

LESSON3 - MODEL TRAINING

Section 1: LESSON OVERVIEW

Model training is one of the most important processes in machine learning (ML), used to transform data into trained ML models. It allows you to build, to train and to check the quality of your ML models.

In the lesson we will discuss data management, preparation and handling which are essential to master as having proper, accurate and clean data is probably the most important ingredient for building successful ML models. Specifically we will go over:

- **Data importing and transformation**
- The **data management** process, including:
 - The use of **datastores** and **datasets**
 - **Versioning**
 - **Feature engineering**
 - How to monitor for **data drift**

Next, we will introduce the basics of **model training**, covering:

- The **core model training process**
- The two fundamental ML models: **classifier** and **regressor**
- The **model evaluation process** and **relevant metrics**

Finally we will conclude with an introduction to two core techniques used to make decisions based on multiple (rather than single) trained models to improve the performance of the predictions:

- **Ensemble learning**
- **Automated ML**

Section 3: DATA IMPORT & TRANSFORMATION

Most ML algorithms are highly sensitive to the quality of the data they train on. **Data import and transformation** is very important in the process of preparing the input for the ML models. One key process is **data wrangling**, which cleans, transforms/restructures and potentially enriches data to make it more appropriate for data analysis. Data wrangling is an **iterative process**, requiring a lot of trials and errors: you do some transformation on the data, then check the results, then come back to data transformation to make improvements, check again, and so on... A few steps are involved in data wrangling:

- **Data discovery and exploration:** exploring the raw data and check the general quality of the dataset (e.g. counting the number of missing values)
- **Data transformation:** transform the raw data by **restructuring** (e.g. normalise features values), **normalising** and **cleaning** (e.g. imputation of missing values and any error detection) the data
- **Data validation and publication:** the data is made available to the processes through which you train your ML algorithms.

Section 4: Lab (Import, transform & export data)

Lab Overview

In this lab you learn how to import your own data in the designer to create custom solutions. There are two ways you can import data into the designer in Azure Machine Learning Studio:

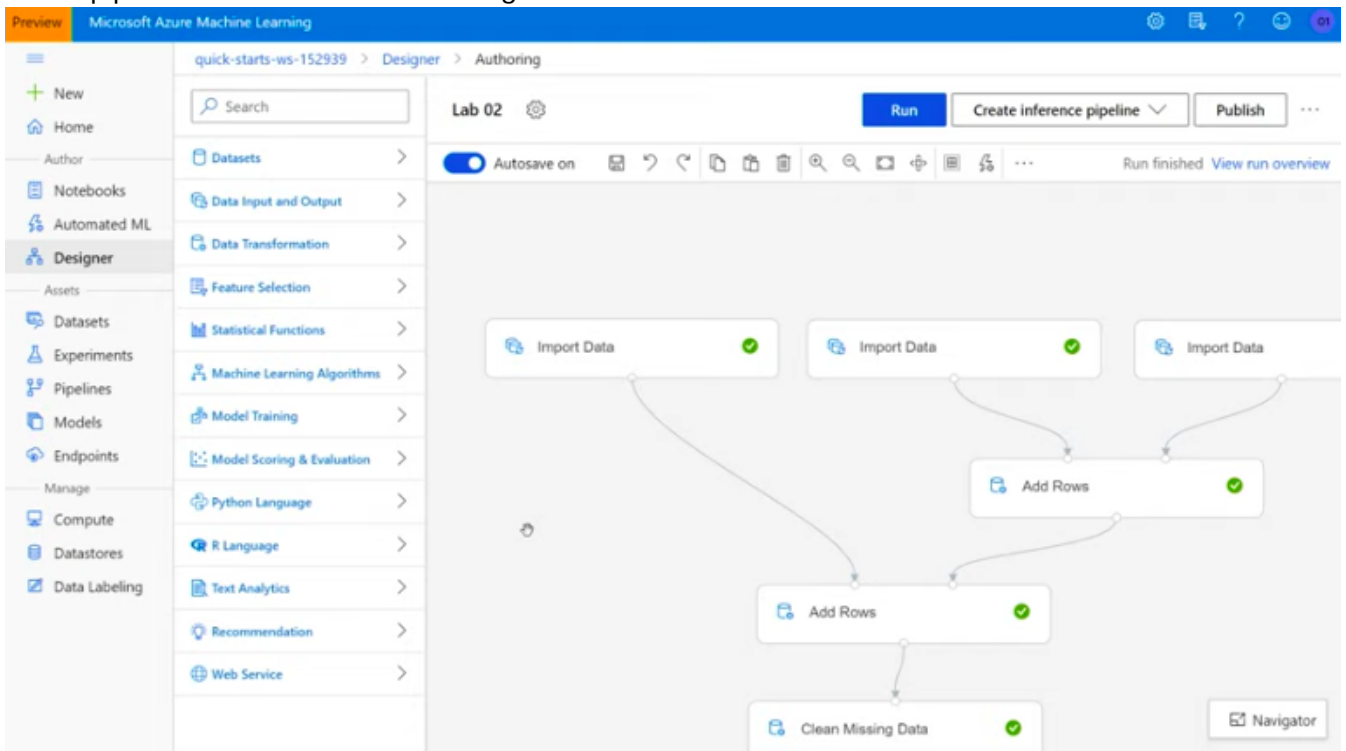
- Azure Machine Learning datasets: Register datasets in Azure Machine Learning to enable advanced features that help you manage your data.
- Import Data module: Use the Import Data module to directly access data from online datasources.

While the use of datasets is recommended to import data, you can also use the **Import Data module** from the designer. Data comes into the designer from either a Datastore or from Tabular Datasets. Datastores will be covered later in this course, but just for a quick definition, you can use Datastores to access your storage without having to hard code connection information in your scripts. As for the second option, the Tabular datasets, the following datasources are supported in the designer: Delimited files, JSON files, Parquet files or SQL queries.

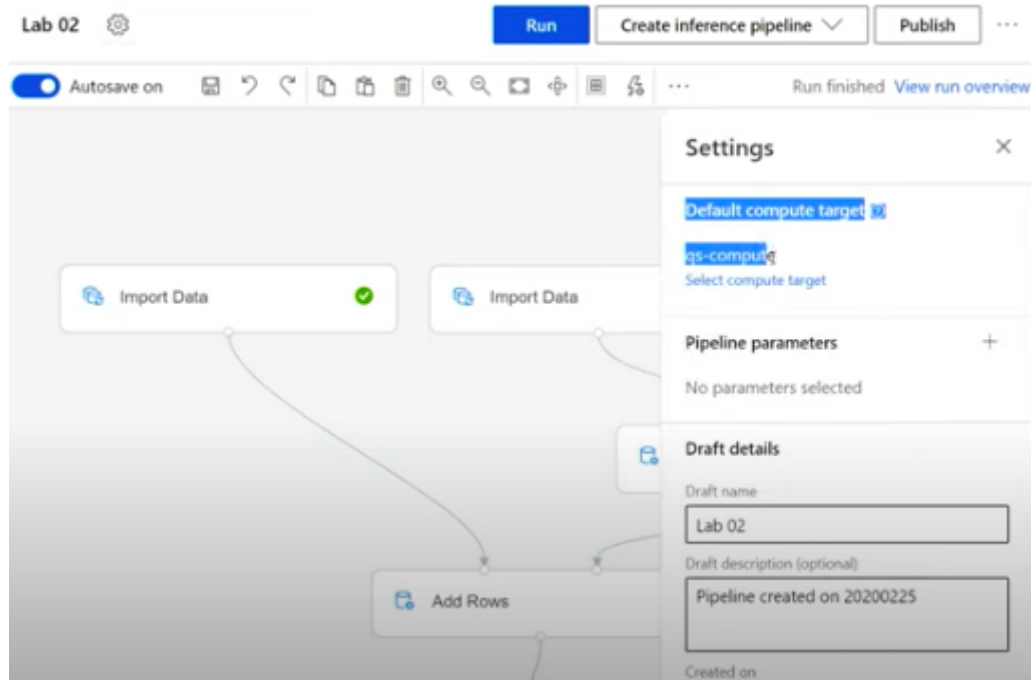
The following exercise focuses on the Import Data module to load data into a machine learning pipeline from several datasets that will be merged and restructured. We will be using some sample data from the UCI dataset repository to demonstrate how you can perform basic data import transformation steps with the modules available in Azure Machine Learning designer.

