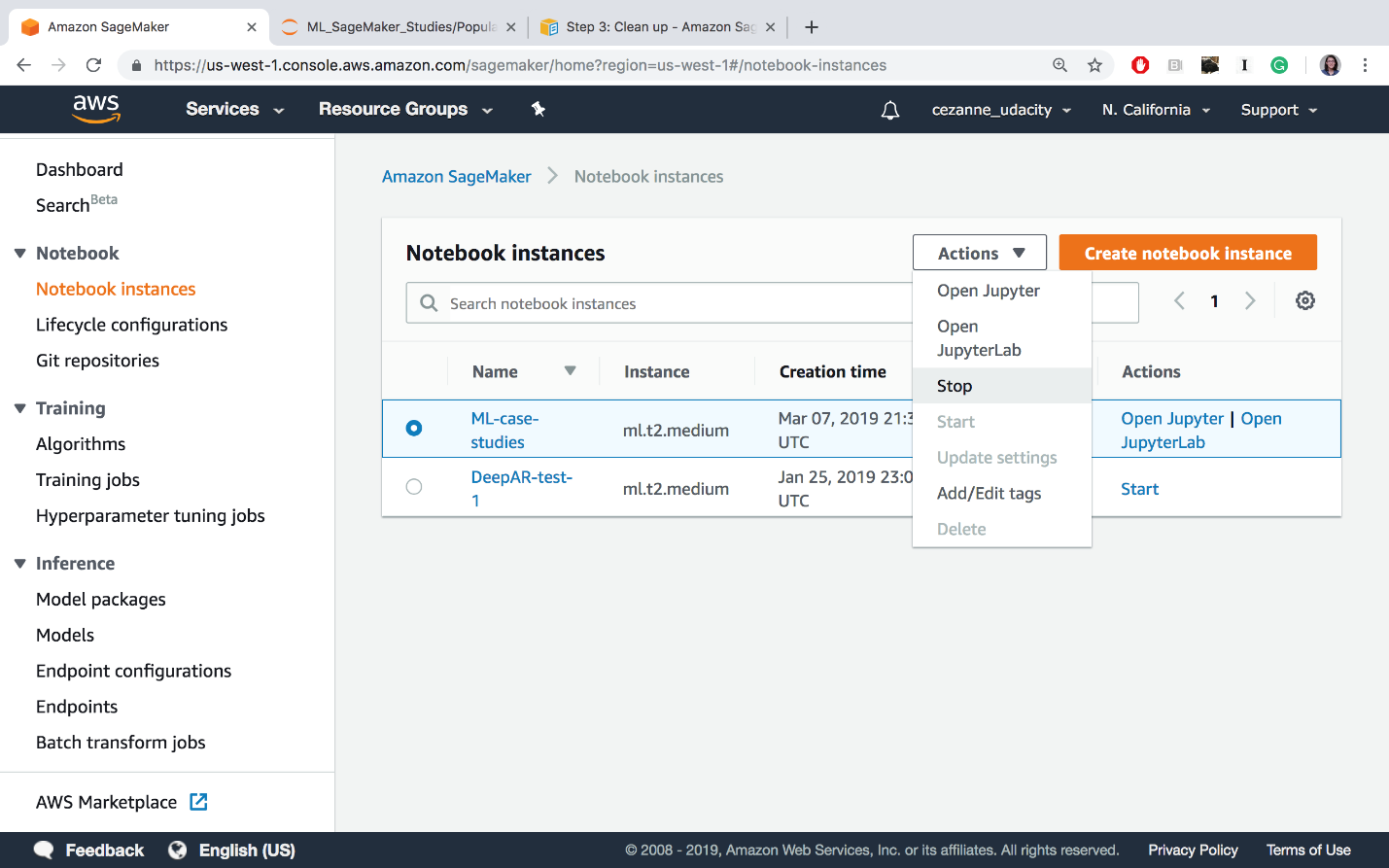
* Open the Amazon S3 console at <https://console.aws.amazon.com/s3/> and delete or empty the bucket that you created for storing model artifacts and the training dataset.



Delete or empty your S3 bucket (empty is recommended until the end of the course, when you should delete this bucket entirely)

* Open the Amazon CloudWatch console at <https://console.aws.amazon.com/cloudwatch/> and delete all of the log groups that have names starting with /aws/sagemaker/.

At the end of this course, you may also choose to delete the entire notebook instance and IAM Role, but you may keep these as is, for now. In between lessons, if you are taking a break, you may want to **Stop** your notebook and pause it from continuously running.



Stopping the ML-case-studies notebook

Cleaning up resources at the end of an exercise or lesson is a great practice to get into!

## IMPORTANT

**To avoid incurring additional charges, it is suggested that you DELETE any unused notebooks and data resources on S3 and CloudWatch.**