COURSE NOTES: CERTIFIED AI DEVELOPER

What is AI?

Artificial intelligence (AI) is technology that enables computers and machines to simulate human learning, comprehension, problem solving, decision making, creativity and autonomy.

Applications and devices equipped with AI can see and identify objects. They can understand and respond to human language. They can learn from new information and experience. They can make detailed recommendations to users and experts. They can act independently, replacing the need for human intelligence or intervention (a classic example being a self-driving car).

But in 2024, most AI researchers and practitioners—and most AI-related headlines—are focused on breakthroughs in generative AI (gen AI), a technology that can create original text, images, video and other content. To fully understand generative AI, it's important to first understand the technologies on which generative AI tools are built: machine learning (ML) and deep learning.

Machine learning

A simple way to think about AI is as a series of nested or derivative concepts that have emerged over more than 70 years:

Directly underneath AI, we have machine learning, which involves creating models by training an algorithm to make predictions or decisions based on data. It encompasses a broad range of techniques that enable computers to learn from and make inferences based on data without being explicitly programmed for specific tasks.

There are many types of machine learning techniques or algorithms, including linear regression, logistic regression, decision trees, random forest, support vector machines (SVMs), k-nearest neighbor (KNN), clustering and more. Each of these approaches is suited to different kinds of problems and data.

But one of the most popular types of machine learning algorithm is called a neural network (or artificial neural network). Neural networks are modeled after the human brain's structure and function. A neural network consists of interconnected layers of nodes (analogous to neurons) that work together to process and analyze complex data. Neural networks are well suited to tasks that involve identifying complex patterns and relationships in large amounts of data.

The simplest form of machine learning is called supervised learning, which involves the use of labeled data sets to train algorithms to classify data or predict outcomes accurately. In supervised learning, humans pair each training example with an output label. The goal is for

the model to learn the mapping between inputs and outputs in the training data, so it can predict the labels of new, unseen data.

Deep learning

Deep learning is a subset of machine learning that uses multilayered neural networks, called deep neural networks, that more closely simulate the complex decision-making power of the human brain.

Deep neural networks include an input layer, at least three but usually hundreds of hidden layers, and an output layer, unlike neural networks used in classic machine learning models, which usually have only one or two hidden layers.

These multiple layers enable unsupervised learning: they can automate the extraction of features from large, unlabeled and unstructured data sets, and make their own predictions about what the data represents.

Because deep learning doesn't require human intervention, it enables machine learning at a tremendous scale. It is well suited to natural language processing (NLP), computer vision, and other tasks that involve the fast, accurate identification complex patterns and relationships in large amounts of data. Some form of deep learning powers most of the artificial intelligence (AI) applications in our lives today.

Deep learning also enables:

- Semi-supervised learning, which combines supervised and unsupervised learning by using both labeled and unlabeled data to train AI models for classification and regression tasks.
- Self-supervised learning, which generates implicit labels from unstructured data, rather than relying on labeled data sets for supervisory signals.
- Reinforcement learning, which learns by trial-and-error and reward functions rather than by extracting information from hidden patterns.
- Transfer learning, in which knowledge gained through one task or data set is used to improve model performance on another related task or different data set.

Generative AI

Generative AI, sometimes called "gen AI", refers to deep learning models that can create complex original content—such as long-form text, high-quality images, realistic video or audio and more—in response to a user's prompt or request.

At a high level, generative models encode a simplified representation of their training data, and then draw from that representation to create new work that's similar, but not identical, to the original data.

Generative models have been used for years in statistics to analyze numerical data. But over the last decade, they evolved to analyze and generate more complex data types. This evolution coincided with the emergence of three sophisticated deep learning model types:

- Variational autoencoders or VAEs, which were introduced in 2013, and enabled models that could generate multiple variations of content in response to a prompt or instruction.
- Diffusion models, first seen in 2014, which add "noise" to images until they are unrecognizable, and then remove the noise to generate original images in response to prompts.
- Transformers (also called transformer models), which are trained on sequenced data to generate extended sequences of content (such as words in sentences, shapes in an image, frames of a video or commands in software code). Transformers are at the core of most of today's headline-making generative AI tools, including ChatGPT and GPT-4, Copilot, BERT, Bard and Midjourney.

How generative AI works

In general, generative AI operates in three phases:

1. **Training,** to create a foundation model.
2. **Tuning,** to adapt the model to a specific application.
3. **Generation, evaluation and more tuning,** to improve accuracy.