Section 5: Walkthrough - Import, Transform and Export Data

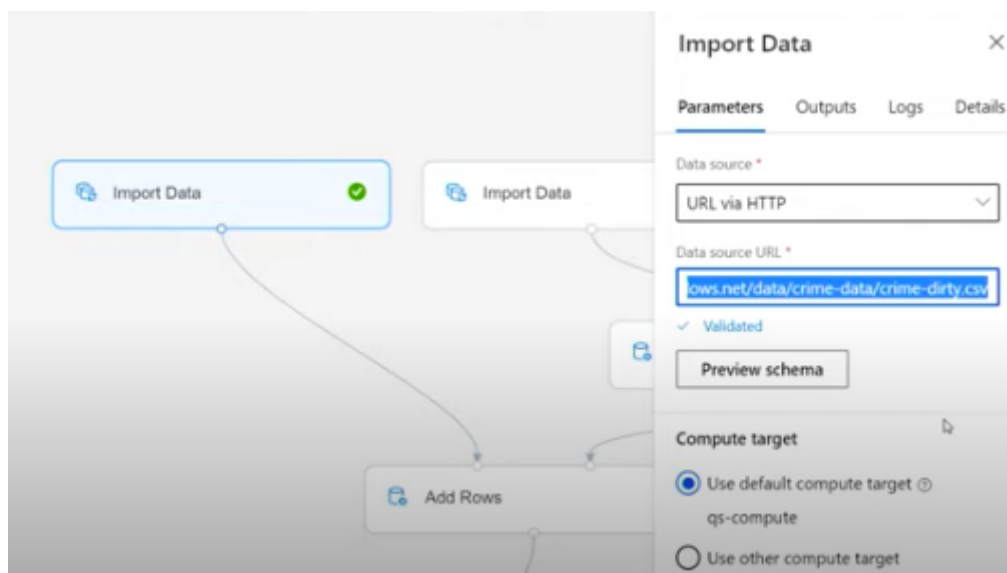
- A new pipeline was created via the Designer.



- The first thing required was to set the default compute target for that particular pipeline.



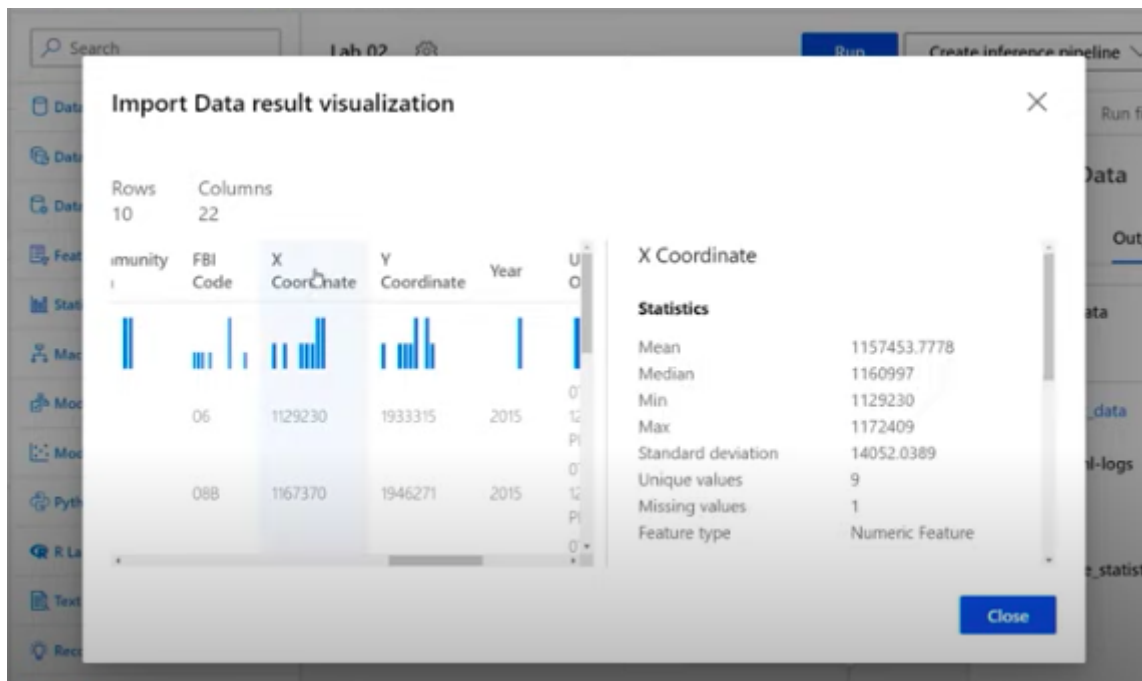
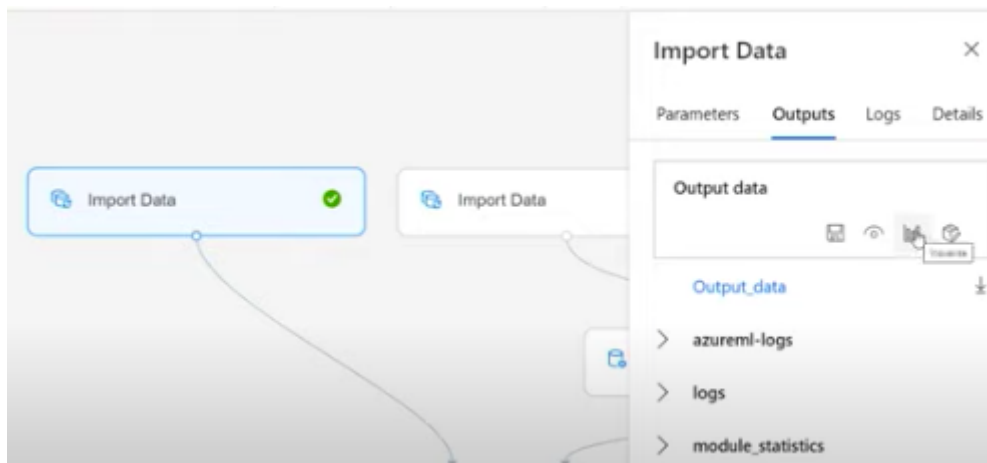
- The next step was to import the data that will be used in this particular experiment. The dataset was selected by using in the Import Data module's parameters the "URL via HTTP" option (so you can provide the URL for the dataset to import).



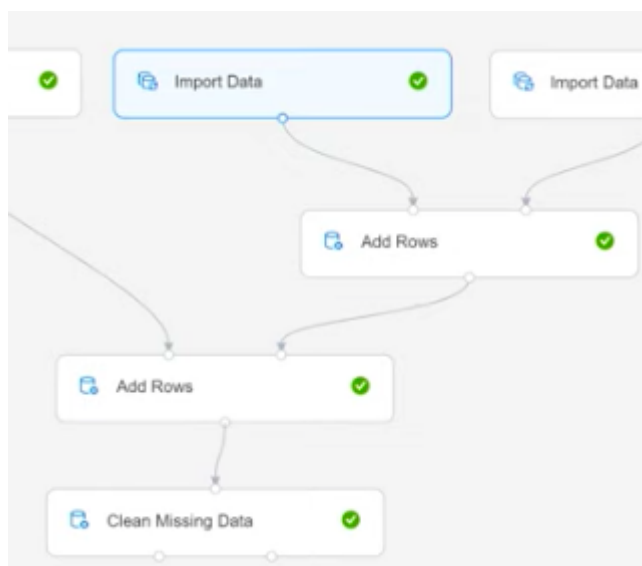
- When the experiment has successfully run you can see the green tick icon associated to the steps that have been successfully completed



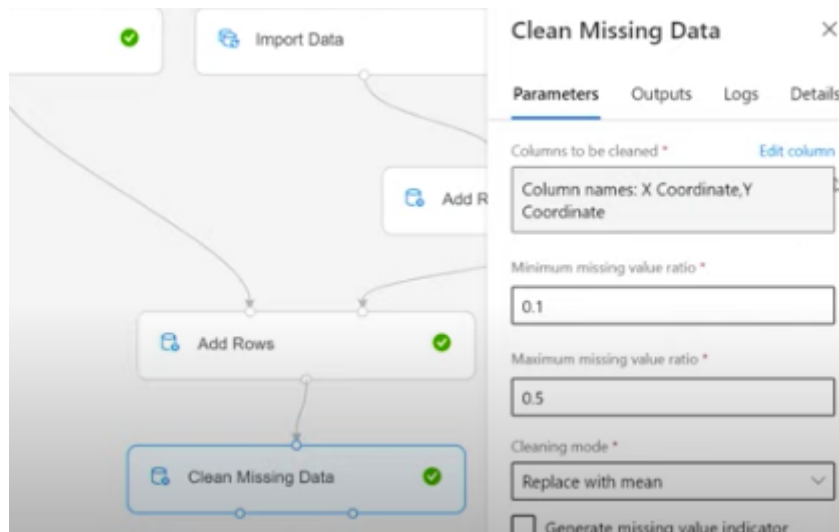
- You can then look at the output generated by each completed step, by selecting 'Visualise'. In this case, details are given for the imported dataset (statistics related to each column)



