LEX DP-100 - Designing and Implementing a Data Science Solution on Azure

# Scope for DP-100: Designing and Implementing a Data Science Solution on Azure course:

* Set up an Azure Machine Learning workspace (30-35%)
* Run experiments and train models (25-30%)
* Optimize and manage models (20-25%)
* Deploy and consume models (20-25%)

[Exam Outline](https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE3VUjA)

Navigate to the next topic to access the Microsoft Learn content

Complete the self assessment to mark your course completion

# [Create machine learning models](https://learn.microsoft.com/en-US/training/paths/create-machine-learn-models/?WT.mc_id=api_CatalogApi)

* 5 hr 20 min
* Learning Path
* 0 of 5 modules completed

Intermediate

AI Engineer

Data Scientist

Developer

Student

Visual Studio Code

Azure

[Create machine learning models - Training | Microsoft Learn](https://learn.microsoft.com/en-US/training/paths/create-machine-learn-models/?WT.mc_id=api_CatalogApi)

Microsoft Learn provides several interactive ways to get an introduction to classic machine learning. These learning paths will get you productive on their own, and also are an excellent base for moving on to deep learning topics.

From the most basic classical machine learning models, to exploratory data analysis and customizing architectures, you’ll be guided by easy to digest conceptual content and interactive Jupyter notebooks, all without leaving your browser.

Choose your own path depending on your educational background and interests.

### [Option 1: The complete course: Foundations of data science for machine learning](https://learn.microsoft.com/en-us/training/paths/machine-learning-foundations-using-data-science/)

This path is recommended for most people. It has all the same modules as the other two learning paths with a custom flow that maximizes reinforcement of concepts. If you want to learn about both the underlying concepts and how to get into building models with the most common machine learning tools this path is for you. It's also the best path if you plan to move beyond classic machine learning, and get an education in deep learning and neural networks, which we only introduce here.

### [Option 2: The Understand data science for machine learning learning path](https://learn.microsoft.com/en-us/training/paths/understand-machine-learning)

If you are looking to understand how machine learning works and don't have much mathematical background then this path is for you. It makes no assumptions about previous education (other than a light familiarity with coding concepts) and teaches with code, metaphor, and visual that give you the ah ha moment. It's hands-on, but focuses more on understanding fundamentals and less on the power of the tools and libraries available.

## ✔ Option 3: The Create machine learning models learning path

If you already have some idea what machine learning is about or you have a strong mathematical background you may best enjoy jumping right in to the Create Machine Learning Models learning path. These modules teach some machine learning concepts, but move fast so they can get to the power of using tools like scikit-learn, TensorFlow, and PyTorch. This learning path is also the best one for you if you're looking for just enough familiarity to understand machine learning examples for products like Azure ML or Azure Databricks.

✔ You are currently on this path, scroll down to begin.

**Testing hypotheses**

Data exploration and analysis is typically an *iterative* process, in which the data scientist takes a sample of data and performs the following kinds of tasks to analyze it and test hypotheses:

* **Clean data** to handle errors, missing values, and other issues.
* **Apply statistical techniques to better understand the data** and how the sample might be expected to represent the real-world population of data, allowing for random variation.
* **Visualize data** to determine relationships between variables, and in the case of a machine learning project, identify *features* that are potentially predictive of the *label*.
* **Revise the hypothesis** and repeat the process.

**Real-world data will always have issues, but data scientists can often overcome these** issues by:

* Checking for missing values and badly recorded data.
* Considering removing obvious outliers.
* Examining what real-world factors might affect their analysis and determining if their dataset size is large enough to reduce the impact of these factors.
* Checking for biased raw data and considering their options to fix the bias, if found.

bike\_data['day'] = pd.DatetimeIndex(bike\_data['dteday']).day

# **Discover new regression models**

In Unit 2, we looked at fitting a straight line to data points. However, regression can fit many kinds of relationships, including those with multiple factors and those where the importance of one factor depends on another.