Training

Generative AI begins with a "foundation model"; a deep learning model that serves as the basis for multiple different types of generative AI applications.

The most common foundation models today are large language models (LLMs), created for text generation applications. But there are also foundation models for image, video, sound or music generation, and multimodal foundation models that support several kinds of content.

To create a foundation model, practitioners train a deep learning algorithm on huge volumes of relevant raw, unstructured, unlabeled data, such as terabytes or petabytes of data text or images or video from the internet. The training yields a neural network of billions of *parameters*—encoded representations of the entities, patterns and relationships in the data—that can generate content autonomously in response to prompts. This is the foundation model.

This training process is compute-intensive, time-consuming and expensive. It requires thousands of clustered graphics processing units (GPUs) and weeks of processing, all of

which typically costs millions of dollars. Open source foundation model projects, such as Meta's Llama-2, enable gen AI developers to avoid this step and its costs.

Tuning

Next, the model must be tuned to a specific content generation task. This can be done in various ways, including:

- Fine-tuning, which involves feeding the model application-specific labeled data—questions or prompts the application is likely to receive, and corresponding correct answers in the wanted format.
- Reinforcement learning with human feedback (RLHF), in which human users evaluate the accuracy or relevance of model outputs so that the model can improve itself. This can be as simple as having people type or talk back corrections to a chatbot or virtual assistant.

Generation, evaluation and more tuning

Developers and users regularly assess the outputs of their generative AI apps, and further tune the model—even as often as once a week—for greater accuracy or relevance. In contrast, the foundation model itself is updated much less frequently, perhaps every year or 18 months.

Another option for improving a gen AI app's performance is retrieval augmented generation (RAG), a technique for extending the foundation model to use relevant sources outside of the training data to refine the parameters for greater accuracy or relevance.

Benefits of AI

AI offers numerous benefits across various industries and applications. Some of the most commonly cited benefits include:

- **Automation of repetitive tasks.**
- **More and faster insight from data.**
- **Enhanced decision-making.**
- **Fewer human errors.**
- **24x7 availability.**
- **Reduced physical risks.**

Automation of repetitive tasks

AI can automate routine, repetitive and often tedious tasks—including digital tasks such as data collection, entering and preprocessing, and physical tasks such as warehouse stock-

picking and manufacturing processes. This automation frees to work on higher value, more creative work.

Enhanced decision-making

Whether used for decision support or for fully automated decision-making, AI enables faster, more accurate predictions and reliable, data-driven decisions. Combined with automation, AI enables businesses to act on opportunities and respond to crises as they emerge, in real time and without human intervention.

Fewer human errors

AI can reduce human errors in various ways, from guiding people through the proper steps of a process, to flagging potential errors before they occur, and fully automating processes without human intervention. This is especially important in industries such as healthcare where, for example, AI-guided surgical robotics enable consistent precision.

Machine learning algorithms can continually improve their accuracy and further reduce errors as they're exposed to more data and "learn" from experience.

Round-the-clock availability and consistency

AI is always on, available around the clock, and delivers consistent performance every time. Tools such as AI chatbots or virtual assistants can lighten staffing demands for customer service or support. In other applications—such as materials processing or production lines—AI can help maintain consistent work quality and output levels when used to complete repetitive or tedious tasks.

Reduced physical risk

By automating dangerous work—such as animal control, handling explosives, performing tasks in deep ocean water, high altitudes or in outer space—AI can eliminate the need to put human workers at risk of injury or worse. While they have yet to be perfected, self-driving cars and other vehicles offer the potential to reduce the risk of injury to passengers.

AI use cases

The real-world applications of AI are many. Here is just a small sampling of use cases across various industries to illustrate its potential:

Customer experience, service and support

Companies can implement AI-powered chatbots and virtual assistants to handle customer inquiries, support tickets and more. These tools use natural language processing (NLP) and generative AI capabilities to understand and respond to customer questions about order status, product details and return policies.

Chatbots and virtual assistants enable always-on support, provide faster answers to frequently asked questions (FAQs), free human agents to focus on higher-level tasks, and give customers faster, more consistent service.

Fraud detection

Machine learning and deep learning algorithms can analyze transaction patterns and flag anomalies, such as unusual spending or login locations, that indicate fraudulent transactions. This enables organizations to respond more quickly to potential fraud and limit its impact, giving themselves and customers greater peace of mind.

Personalized marketing

Retailers, banks and other customer-facing companies can use AI to create personalized customer experiences and marketing campaigns that delight customers, improve sales and prevent churn. Based on data from customer purchase history and behaviors, deep learning algorithms can recommend products and services customers are likely to want, and even generate personalized copy and special offers for individual customers in real time.