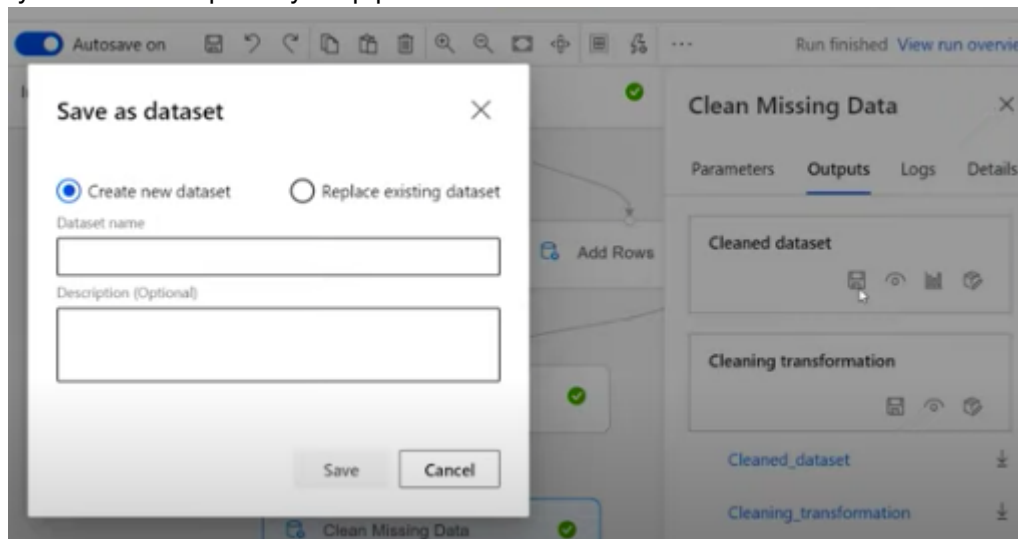
- The second exercise was to use multiple datasets that are imported and restructure them (in this example, merging datasets using **add rows** task and **cleaning missing data**):



- For using the cleaning missing data task you specify the name of the columns you want to be cleaned. You also need to specify the minimum and maximum missing value ratios:



- You can finally select the output of your pipeline and save it as a new dataset:



Section 6: Managing Data

Azure ML has two **data management tools**, Datastores and Datasets. In combination, they offer a secure, scalable and reproducible way to deliver data into most of ML core tasks available on Azure:

- Datastores** are the **abstraction of an Azure storage service**. They store all the information needed to **connect to a particular storage service** in a **secure way**, as the connection information is **hidden** from the entire process (i.e. the **connection information is kept internal**, so it is **not** exposed in scripts). The connection is done using the name of the specific Azure storage service, so no keys or passwords are required. Datastores provide an access mechanism that is independent of the computer resource used to drive a ML process (i.e. **compute location independence**). This means that Datastores can be shared and can be accessed by multiple instances of ML processes.
- Datasets** are resources for **exploring, transforming and managing data** in Azure ML. A dataset is a **reference that points to the data in storage**. It is used to get **specific data files** in the data stores. Datasets are seamlessly integrated with other ML features and thus allow to accomplish a number of ML tasks (e.g. training tasks, real-time inferencing, data labelling, dataset monitoring for data drift). Datasets can be created from: local files, public URLs, Azure Open Datasets, or specific files you upload in the datastores... You can use specific programming languages (e.g. Python) with datasets.

Data management in Azure Machine Learning

How do I securely connect to my data that's in my Azure Storage?

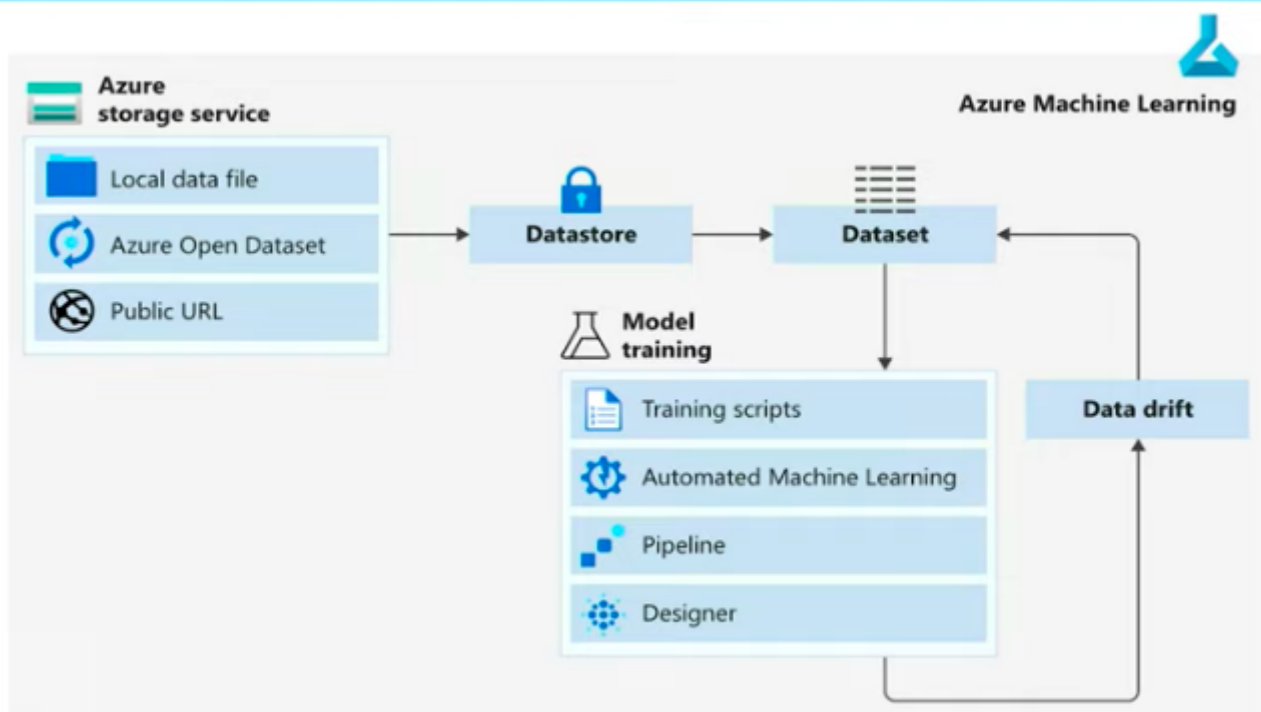
Datastores

How do I get specific data files in my Datastore?

Datasets


The Data Access workflow:

Steps:



1) **Create a datastore**, so that you can access storage services in Azure. A variety of **Datastore types** are supported:

Datastores supported in AML service



- Azure Blob Container
- Azure File Share
- Azure Data Lake
- Azure Data Lake Gen2
- Azure SQL Database
- Azure PostgreSQL
- Databricks File System

Note: Databricks are file systems for Spark environments.

Every Azure ML service will come with a default preconfigured datastore (**default datastore**). The **Azure blob container** and **blob datastore** that get automatically deployed when you create an Azure ML workspace are automatically configured to act as that default datastore. This default configuration can be changed and the type of default datastore then changed (or more datastores can be added).

2) **Create a dataset**, which you will subsequently use for the model training in your ML experiment. You can either **directly mount** your dataset in your experiments **compute target** (when you do for example model training), or you can **consume it directly in Azure ML solutions** (e.g. automated ML processes, experiment runs, ML pipelines, Azure ML designer).

3) **Create a dataset monitor** to detect issues in the data (e.g. **data drift**).

Data Drift --> Over time, the input data you are feeding into your model is likely to change. Data drift can be problematic for model accuracy. Since you trained the model on a certain set of data, it can become increasingly inaccurate if the data changes more and more over time. For example, if you train a model to detect spam in email, it may become less accurate as new types of spam arise that are different from the spam on which the model was trained.

You can set up **dataset monitors** to detect data drift and other issues in your data. When data drift is detected, you can have the **system automatically update the input dataset**, so that you can **re-train the model** and maintain its accuracy.

Section 8: More about Datasets

Important points about Datasets:

- Used to **interact with the data in the Datastore** and to **package data into consumable objects**.
- They are used as **inputs** and **outputs** for **ML pipelines**.
- They can be created from **local files**, **public URLs**, **Azure Open Datasets**, and **files uploaded to the datastores**. Using one of the first 3 options will **connect the dataset to the default datastore** that was automatically created and a copy of the data will be automatically placed in the datastore
- They are **not** copies of the data but **references that point to the original data**. This means that **no extra storage cost** is incurred when you create a new dataset.
- Once a dataset is **registered in Azure ML workspace**, you can **share it and reuse it** across various other experiments without data ingestion complexities.
- You **don't need** to have an Azure storage service to create a dataset, as you can make it directly from a variety of other sources (including your own local files).

Datasets allow you to:

- Have a **single copy of some data** in your storage, but **reference it multiple times**, so that you don't need to create multiple copies each time you need that data available.
- **Access data** during model training **without specifying connection strings or data paths**.
- **More easily share data and collaborate** with other users.
- Bookmark the state of your data by using **dataset versioning**, providing support for **full traceability**.

Dataset versioning is usually necessary when:

- **New data is available** for **retraining**.
- When you are **applying different approaches** to data preparation or feature engineering.

Two **Dataset types** are supported in Azure ML Workspace

- **Tabular dataset** --> represents data in a tabular format created by parsing the provided file or list of files.
- **Web URL (File Dataset)** --> a set of references to single or multiple files in datastores or from public URLs.