## Experimenting with models

Regression models are often chosen because they work with small data samples, are robust, easy to interpret, and a variety exist.

**Linear regression** is the simplest form of regression, with no limit to the number of features used. Linear regression comes in many forms, often named by the number of features used and the shape of the curve that fits.

**Decision trees** take a step-by-step approach to predicting a variable. If we think of our bicycle example, the decision tree might be first split examples between ones that are during Spring/Summer and Autumn/Winter, make a prediction based on the day of the week. Spring/Summer-Monday might have a bike-rental rate of 100 per day, while Autumn/Winter-Monday might have a rental rate of 20 per day.

**Ensemble algorithms** construct not just one decision tree, but a large number of trees, allowing better predictions on more complex data. Ensemble algorithms, such as Random Forest, are widely used in machine learning and data science due to their strong prediction abilities.

Data scientists often experiment with using different models. In the following exercise, we'll experiment with different types of models to compare how they perform on the same data.

# **Improve models with hyperparameters**

Simple models with small datasets can often be fit in a single step, while larger datasets and more complex models must be fit by repeatedly using the model with training data and comparing the output with the expected label. If the prediction is accurate enough, we consider the model trained. If not, we adjust the model slightly and loop again.

Hyperparameters are values that change the way that the model is fit during these loops. Learning rate, for example, is a hyperparameter that sets how much a model is adjusted during each training cycle. A high learning rate means a model can be trained faster; but if it’s too high, the adjustments can be so large that the model is never "finely tuned" and not optimal.

## Preprocessing data

Preprocessing refers to changes you make to your data before it's passed to the model. We've previously read that preprocessing can involve cleaning your dataset. While this is important, preprocessing can also include changing the format of your data so it's easier for the model to use. For example, data described as "red," "orange," "yellow," "lime," and "green" might work better if converted into a format more native to computers, such as numbers stating the amount of red and the amount of green.

### Scaling features

The most common preprocessing step is to scale features so they fall between zero and one. For example, the weight of a bike and the distance a person travels on a bike may be two very different numbers, but by scaling both numbers to between zero and one allows models to learn more effectively from the data.

### Using categories as features

In machine learning, you can also use categorical features such as "bicycle," "skateboard," or "car." These features are represented by 0 or 1 values in **one-hot vectors**; vectors that have a 0 or 1 for each possible value. For example, bicycle, skateboard, and car might respectively be (1,0,0), (0,1,0), and (0,0,1).

## Optimize Hyperparameters

Take a look at the **GradientBoostingRegressor** estimator definition in the preceding output and note that it, like the other estimators we tried previously, includes a large number of parameters that control the way the model is trained. In machine learning, the term parameters refers to values that can be determined from data; values that you specify to affect the behavior of a training algorithm are more correctly referred to as hyperparameters.

The specific hyperparameters for an estimator vary based on the algorithm that the estimator encapsulates. In the case of the **GradientBoostingRegressor** estimator, the algorithm is an ensemble that combines multiple decision trees to create an overall predictive model. You can learn about the hyperparameters for this estimator in the [Scikit-Learn documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html).

We won't go into the details of each hyperparameter here, but they work together to affect the way the algorithm trains a model. In many cases, the default values provided by Scikit-Learn will work well, but there could be some advantage in modifying hyperparameters to get better predictive performance or reduce training time.

So how do you know what hyperparameter values you should use? Well, in the absence of a deep understanding of how the underlying algorithm works, you'll need to experiment. Fortunately, SciKit-Learn provides a way to tune hyperparameters by trying multiple combinations and finding the best result for a given performance metric.

Let's try using a grid search approach to try combinations from a grid of possible values for the **learning\_rate** and **n\_estimators** hyperparameters of the **GradientBoostingRegressor** estimator.

For example, in scientific studies, the R-squared may need to be above 0.95 for a regression model to be considered reliable. In other domains, an R-squared of just 0.3 may be sufficient if there is extreme variability in the dataset.