Human resources and recruitment

AI-driven recruitment platforms can streamline hiring by screening resumes, matching candidates with job descriptions, and even conducting preliminary interviews using video analysis. These and other tools can dramatically reduce the mountain of administrative paperwork associated with fielding a large volume of candidates. It can also reduce response times and time-to-hire, improving the experience for candidates whether they get the job or not.

Application development and modernization

Generative AI code generation tools and automation tools can streamline repetitive coding tasks associated with application development, and accelerate the migration and modernization (reformatting and replatforming) of legacy applications at scale. These tools can speed up tasks, help ensure code consistency and reduce errors.

Predictive maintenance

Machine learning models can analyze data from sensors, Internet of Things (IoT) devices and operational technology (OT) to forecast when maintenance will be required and predict equipment failures before they occur. AI-powered preventive maintenance helps prevent downtime and enables you to stay ahead of supply chain issues before they affect the bottom line.

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
      print("Five is greater than two!")
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example

Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
      print("Five is greater than two!")
```

Python Variables

In Python, variables are created when you assign a value to it:

Example

Variables in Python:

```
x = 5
y = "Hello, World!"
```

Python has no command for declaring a variable.

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

Example

Comments in Python:

```
#This is a comment.
print("Hello, World!")
```

Variables

Variables are containers for storing data values.

Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

```
x = 5
y = "John"
print(x)
print(y)
```

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

Example

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

Get the Type

You can get the data type of a variable with the type() function.

Example

```
x = 5
y = "John"
```

```
print(type(x))
print(type(y))
```

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

Example

```
x = "John"
# is the same as
x = 'John'
```

Case-Sensitive

Variable names are case-sensitive.

Example

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
```

myVar = "John"
MYVAR = "John"
myvar2 = "John"

Example

Illegal variable names:

2myvar = "John"
my-var = "John"
my var = "John"

Remember that variable names are case-sensitive

Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

Camel Case

Each word, except the first, starts with a capital letter:

myVariableName = "John"

Pascal Case

Each word starts with a capital letter:

MyVariableName = "John"

Snake Case

Each word is separated by an underscore character:

my_variable_name = "John"

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
```

```
print(y)
print(z)
```

Note: Make sure the number of variables matches the number of values, or else you will get an error.

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Example

Unpack a list:

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

Output Variables

The Python print() function is often used to output variables.

```
x = "Python is awesome"
print(x)
```

In the print() function, you output multiple variables, separated by a comma:

Example

```
x = "Python"
y = "is"
```

```
z = "awesome"
print(x, y, z)
```

You can also use the + operator to output multiple variables:

Example

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

Notice the space character after "Python " and "is ", without them the result would be "Pythonisawesome".

For numbers, the + character works as a mathematical operator:

Example

```
x = 5
y = 10
print(x + y)
```

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

Example

```
x = 5
y = "John"
print(x + y)
```

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

Example

```
x = 5
y = "John"
print(x, y)
```

Global Variables

Variables that are created outside of a function (as in all of the examples in the previous pages) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Create a variable outside of a function, and use it inside the function

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside a function, with the same name as the global variable

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

Example

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Also, use the global keyword if you want to change a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

Getting the Data Type

You can get the data type of any object by using the type() function:

Print the data type of the variable x:

```
x = 5
print(type(x))
```

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview
x = None	NoneType

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
x = str("Hello World")	str

x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list(("apple", "banana", "cherry"))	list
x = tuple(("apple", "banana", "cherry"))	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

Python Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the type() function:

Example

```
print(type(x))
print(type(y))
print(type(z))
```

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example

Floats:

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
```

```
print(type(y))
print(type(z))
```

Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods:

Example

Convert from one type to another:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
```

```
print(type(b))
print(type(c))
```

Note: You cannot convert complex numbers into another number type.

Random Number

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Example

Import the random module, and display a random number from 1 to 9:

```
import random

print(random.randrange(1, 10))
```

Boolean Values

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

When you run a condition in an if statement, Python returns True or False:

Example

Print a message based on whether the condition is True or False:

```
a = 200
b = 33

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Evaluate Values and Variables

The bool() function allows you to evaluate any value, and give you True or False in return,

Example

Evaluate a string and a number:

```
print(bool("Hello"))
print(bool(15))
```

Example

Evaluate two variables:

```
x = "Hello"
y = 15

print(bool(x))
print(bool(y))
```

Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

Example

The following will return True:

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

Some Values are False

In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

Example

The following will return False:

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a __len__ function that returns 0 or False:

Example

```
class myclass():
  def __len__(self):
    return 0

myobj = myclass()
print(bool(myobj))
```

Functions can Return a Boolean

You can create functions that returns a Boolean Value:

Example

Print the answer of a function:

```
def myFunction() :
  return True

print(myFunction())
```

You can execute code based on the Boolean answer of a function:

Example

Print "YES!" if the function returns True, otherwise print "NO!":

```
def myFunction() :
  return True

if myFunction():
  print("YES!")
else:
  print("NO!")
```

Python also has many built-in functions that return a boolean value, like the isinstance() function, which can be used to determine if an object is of a certain data type:

Example

Check if an object is an integer or not:

```
x = 200
print(isinstance(x, int))
```

Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

```
print(10 + 5)
```

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example	Try it
+	Addition	x + y	
-	Subtraction	x - y	
*	Multiplication	x * y	
/	Division	x / y	
%	Modulus	x % y	

** Exponentiation x ** y

// Floor division x // y

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As	Try it
=	x = 5	x = 5	
+=	x += 3	x = x + 3	
-=	x -= 3	x = x - 3	
*=	x *= 3	x = x * 3	
/=	x /= 3	x = x / 3	
%=	x %= 3	x = x % 3	
//=	x //= 3	x = x // 3	
**=	x **= 3	x = x ** 3	
&=	x &= 3	x = x & 3	
|=	x |= 3	x = x | 3	
^=	x ^= 3	x = x ^ 3	
>>=	x >>= 3	x = x >> 3	
<<=	x <<= 3	x = x << 3	
:=	print(x := 3)	x = 3 print(x)	

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example	Try it
==	Equal	x == y	
!=	Not equal	x != y	
>	Greater than	x > y	
<	Less than	x < y	
>=	Greater than or equal to	x >= y	
<=	Less than or equal to	x <= y	

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example	Try it
and	Returns True if both statements are true	x < 5 and x < 10	
or	Returns True if one of the statements is true	x < 5 or x < 4	
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)	

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example	Try it
is	Returns True if both variables are the same object	x is y	
is not	Returns True if both variables are not the same object	x is not y	

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example	Try it
in	Returns True if a sequence with the specified value is present in the object	x in y	
not in	Returns True if a sequence with the specified value is not present in the object		x not in y

---

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example	Try it
&	AND	Sets each bit to 1 if both bits are 1	x & y	
|	OR	Sets each bit to 1 if one of two bits is 1		x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1		x ^ y
~	NOT	Inverts all the bits		~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off		x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off		x >> 2

Operator Precedence

Operator precedence describes the order in which operations are performed.

Example

Parentheses has the highest precedence, meaning that expressions inside parentheses must be evaluated first:

print((6 + 3) - (6 + 3))

Example

Multiplication * has higher precedence than addition +, and therefore multiplications are evaluated before additions:

print(100 + 5 * 3)

The precedence order is described in the table below, starting with the highest precedence at the top:

Operator	Description	Try it
()	Parentheses	
**	Exponentiation	
+x -x ~x	Unary plus, unary minus, and bitwise NOT	
* / // %	Multiplication, division, floor division, and modulus	
+ -	Addition and subtraction	
<< >>	Bitwise left and right shifts	
&	Bitwise AND	
^	Bitwise XOR	
|	Bitwise OR	
== != > >= < <= is is not in not in	Comparisons, identity, and membership operators	
not	Logical NOT	
and	AND	
or	OR	

If two operators have the same precedence, the expression is evaluated from left to right.

Example

Addition + and subtraction - has the same precedence, and therefore we evaluate the expression from left to right:

print(5 + 4 - 7 + 3)

Python Arrays

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

Arrays

Note: This page shows you how to use LISTS as ARRAYS, however, to work with arrays in Python you will have to import a library, like the NumPy library.

Arrays are used to store multiple values in one single variable:

Create an array containing car names:

cars = ["Ford", "Volvo", "BMW"]

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Access the Elements of an Array

You refer to an array element by referring to the *index number*.

Example

Get the value of the first array item:

x = cars[0]

Example

Modify the value of the first array item:

cars[0] = "Toyota"

The Length of an Array

Use the len() method to return the length of an array (the number of elements in an array).

Example

Return the number of elements in the cars array:

```
x = len(cars)
```

Note: The length of an array is always one more than the highest array index.

Looping Array Elements

You can use the for in loop to loop through all the elements of an array.