Section 9 and 10: Lab (Create & Version a Dataset) + Walkthrough

Lab Overview

To access your data in your storage account, Azure Machine Learning offers datastores and datasets. Create an Azure Machine Learning datasets to interact with data in your datastores and package your data into a consumable object for machine learning tasks. Register the dataset to your workspace to share and reuse it across different experiments without data ingestion complexities.

Datasets can be created from local files, public urls, Azure Open Datasets, or specific file(s) in your datastores. To create a dataset from an in memory pandas dataframe, write the data to a local file, like a csv, and create your dataset from that file. Datasets aren't copies of your data, but are references that point to the data in your storage service, so no extra storage cost is incurred.

In this lab, we are using a subset of NYC Taxi & Limousine Commission - green taxi trip records available from Azure Open Datasets to show how you can register and version a Dataset using the AML designer interface. In the first exercises we use a modified version of the original CSV file, which includes collected records for five months (January till May). The second exercise demonstrates how we can create a new version of the initial dataset when new data is collected (in this case, we included records collected in June in the CSV file).

Keep in mind that datasets and different dataset versions are just references so it is important that the original data files are not changed or overwritten in any way because that will invalidate the datasets and dataset versions.

Steps:

1) Upload the dataset from a local file, to create the first version of the dataset:

nyc-taxi-sample-dataset Version 1

Details Consume Explore Models

Refresh Generate profile Unregister New version

Attributes

Properties
Tabular

Description
--

Created by
ODL_User 152939

Datstore
workspaceblobstore

Relative path
UI/02-25-2020_022636.UTC/nyc-taxi-sample-data-5months.csv

Profile
No profile generated

Files in dataset
1

Tags
No data

2) Explore the uploaded dataset (all columns selected in version 1):

nyc-taxi-sample-dataset Version 1

Details Consume **Explore** Models

Refresh Generate profile Unregister New version

Profile: This is the quick profile generated by sampled data. Please generate a profile from the action bar to view the full profile.

Preview Profile

Number of columns: 14 Number of rows: 50 (of 9776)

normalizeHoli...	isPaidTimeOff	snowDepth	precipTime	precipDepth	temperature	totalAmount
None	false	29.05882353	24	3	6.185714286	44.3
None	false	0	6	0	4.571929825	44.8
None	false	0	1	0	4.384090909	18.96
None	false	29.05882353	24	3	6.185714286	16.3
None	false	0	1	0	3.846428571	5.3

3) A new (slightly modified) version of the same dataset was uploaded as version 2:

quick-starts-ws-152939 > Datasets > nyc-taxi-sample-dataset

nyc-taxi-sample-dataset Version 1

Details Consume **Explore** Models

Refresh Generate profile Unregister New version

Profile: This is the quick profile generated by sampled data. Please generate a profile from the action bar to view the full profile.

Preview Profile

4) Three of the columns present in version 1 have been removed from version 2 (notice columns now are 11 rather than 14):

nyc-taxi-sample-dataset

Version 2 (latest)

Details

Consume

Explore

Models

Refresh

Generate profile

Unregister

New version

Profile: This is the quick profile generated by sampled data. Please generate a profile from the action bar to view the full profile.

Preview

Profile

Number of columns: 11

Number of rows: 50 (of 10000)

tripDistance	hour_of_day	day_of_week	day_of_month	month_num	normalizeHoli...	isPaidTimeOff	
	15	2	27	1	None	false	6.1
	13	4	15	1	None	false	4.5
	23	4	8	1	None	false	4.3
	18	2	27	1	None	false	6.1
	17	6	3	1	None	false	3.8
	9	1	12	1	None	false	0.1
	23	4	22	1	None	false	-2.
	12	4	8	1	None	false	4.3
	0	1	19	1	None	false	-5.

Section 11: Introducing Features

In the previous lesson, we took a look at some examples of tabular data:

Introducing Features

Features
(also known as fields or variables)

SKU	Maker	Color	Quantity	Price
908721	Guess	Blue	789	45.33
456552	Tillys	Red	244	22.91
789921	A&F	Green	387	25.92
872266	Guess	Blue	154	17.56

Instances
(also known as records, observations or cases)

An instance (the whole row)

A numeric feature (the whole column)

We have been referring to this as a data **table**, but you will also see data in this format called a **matrix**. The term matrix is commonly used in mathematics, and it refers to a rectangular array—which, just like a table, contains data arranged in rows and columns. The **rows** in the table are sometimes called **cases** or **instances**. The **columns** in a table can be referred to as **features** and they represent the properties of the cases. In the above example, color and quantity are features of the products.

In the last lesson, we mentioned briefly that **feature engineering** is an important part of data preparation. In many cases, the set of initial features in the data is not enough to produce high quality trained machine learning models. You can use feature engineering to **derive new features based on the values of existing features**. This process can be as simple as applying a mathematical function to a feature (such as adding 1 to

all values in an existing feature) or it can be as complex as training a separate machine learning model to create values for new features. Different techniques are applied depending on the form of the original data (e.g. relational data, text, images, videos, sound, etc...)

Once you have the features, another important task is **selecting the features that are most important or most relevant**. This process is called **feature selection**.

Many machine learning algorithms cannot accommodate a large number of features (the "**curse of dimensionality**"), so it is often necessary to do **dimensionality reduction** to **decrease the number of features**.

Section 12: Feature Engineering

Feature engineering is one of the core techniques that you can use to increase the chances of success when you attempt to solve various ML problems. Its main purpose is to **increase the power of ML algorithms**. It achieves this by **using the values of existing features to derive new ones** that might prove more helpful to the model during the training process. Feature engineering is a very important pre-training task as it lays the foundation for good models with good capabilities in terms of prediction and state stability.

However, features engineering is **not** always necessary: there are cases where the existing features are just fine and you can train and get a good model just using those features.

There are various stages where the Features Engineering step might be implemented; for example:

- Right at the **data source** (e.g. when your data source is a relational database or if you can run the feature engineering process directly using Python libraries within a Python environment)
- In a **dedicated environment** (e.g. when you deal with large amounts of streaming data in a streaming environment that supports parallel data processing, e.g. Spark environment)
- During the **model training process** (e.g. new features can be derived as part of the training process itself)

There is an important distinction to be made between the way classical ML and Deep Learning approach feature engineering:

- With **classic ML**, feature engineering is typically an **explicit** kind of process as you will need to explicitly create the new feature that will be used by the models.
- With **Deep Learning**, most of features engineering occurs **naturally within the neural network itself** (features engineering is usually performed during the training process itself). Deep Learning can be used for the implementation of certain Feature Engineering processes (e.g. **embedding**)

Examples of Features Engineering Tasks

- **Aggregation**: simple types of mathematical functions (e.g. count, distinct count, sum, average, mean, median, ...)
- **Part of**: you extract a part of a certain data structure (e.g. part of a date)
- **Binning**: you group your entities into bins and then apply aggregations over those bins (e.g. binning data based on customers' age category)
- **Flagging**: deriving Boolean conditions that are expressed through Boolean values (True/False).
- **Frequency-based**: you calculate the various frequencies of occurrence for certain amounts of data.
- **Embedding**: also known as **Feature Learning**
- **Deriving by Example**: you aim to learn values of new features using examples of existing features

Features Engineering Approaches by Data Type

Some of the mostly used types of data used in ML are numbers, text and images. The Feature Engineering process is applied differently depending on the data type:

- **Numerical data:** usually is presented in tabular format and it is subject to **classical Features Engineering tasks** (e.g. aggregation, part-of, binning, flagging, frequency-based). Also more advanced Feature Engineering tasks might be used (e.g. **embedding** and **deriving-by-example**).
- **Text:** by its nature is not really suitable for numerical processing and first needs to be translated into a numerical format that can be used by the ML algorithms. **Text Embedding** is the process through which sequences of words/natural language are transformed into numerical format that usually appears as numerical vectors. Several approaches are common for text embedding: e.g. text frequency-inverse document frequency (**Tf-idf**) and **word embedding**.
- **Image:** similarly to text, the RGB format (or any other equivalent) are not really suitable for ML algorithms and need to be transformed. One image can be considered as a matrix of tuples of 3 values (R,G,B channels). You can then obtain (N columns * N rows * 3 channels) = N numerical values, which are the numerical representation of your image. This representation can take a variety of forms: e.g. a **simple vector** of length N, or a **complex matrix** with the same dimensions as the original image). In most cases, features engineering is **not** really required before the training process of images, especially when they are fed into neural networks because Features Engineering naturally happens within the hidden layers of the neural network. Specialised cases of neural nets (e.g. **Convolutional Layers** or **Max Pooling Layers**) are dedicated to learn those features and are capable of learning increasing complex patterns within the image data.

Section 14: Feature Selection

Feature selection is the process that aims to **find the features that are most useful for a given model**. This process of **features filtering** lays the foundation of accurate ML models. There are mainly two reasons for feature selection:

- Elimination of **irrelevant, redundant** or **(highly) correlated features**. These features will not provide any additional value to the ML algorithm. So it's better to **remove** these features (choose and keep only one of them) to simplify the situation and improve performance.
- **Reduction of dimensionality** to increase performance. Many ML algorithms suffer from the **curse of dimensionality** (that is, they do not perform well when given a large number of variables/features/dimensions).