**The R-squared metric is a measure of how much of the variance the model can explain.**

# Regression Challenge

[ml-basics/challenges/02 - Real Estate Regression Challenge.ipynb at master · MicrosoftDocs/ml-basics · GitHub](https://github.com/MicrosoftDocs/ml-basics/blob/master/challenges/02%20-%20Real%20Estate%20Regression%20Challenge.ipynb)

**Classification**

* **Accuracy**: (TP+TN)/(TP+TN+FP+FN) - out all of the predictions, how many were correct?
* **Recall**: TP/(TP+FN) - of all the cases that *are* positive, how many did the model identify?
* **Precision**: TP/(TP+FP) - of all the cases that the model predicted to be positive, how many actually *are* positive?

# Create [multiclass](https://learn.microsoft.com/en-us/training/modules/train-evaluate-classification-models/6-multiclass-classification) classification models unit 6/9

It's also possible to create multiclass classification models, in which there are more than two possible classes. For example, the health clinic might expand the diabetes model to classify patients as:

* Non-diabetic
* Type-1 diabetic
* Type-2 diabetic

The individual class probability values would still add up to a total of 1 as the patient is definitely in only one of the three classes, and the most probable class would be predicted by the model.

## Using Multiclass classification models

Multiclass classification can be thought of as a combination of multiple binary classifiers. There are two ways in which you approach the problem:

* **One vs Rest (OVR)**, in which a classifier is created for each possible class value, with a positive outcome for cases where the prediction is this class, and negative predictions for cases where the prediction is any other class. For example, a classification problem with four possible shape classes (square, circle, triangle, hexagon) would require four classifiers that predict:
  + square or not
  + circle or not
  + triangle or not
  + hexagon or not
* **One vs One (OVO)**, in which a classifier for each possible pair of classes is created. The classification problem with four shape classes would require the following binary classifiers:
  + square or circle
  + square or triangle
  + square or hexagon
  + circle or triangle
  + circle or hexagon
  + triangle or hexagon

In both approaches, the overall model must take into account all of these predictions to determine which single category the item belongs to.

Fortunately, in most machine learning frameworks, including Scikit-Learn, implementing a multiclass classification model isn't significantly more complex than binary classification - and in most cases, the estimators used for binary classification implicitly support multiclass classification by abstracting an OVR algorithm, an OVO algorithm, or by allowing a choice of either.

[Wine Classification Challenge](https://github.com/MicrosoftDocs/ml-basics/blob/master/challenges/03%20-%20Wine%20Classification%20Challenge.ipynb)

**Clustering**

# Evaluate different types of clustering

Completed100 XP

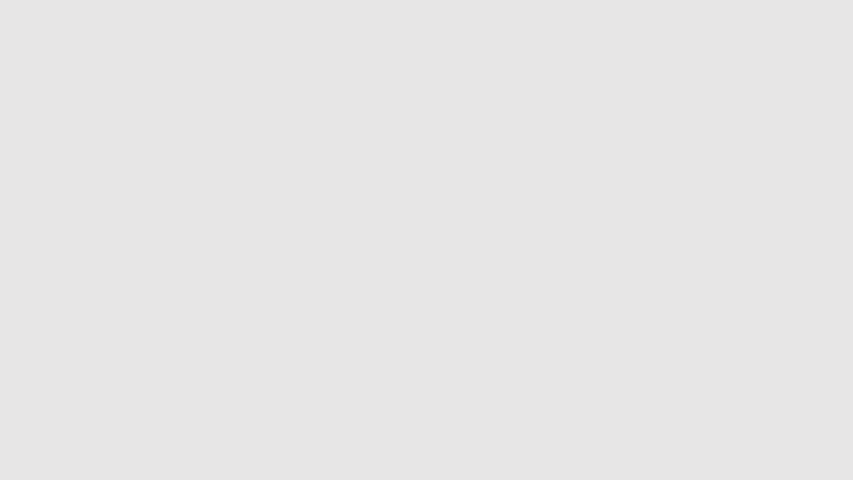
* 5 minutes

## Training a clustering model

There are multiple algorithms you can use for clustering. One of the most commonly used algorithms is K-Means clustering that, in its simplest form, consists of the following steps:

1. The feature values are vectorized to define n-dimensional coordinates (where n is vthe number of features). In the flower example, we have two features: number of petals and number of leaves. So, the feature vector has two coordinates that we can use to conceptually plot the data points in two-dimensional space.
2. You decide how many clusters you want to use to group the flowers - call this value ***k***. For example, to create three clusters, you would use a k value of 3. Then k points are plotted at random coordinates. These points become the center points for each cluster, so they're called centroids.
3. Each data point (in this case a flower) is assigned to its nearest centroid.
4. Each centroid is moved to the center of the data points assigned to it based on the mean distance between the points.
5. After the centroid is moved, the data points may now be closer to a different centroid, so the data points are reassigned to clusters based on the new closest centroid.
6. The centroid movement and cluster reallocation steps are repeated until the clusters become stable or a predetermined maximum number of iterations is reached.

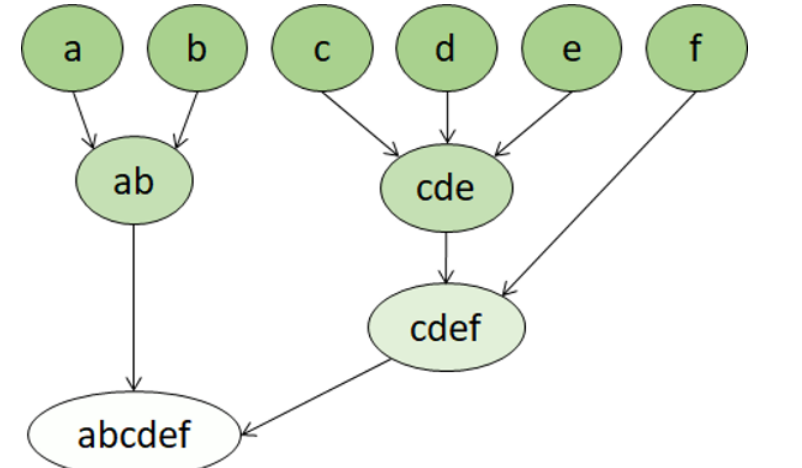
The following animation shows this process:



## Hierarchical Clustering

Hierarchical clustering is another type of clustering algorithm in which clusters themselves belong to larger groups, which belong to even larger groups, and so on. The result is that data points can be clusters in differing degrees of precision: with a large number of very small and precise groups, or a small number of larger groups.

For example, if we apply clustering to the meanings of words, we may get a group containing adjectives specific to emotions ('angry,' 'happy,' and so on). This group belongs to a group containing all human-related adjectives ('happy,' 'handsome,' 'young'), which belongs to an even higher group containing all adjectives ('happy,' 'green,' 'handsome,' 'hard,' and so on).

****

K-Means Clustering

The algorithm we'll use to create our test clusters is *K-Means*. This is a commonly used clustering algorithm that separates a dataset into *K* clusters of equal variance. The number of clusters, *K*, is user-defined. The algorithm has the following steps:

1. A set of K centroids are randomly chosen.
2. Clusters are formed by assigning the data points to their closest centroid.
3. The mean of each cluster is computed and the centroid is moved to the mean.
4. Steps 2 and 3 are repeated until a stopping criteria is met. Typically, the algorithm terminates when each new iteration results in negligible movement of centroids and the clusters become static.
5. When the clusters stop changing, the algorithm has *converged*, defining the locations of the clusters. Note that the random starting point for the centroids means that re-running the algorithm could result in slightly different clusters, so training usually involves multiple iterations, re-initializing the centroids each time, and the model with the best WCSS (*within cluster sum of squares*) is selected.

Let's try using K-Means on our seeds data with a K value of 3.