Example

Print each item in the cars array:

```
for x in cars:
  print(x)
```

Adding Array Elements

You can use the append() method to add an element to an array.

Example

Add one more element to the cars array:

```
cars.append("Honda")
```

Removing Array Elements

You can use the pop() method to remove an element from the array.

Example

Delete the second element of the cars array:

```
cars.pop(1)
```

You can also use the remove() method to remove an element from the array.

Example

Delete the element that has the value "Volvo":

```
cars.remove("Volvo")
```

Note: The list's remove() method only removes the first occurrence of the specified value.

Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

ExampleGet your own Python Server

Save this code in a file named mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

Note: When using a function from a module, use the syntax: *module_name.function_name*.

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example

Save this code in the file mymodule.py

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Naming a Module

You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module

You can create an alias when you import a module, by using the as keyword:

Example

Create an alias for mymodule called mx:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

Example

Import and use the platform module:

```
import platform

x = platform.system()
print(x)
```

Using the dir() Function

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

Example

List all the defined names belonging to the platform module:

```
import platform

x = dir(platform)
print(x)
```

Note: The dir() function can be used on *all* modules, also the ones you create yourself.

Import From Module

You can choose to import only parts from a module, by using the from keyword.

Example

The module named mymodule has one function and one dictionary:

```
def greeting(name):
  print("Hello, " + name)
```

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import only the person1 dictionary from the module:

```
from mymodule import person1

print (person1["age"])
```

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

File Handling

The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Open a File on the Server

Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

```
f = open("demofile.txt", "r")
print(f.read())
```

If the file is located in a different location, you will have to specify the file path, like this:

Example

Open a file on a different location:

```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Example

Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Read Lines

You can return one line by using the readline() method:

Example

Read one line of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
```

By calling readline() two times, you can read the two first lines:

Example

Read two lines of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

By looping through the lines of the file, you can read the whole file, line by line:

Example

Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Close Files

It is a good practice to always close the file when you are done with it.

Example

Close the file when you are finished with it:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Example