We can improve the situation of having too many features through **dimensionality reduction**, a technique considered both a type of feature engineering (as it produces new features able to summarise the original ones) and a type of feature learning. Commonly used techniques (algorithms) for dimensionality reduction are:

- **PCA (Principal Component Analysis):** it is a **linear technique** based on exact mathematical calculation (it is considered more a statistical approach than a purely ML approach)
- **t-SNE (t-Distributed Stochastic Neighboring Entities):** a probabilistic approach to reduce dimensionality; its target number of dimensions is usually **2 or 3**, thus it is commonly used to **visualise multidimensional data** (i.e. in the form of a scatter plot). It is the capability of keeping close points from the multidimensional space close in the two-dimensional space.
- **Feature embedding:** it is based on **training a separate ML model** to encode a large number of features into a smaller number of features. It allows to encode a larger number of features into a smaller number of "super-features."

Azure ML prebuilt modules:

- **Filter-based feature selection:** identify columns in the input dataset that have the **greatest predictive power**

- **Permutation feature importance:** determine the best features to use by computing a set of **feature importance scores** for the dataset

Section 15: Lab (Engineer and Select Features)

This lab demonstrates the feature engineering process for building a regression model using bike rental demand prediction as an example. In machine learning predictions, effective feature engineering will lead to a more accurate model. We will use the Bike Rental UCI dataset as the input raw data for this experiment. This dataset is based on real data from the Capital Bikeshare company, which operates a bike rental network in Washington DC in the United States. The dataset contains 17,379 rows and 17 columns, each row representing the number of bike rentals within a specific hour of a day in the years 2011 or 2012. Weather conditions (such as temperature, humidity, and wind speed) were included in this raw feature set, and the dates were categorized as holiday vs. weekday etc.

The field to predict is cnt which contains a count value ranging from 1 to 977, representing the number of bike rentals within a specific hour. Our main goal is to construct effective features in the training data, so we build two models using the same algorithm, but with two different datasets. Using the Split Data module in the visual designer, we split the input data in such a way that the training data contains records for the year 2011, and the testing data, records for 2012. Both datasets have the same raw data at the origin, but we added different additional features to each training set:

Set A = weather + holiday + weekday + weekend features for the predicted day
Set B = number of bikes that were rented in each of the previous 12 hours
We are building two training datasets by combining the feature set as follows:

Training set 1: feature set A only
Training set 2: feature sets A+B
For the model, we are using regression because the number of rentals (the label column) contains continuous real numbers. As the algorithm for the experiment, we will be using the Boosted Decision Tree Regression.

Section 17: Data Drift

Data drift is **change in the input data** for a model. Over time, data drift causes **degradation in the model's performance**, as the input data drifts farther and farther from the data on which the model was trained. Monitoring, identifying and detecting data drift helps in diagnosing model's performance issues and enables you to trigger the re-training process to avoid those issues.

There are multiple **causes** of data drift:

- Most commonly, **changes in the upstream process** (e.g. changes in sensors used to provide input data to your model, for example the units of measurement change)
- **Data quality issues** (e.g. if a sensor breaks)
- **Natural data drift** in the data: data is not static but changes over time (e.g. customer behaviour data can naturally drift in time)
- **Covariate shift/change in the relationships between the features** (e.g. at a given point in time 2 features might be highly correlated but not some time afterwards)

With Azure ML you can use **dataset monitors** that can be set up to provide alerts that will assist you in detecting data drift in the dataset in time. This helps in developing ML models that keep their level of performance in time.

Monitoring for Data Drift

Data drift is one of the main reasons that **model performance degrades over time**. A model training process does not finish once you train your first successful model. Rather, it is just the beginning, because its performance will need to be constantly monitored, including by detecting any data drift.

Fortunately, Azure ML allows you to set up **dataset monitors** that can **alert** you about data drift and even take automatic actions to correct data drift. Data monitors analyse the historical version of your data and enable you to profile new data over time. The **data drift algorithm** provides an **overall measure** in the **change** that occurs within your data.

In Azure ML data drift is **monitored using datasets**. The process of monitoring for data drift involves:

- Specifying a **baseline dataset** – usually the **training dataset**
- Specifying a **target dataset** – usually the **input data for the model** **Comparing these two datasets** over time, to monitor for differences (with a set of complex techniques).

Here are different types of comparisons/scenarios you might want to make when monitoring for data drift:

- **Comparing input data vs. training data.** This is a proxy for model accuracy; that is, an increased difference between the input vs. training data is likely to result in a decrease in model accuracy. This involves monitoring a model's input data for drift from the model's training data.
- **Comparing different samples of time series data.** In this case, you are checking for a difference between one time period and another. For example, a model trained on data collected during one season may perform differently when given data from another time of year. Detecting this seasonal drift in the data will alert you to potential issues with your model's accuracy. You can use this comparison if you are dealing with a time series dataset.
- **Performing analysis on past data.** This is done to better understand the dynamics of the data and potentially make choices to improve the model.

The **data drift results** are produced in the form of **charts** in Azure ML. In the image below, there is an example where data from January 2019 was used as the baseline dataset and all the rest of 2019 data was used as a target.

- On the left there is the **data drift magnitude**, which is given as a Percentage between the baseline and the target dataset over time (0% = identical datasets, 100% = datasets completely separable because of data drift).
- On the right there is **drift contribution by feature**, which helps in understanding which of the particular features have contributed the most to the data drift process.



Section 18: Model Training Basics

Our ultimate goal is to produce a **model we can use to make predictions**. Put another way, we want to be able to give the model a set of input features, X , and have it predict the value of some output feature, y . It is necessary to first establish the problem you are trying to solve (e.g. is it a classification problem? or is it a regression problem?). The framing of the problem will influence both the choice of the algorithm you will choose in the training process as well as the various approaches you can take to get the desired results. An essential prerequisite before training your model is understanding and pre-processing the data, if necessary performing features' engineering and selection.

The next steps into the model training process are various:

- Decide whether you need to scale or encode your data
- Split your data into 3 main sets: training, validation and testing set (the first two are typically used in the training oprocess, while the latter used to validate the outcome of the training process i.e. the trained model).
- Choose an appropriate algorithm for the problem to model.
- Finally, the model training is an iterative process that involves: selecting the hyperparameters, trianing the model and the evaluating the model performance.
- Once you have the trained model, you run it on the test dataset, to test the performance of your model on new data.

Parameters and Hyperparameters

When we train a model, a large part of the process involves **learning the values of the parameters of the model**. For example, earlier we looked at the general form for **linear regression**:

$$y = B_0 + B_1 * x_1 + B_2 * x_2 + B_3 * x_3... + B_n * x_n$$

The coefficients in this equation, $B_0 \dots B_n$, determine the intercept and slope of the regression line. When training a linear regression model, we use the training data to figure out what the value of these parameters should be. Thus, we can say that a major goal of model training is to learn the values of the model parameters.

In contrast, **some model parameters are not learned from the data**. These are called **hyperparameters** and their values are **set before training**. Here are some examples of hyperparameters:

- The **number of layers** in a deep neural network
- The **number of clusters** (such as in a k-means clustering algorithm)
- The **learning rate** of the model

We must choose some values for these hyperparameters, but we do not necessarily know what the best values will be prior to training. Because of this, a common approach is to take a best guess, train the model, and then tune adjust or **tune the hyperparameters** based on the **model's performance**.

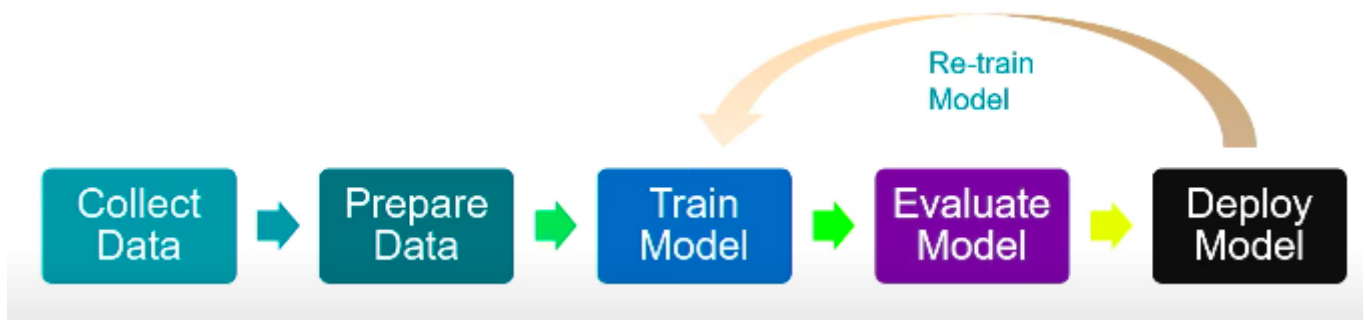
Splitting the Data

We typically want to split our data into three parts:

- **Training data:** to **learn the values for the parameters**.
- **Validation data:** we **check the model's performance** on the validation data and **tune the hyperparameters** until the model performs well with the validation data. For instance, perhaps we need to have more or fewer layers in our neural network. We can adjust this hyperparameter and then test the model on the validation data once again to see if its performance has improved.
- **Test data:** Finally, once we believe we have our finished model (with both parameters and hyperparameters optimized), we will want to do a **final check of its performance**, and we need to do this on some fresh test data that we did not use during the training process.