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the overwriting:
f = open("demofile3.txt", "r")
print(f.read())
```

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exists

"a" - Append - will create a file if the specified file does not exists

"w" - Write - will create a file if the specified file does not exists

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

What is a machine learning Model?

A machine learning model is a program that can find patterns or make decisions from a previously unseen dataset. For example, in natural language processing, machine learning models can parse and correctly recognize the intent behind previously unheard sentences or combinations of words. In image recognition, a machine learning model can be taught to recognize objects - such as cars or dogs. A machine learning model can perform such tasks by having it 'trained' with a large dataset. During training, the machine learning algorithm is optimized to find certain patterns or outputs from the dataset, depending on the task. The output of this process - often a computer program with specific rules and data structures - is called a machine learning model.

What is a machine learning Algorithm?

A machine learning algorithm is a mathematical method to find patterns in a set of data. Machine Learning algorithms are often drawn from statistics, calculus, and linear algebra. Some popular examples of machine learning algorithms include linear regression, decision trees, random forest, and XGBoost.

What is Model Training in machine learning?

The process of running a machine learning algorithm on a dataset (called training data) and optimizing the algorithm to find certain patterns or outputs is called model training. The resulting function with rules and data structures is called the trained machine learning model.

What are the different types of Machine Learning?

In general, most machine learning techniques can be classified into supervised learning, unsupervised learning, and reinforcement learning.

What is Supervised Machine Learning?

In supervised machine learning, the algorithm is provided an input dataset, and is rewarded or optimized to meet a set of specific outputs. For example, supervised machine learning is widely deployed in image recognition, utilizing a technique called classification. Supervised machine learning is also used in predicting demographics such as population growth or health metrics, utilizing a technique called regression.

What is Unsupervised Machine Learning?

In unsupervised machine learning, the algorithm is provided an input dataset, but not rewarded or optimized to specific outputs, and instead trained to group objects by common characteristics. For example, recommendation engines on online stores rely on unsupervised machine learning, specifically a technique called clustering.