Section 19: Model Training in Azure ML

Azure ML Service provides the support for the entire data science process. The data science process:

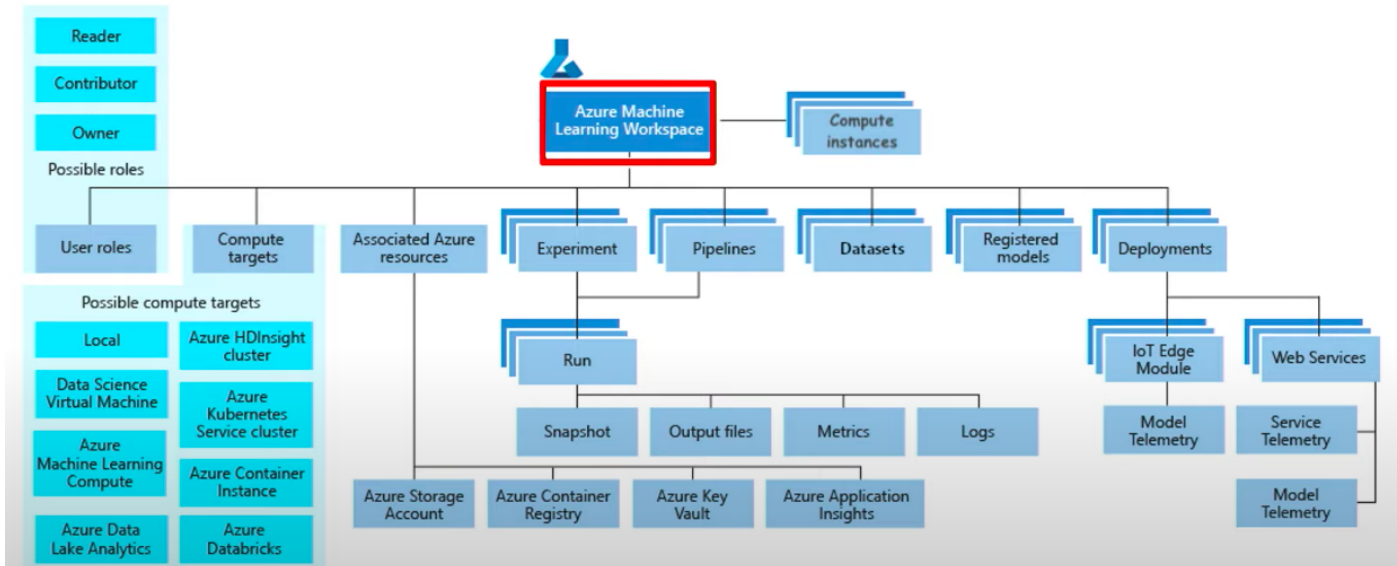


- 1) collecting the data
- 2) data preparation (i.e. data wrangling to get the data in a format suitable for analysis)
- 3) model training (you select a ML algorithm and split your dataset into training/validation/testing sets; the model is continuously evaluated to identify its best performing version; sanity checks on the model's outcomes)
- 4) model evaluation (you see how the model performs on data that has never seen)
- 5) model deployment (typically in an operational environment; you need to package it as well as the dependencies it requires; you can then use it, for example in a web service with an API to integrate it into an application, or for back-end processing that involve batch scoring)
- 6) From time to time, re-assess the quality of your model as the data environment is changing (i.e. model re-training)

Azure ML Service is a **comprehensive environment** to implement all those steps required in the **data science process** and gives you a centralised space to work with all the artifacts involved in the process. It is a **ML managed service** because it provides a platform that simplifies the implementation of the various tasks and offers all the necessary kinds of services that will help creating high quality ML models.

Taxonomy of Azure ML

There are several types of **artifacts** and several classes of **concepts** in Azure ML related to the implementation of the various steps of the data science process:



- Everything in Azure ML revolves around the concept of a **Workspace**, which needs to be created to get started with all the rest of the services.
- You will then typically create a bunch of **Compute instances**, Cloud-based workstations that allow you to have access to development environments such as Jupyter Notebooks. However it is not necessary to use code as there is a comprehensive **designer experience** in Azure ML that will enable to achieve the same objectives without coding.
- Datasets** are key components in the data prep and transformation process, to make data available to ML training processes.
- The typical set of tasks that you run within Azure ML is grouped into the **Experiment** container. An Experiment allows you to group various artifacts related to the ML training processes.
 - One of this artifacts is the **Run** process which is delivered and executed in one of the compute resources available (example of Run processes are models' training/validation, feature engineering for example using Python code,...). Every Run process will output a set of artifacts files (**Snapshot, Output Files, Metrics, Logs**)
- There are **various compute targets** that you can use, so the ML process can run on a large variety of environments. For example **Local environments** (you are training a model on your local machine) or **Remote environments** (e.g native **Azure ML Compute, Azure Data Lake Analytics, Azure HDInsight Cluster, Spark clusters, Azure Kubernetes Service Cluster, ...**)
- Once you have created your model, it can get into the Model Registry as a **Registered Model**, allowing versioning of the model in order to achieve traceability.
- Deployments** are done after the model has been registered. They can be in the form of either **Web Services** or **IoT Edge Modules**. Once your model is in operation, it is important to have capabilities that enable you to collect telemetry data, either about the models themselves (**Model Telemetry**) or about the services exposed using those models (**Service Telemetry**).

Section 20: Training Classifiers

Two of the main types of **supervised learning** are **classification** and **regression**.

In a **classification problem**, the outputs are **categorical or discrete**.

There are **three main types** of classification problems:

- **Binary classification.** Some complex ML tasks perform binary classification (e.g. anomaly or fraud detection)
- **Multi-class single-label classification.** The output consists of multiple classes (e.g. recognition of written numbers, requiring classification of the digits 0-9). This makes the assumption that **each sample is assigned to one and only one label**: a fruit can be either an apple or a pear but not both at the same time.
- **Multi-class multi-label classification.** Multi-label classification **assigns to each sample a set of target labels**. This can be thought of as predicting properties of a data-point that are **not mutually exclusive**, such as topics that are relevant for a document. A text might be about any of religion, politics, finance or education at the same time or none of these.

Examples of classification algorithms:

- **Logistic Regression**
- **Support Vector Machine**

Section 21: Training Regressors

The main distinction that sets a regression problem apart from a classification problem is the form of the output: in a regression problem, the **output is numerical or continuous**.

There are **two types** of regression problems:

- **Regression to arbitrary values.** A classic example would be a problem in which you are given data concerning houses and then asked to predict the price; this is a regression problem because price is a continuous, numerical output.
- **Regression to values between 0 and 1.** This is often related to the problem of binary classification (e.g. give the probability of a fraudulent transaction). A lot of optimisation to certain algorithms that can be applied when the output interval is restricted $[0,1]$

Examples of **regression algorithms** include:

- **Linear Regressor**
- **Decision Forest Regressor**

Section 22: Evaluating Model Performance

It is **not enough** to simply train a model on some data and then assume that the model will subsequently perform well on future data. Instead, as we've mentioned previously, we need to **split off a portion of our labeled data** and reserve it for **evaluating our model's final performance**. We refer to this as the **test dataset**.

If a model learns to perform well with the training data, but performs poorly with the test data, then there may be a problem that we will need to address before putting our model out into the real world.

We will also need to decide what **metrics** we will use to **evaluate performance** of the ML models, and whether there are any **primary metrics** or **particular thresholds** that each model needs to meet on these metrics in order for us to decide that it is "good enough. The metrics used will be different depending on the type of problem you try to solve (e.g. different metrics will be used to measure the performance of a training classifier and a training regressor)

When **splitting the available data**, it is important to **preserve the statistical properties of that data**. This means that the data in the training, validation, and test datasets need to have similar statistical properties as the original data to prevent bias in the trained model. This issue can be addressed using specific stats packages that enable you to sample your data. **Splitting the data up randomly** will help ensure that the two datasets are statistically similar.

Section 23: Confusion Matrices

A **confusion matrix** gets its name from the fact that it is easy to see whether the model is getting confused and misclassifying the data.

A confusion matrix can be represented in a general, abstract form that uses the terms positive and negative:

		Actual class	
		Positive	Negative
Predicted class	Positive	True Positives (TP)	False Positives (FP)
	Negative	False Negatives (FN)	True Negatives (TN)

- **True positives** are the positive cases that are correctly predicted as positive by the model
- **False positives** are the negative cases that are incorrectly predicted as positive by the model
- **True negatives** are the negative cases that are correctly predicted as negative by the model
- **False negatives** are the positive cases that are incorrectly predicted as negative by the model

Section 24: Evaluation Metrics for Classification

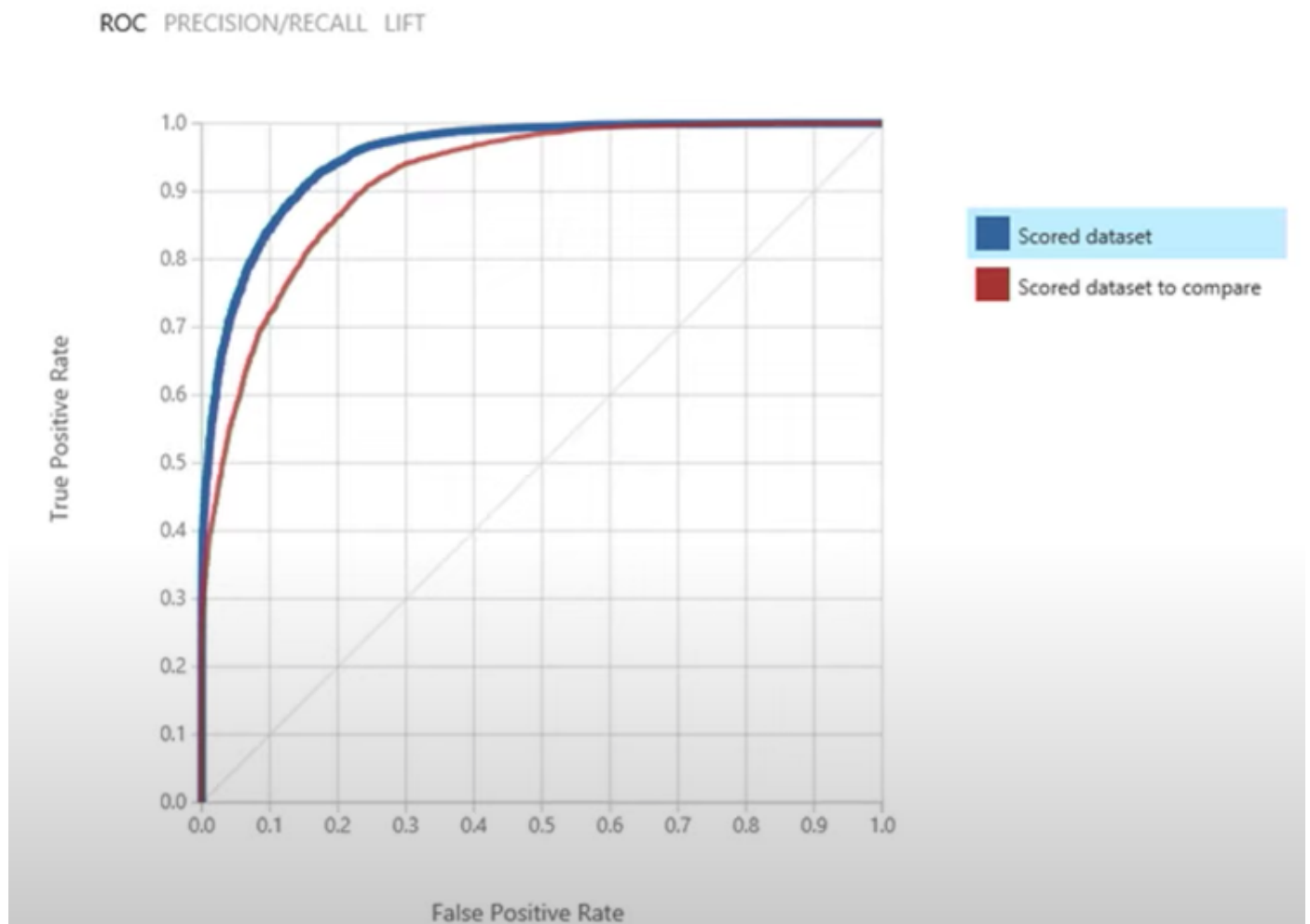
For a binary classifier, the confusion matrix is a 2 by 2 matrix which contains the counts of the observed and predicted output classes. A set of **evaluation metrics** can be inferred from this matrix:

- **Accuracy** = $(TP + TN)/(TP + FP + TN + FN)$. It is the **proportion of correct predictions**.
- **Precision** = $TP/(TP + FP)$. It is the **proportion of the predicted positive cases correctly identified**.
- **Recall** = $TP/(TP + FN)$. It is the **proportion of the actual positive cases correctly identified**.
- **F1 Score** = $2 * (Precision * Recall) / (Precision + Recall)$. It measures the **balance between precision and recall**.

None of these metrics alone is enough to provide an accurate measurement of the metrics for classification. In real life they are always used in pairs (e.g. accuracy and precision, accuracy and recall) to get a complete image of your classification algorithm.

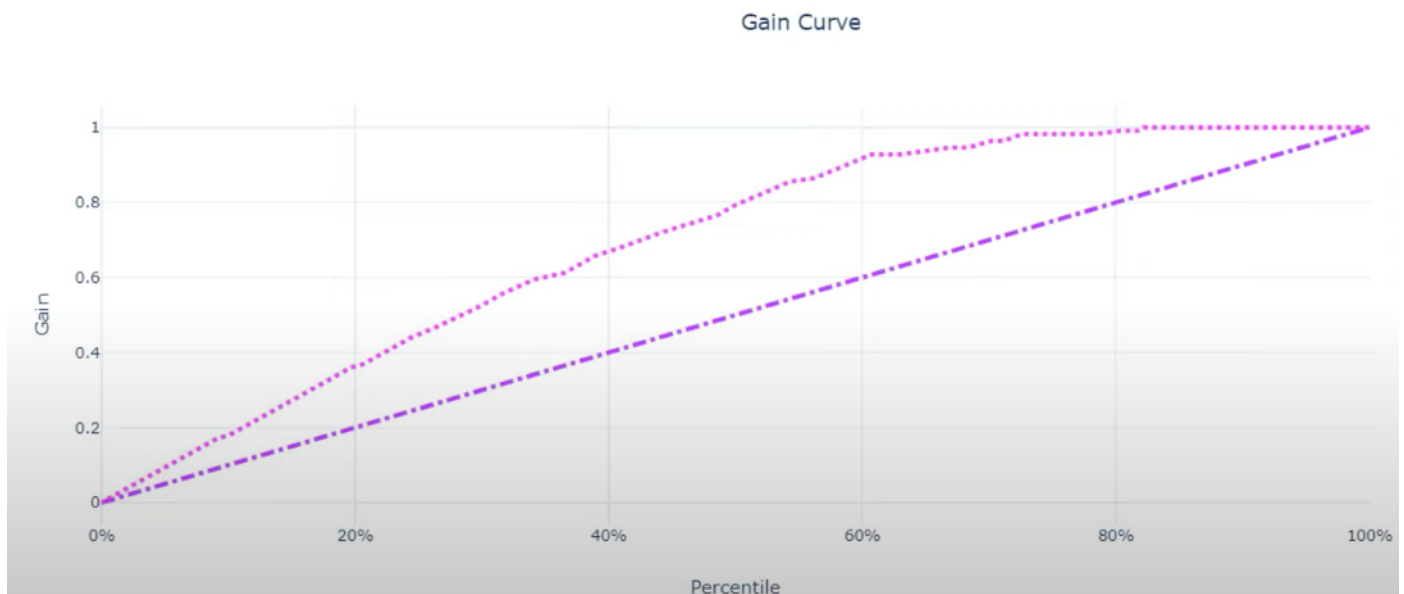
Model Charts for the Evaluation of Classifiers

One of the most important chart used in classification model evaluation is the **ROC (Receiver Operating Characteristics) chart**, a graph of the rate of TP against the rate of FP. Another metric derived from this chart is the **AUC (Area Under the Curve)**, i.e. the area of the surface that lies under the ROC curve. For a random classifier, ROC would be just the main diagonal of the squared area and $AUC = 0.5$, while for an ideal classifier $AUC = 1$. In real life, AUC always falls between 0.5 and 1. Ideally, you would like to have it as close as possible to 1.



Another chart used to evaluate classification models is the **Gain and lift** chart, which deals with **rank ordering the prediction probabilities** and it measures how much better is the classifier's performance in comparison to a random guesser. The diagonal line in the chart corresponds to random guessing. Ideally you want to see the classifier's line as far as possible from the random guessing line.