What is Reinforcement Learning?

In reinforcement learning, the algorithm is made to train itself using many trial and error experiments. Reinforcement learning happens when the algorithm interacts continually with the environment, rather than relying on training data. One of the most popular examples of reinforcement learning is autonomous driving.

What are the different machine learning models?

There are many machine learning models, and almost all of them are based on certain machine learning algorithms. Popular classification and regression algorithms fall under supervised machine learning, and clustering algorithms are generally deployed in unsupervised machine learning scenarios.

Supervised Machine Learning

- Logistic Regression: Logistic Regression is used to determine if an input belongs to a certain group or not
- SVM: SVM, or Support Vector Machines create coordinates for each object in an n-dimensional space and uses a hyperplane to group objects by common features
- Naive Bayes: Naive Bayes is an algorithm that assumes independence among variables and uses probability to classify objects based on features
- Decision Trees: Decision trees are also classifiers that are used to determine what category an input falls into by traversing the leaf's and nodes of a tree
- Linear Regression: Linear regression is used to identify relationships between the variable of interest and the inputs, and predict its values based on the values of the input variables.
- kNN: The k Nearest Neighbors technique involves grouping the closest objects in a dataset and finding the most frequent or average characteristics among the objects.
- Random Forest: Random forest is a collection of many decision trees from random subsets of the data, resulting in a combination of trees that may be more accurate in prediction than a single decision tree.
- Boosting algorithms: Boosting algorithms, such as Gradient Boosting Machine, XGBoost, and LightGBM, use ensemble learning. They combine the predictions from multiple algorithms (such as decision trees) while taking into account the error from the previous algorithm.

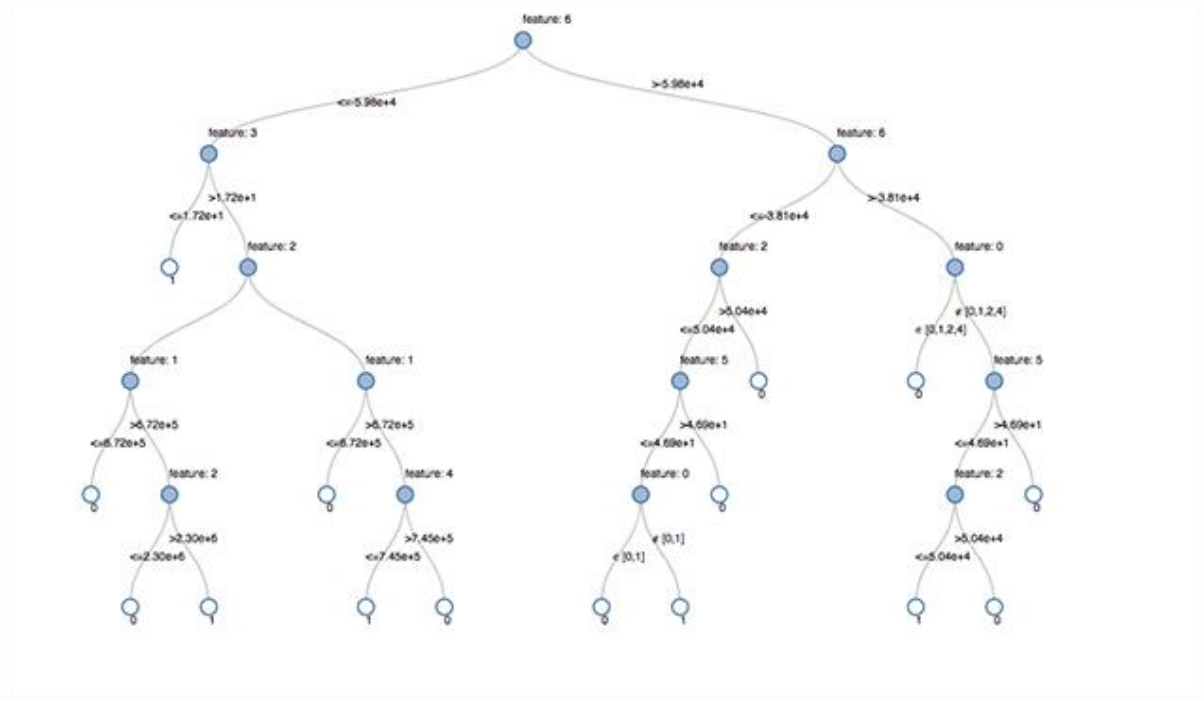
Unsupervised Machine Learning

- K-Means: The K-Means algorithm finds similarities between objects and groups them into K different clusters.

- Hierarchical Clustering: Hierarchical clustering builds a tree of nested clusters without having to specify the number of clusters.

What is a Decision Tree in Machine Learning (ML)?

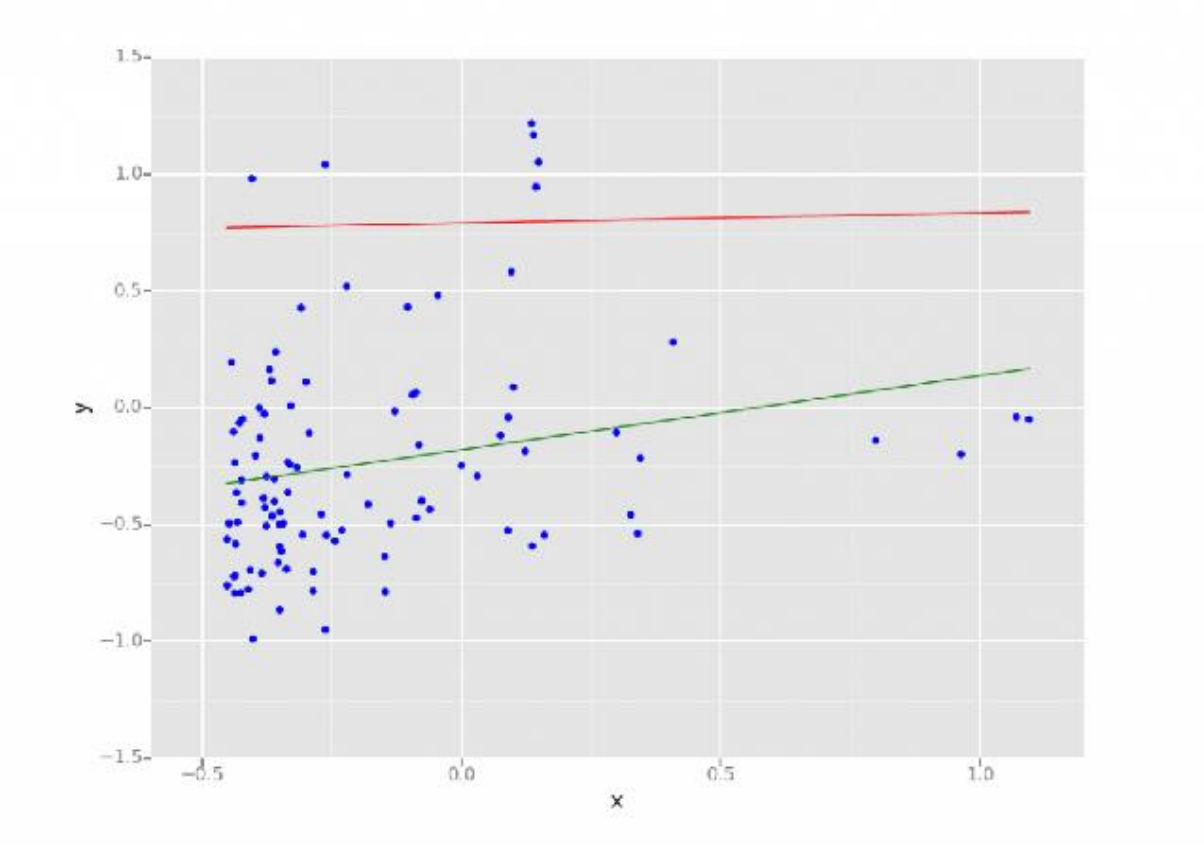