Gain and Lift charts



Section 26: Evaluation Metrics for Regression

With regression metrics, we are using functions that in some way calculate the **numerical difference between the predicted vs. expected values**. Examples:

- **RMSE (Root Mean Squared Error)**: it measures the squared root of the average of the differences between the predictions and the true values.
- **MAE (Mean Absolute Error)**: it measures the mean of the absolute differences between predictions and true values)
- **R-squared**: commonly known in stats as the **coefficient of determination**, measuring **how close the true values are to the fitted regression line**.
- **Spearman correlation**: it measures the **strength and the direction of the correlation** between predicted and true output values

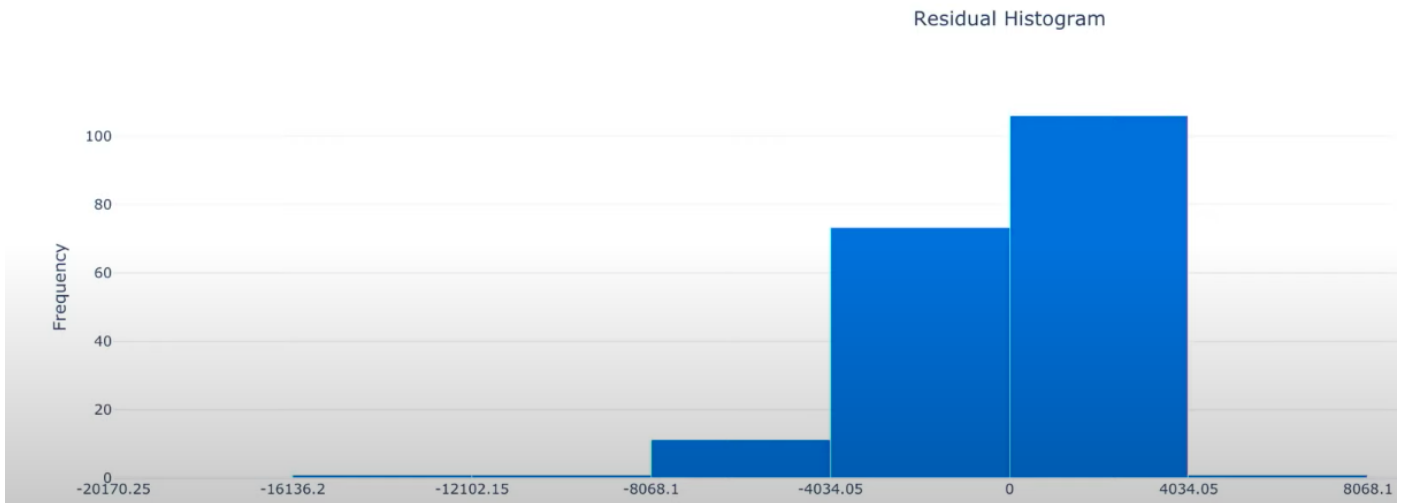
Model Charts for the Evaluation of Regressors

The **Predicted versus True chart** displays the relationship between the predicted values and the true values. An ideal regressor (perfectly accurate) would be represented as a diagonal line. Also, in the bottom part of the chart, you typically see a histogram of how the true values are distributed in the prediction results:



The **Histogram of residuals** represents the distributions of the true values minus the predicted values. When a model has a fairly low bias, this histogram should approach roughly a **normal distribution** (or at least resembling it!):

Histogram of Residuals



Section 27: Lab (Train & Evaluate a Model)

Azure Machine Learning designer (preview) gives you a cloud-based interactive, visual workspace that you can use to easily and quickly prep data, train and deploy machine learning models. It supports Azure Machine Learning compute, GPU or CPU. Machine Learning designer also supports publishing models as web services on Azure Kubernetes Service that can easily be consumed by other applications.

In this lab, we will be using the Flight Delays data set that is enhanced with the weather data. Based on the enriched dataset, we will learn to use the Azure Machine Learning Graphical Interface to process data, build, train, score, and evaluate a classification model to predict if a particular flight will be delayed by 15 minutes or more. To train the model, we will use Azure Machine Learning Compute resource. We will do all of this from the Azure Machine Learning designer without writing a single line of code.

Section 30: Strength in Numbers

No matter how well-trained an individual model is, there is still a significant chance that it could perform poorly or produce incorrect results. Rather than relying on a single model, you can often get better results by **training multiple models** or **using multiple algorithms** and in some way **capturing the collective results**, so to alleviate the potential issues raising from single models.

As we mentioned, there are two main approaches to this: **Ensemble learning** (more theoretical and at the algorithm level; it relies on the inner workings of individual algorithms) and **automated machine learning** (automating as much as possible the process of training individual ML models; thus, the approach is to scale-up the process of training models). Both approaches are using the **principle of the strength in the numbers**: to reduce potential bias of individual ML models, we use the strength of a large number of trained models to improve the accuracy of the prediction.

Let's have a closer look at each of them.

Ensemble Learning

Ensemble learning combines **multiple machine learning models** (trained on the same dataset) to produce **one predictive model**. There are three main types of ensemble algorithms, all relying on the concept that having a large numbers of trained models and combining their output to produce the final one is statistically a better approach to get higher-quality predictions:

- **Bagging or bootstrap aggregation**

- Helps **reduce overfitting for models that tend to have high variance** (such as decision trees)
- Uses **random subsampling of the training data** to produce a **bag of trained models**.
- The **resulting trained models are homogeneous** (equal weights are assigned to the predicted outputs of each individual ML model)
- The final prediction is an **equally-weighted average prediction** from individual models
- **Boosting**
 - Helps **reduce bias** for models.
 - In contrast to bagging, **boosting uses the same input data** to train multiple models using **different hyperparameters**.
 - Boosting **trains model in sequence** by training weak learners one by one, with each new learner correcting errors from previous learners (this approach aims at **producing a strong learner using weaker learners**)
 - The final predictions are a weighted average from the individual models
- **Stacking**
 - Trains a large number of **completely different (heterogeneous) models**
 - Combines the outputs of the individual models into a **meta-model** that yields more accurate predictions

Automated ML (AutoML)

It **automates** many of the **iterative, time-consuming tasks** involved in **model development** (such as **selecting the best features, scaling features optimally, choosing the best algorithms, tuning hyperparameters** and **deciding which are the best metrics** to evaluate the model). The process of automated ML is based on the concept of **strength in variety**. Automated ML allows data scientists, analysts, and developers to build models with greater scale, efficiency, and productivity, all while sustaining model quality. The end result is a **very large number of trained ML models** (order of hundreds or thousands) whose **results** are then **combined** in some way to get better outputs. It helps finding the best performing combination of choices to develop the model to increase the accuracy of the prediction.

With Azure ML is relatively easy to set up an AutoML experiment. You need the input dataset and to allocate a set of compute resources to run the training job. You then need to select the type of task to perform (e.g. classification, regression, forecasting), the primary evaluation metric and define a criteria for finishing the job (at which quality level should the entire job stop?). Based on all these specifications, The AutoML experiment will find the best performing model (often an ensemble model).

AutoML should be considered as a way to get a baseline, i.e. a model with a decent performance but can still be improved.

Section 31: Lab (Train a Two-Class Boosted Decision Tree)

Azure Machine Learning designer gives you a cloud-based interactive, visual workspace that you can use to easily and quickly prep data, train and deploy machine learning models. It supports Azure Machine Learning compute, GPU or CPU. Machine Learning designer also supports publishing models as web services on Azure Kubernetes Service that can easily be consumed by other applications.

In this lab, we will be using the Flight Delays data set that is enhanced with the weather data. Based on the enriched dataset, we will learn to use the Azure Machine Learning Graphical Interface to process data, build, train, score, and evaluate a classification model to predict if a particular flight will be delayed by 15 minutes or more. The classification algorithm used in this lab will be the ensemble algorithm: **Two-Class Boosted Decision Tree**. To train the model, we will use Azure Machine Learning Compute resource. We will do all of this from the Azure Machine Learning designer without writing a single line of code.

Two-Class Boosted Decision Tree: Use this module to create a machine learning model that is based on the boosted decision trees algorithm.

A boosted decision tree is an **ensemble learning method** in which the second tree corrects for the errors of the first tree, the third tree corrects for the errors of the first and second trees, and so forth. Predictions are based on the entire ensemble of trees together that makes the prediction.

Generally, when properly configured, boosted decision trees are the easiest methods with which to get top performance on a wide variety of machine learning tasks. However, they are also one of the more memory-intensive learners, and the current implementation holds everything in memory. Therefore, a boosted decision tree model might not be able to process the large datasets that some linear learners can handle.

Section 34: Lab (Train a Simple Classifier with Automated ML)

Automated machine learning picks an algorithm and hyperparameters for you and generates a model ready for deployment. There are several options that you can use to configure automated machine learning experiments.

Configuration options available in automated machine learning:

Select your experiment type: Classification, Regression or Time Series Forecasting Data source, formats, and fetch data Choose your compute target Automated machine learning experiment settings Run an automated machine learning experiment Explore model metrics Register and deploy model You can create and run automated machine learning experiments in code using the Azure ML Python SDK or if you prefer a no code experience, you can also create your automated machine learning experiments in Azure Machine Learning Studio.

In this lab, you learn how to create, run, and explore automated machine learning experiments in the Azure Machine Learning Studio without a single line of code. As part of this lab, we will be using the Flight Delays data set that is enhanced with the weather data. Based on the enriched dataset, we will use automated machine learning to find the best performing classification model to predict if a particular flight will be delayed by 15 minutes or more. <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-understand-automated-ml> (<https://docs.microsoft.com/en-us/azure/machine-learning/how-to-understand-automated-ml>)