A Decision Tree is a predictive approach in ML to determine what class an object belongs to. As the name suggests, a decision tree is a tree-like flow chart where the class of an object is determined step-by-step using certain known conditions.



A decision tree visualized in the Databricks Lakehouse. Source: https://www.databricks.com/blog/2019/05/02/detecting-financial-fraud-at-scale-with-decision-trees-and-mlflow-on-databricks.html

What is Regression in Machine Learning?

Regression in data science and machine learning is a statistical method that enables predicting outcomes based on a set of input variables. The outcome is often a variable that depends on a combination of the input variables.



A linear regression model performed on the Databricks Lakehouse. Source: https://www.databricks.com/blog/2015/06/04/simplify-machine-learning-on-spark-with-databricks.html

What is a Classifier in Machine Learning?

A classifier is a machine learning algorithm that assigns an object as a member of a category or group. For example, classifiers are used to detect if an email is spam, or if a transaction is fraudulent.

How many models are there in machine learning?

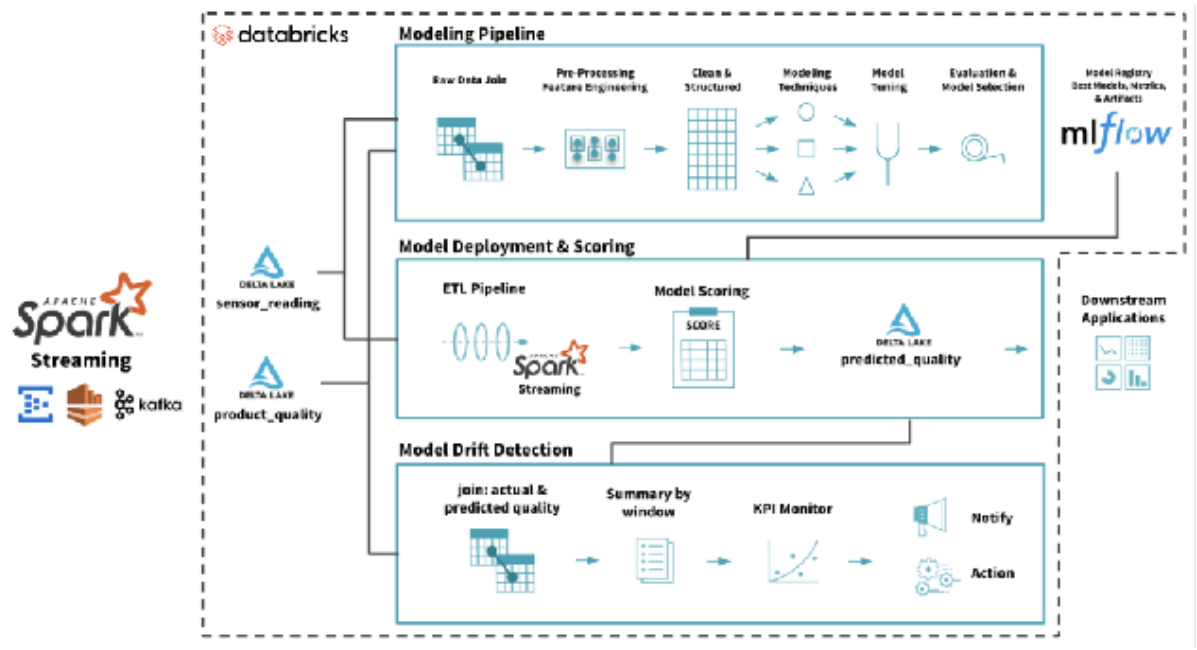
Many! Machine learning is an evolving field and there are always more machine learning models being developed.

What is the best model for machine learning?

The machine learning model most suited for a specific situation depends on the desired outcome. For example, to predict the number of vehicle purchases in a city from historical data, a supervised learning technique such as linear regression might be most useful. On the other hand, to identify if a potential customer in that city would purchase a vehicle, given their income and commuting history, a decision tree might work best.

What is model deployment in Machine Learning (ML)?

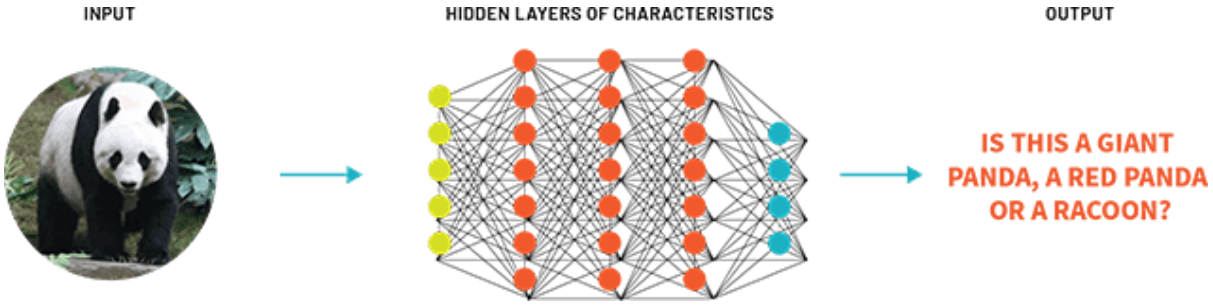
Model deployment is the process of making a machine learning model available for use on a target environment—for testing or production. The model is usually integrated with other applications in the environment (such as databases and UI) through APIs. Deployment is the stage after which an organization can actually make a return on the heavy investment made in model development.



A full machine learning model lifecycle on the Databricks Lakehouse. Source: https://www.databricks.com/blog/2019/09/18/productionizing-machine-learning-from-deployment-to-drift-detection.html

What are Deep Learning Models?

Deep learning models are a class of ML models that imitate the way humans process information. The model consists of several layers of processing (hence the term 'deep') to extract high-level features from the data provided. Each processing layer passes on a more abstract representation of the data to the next layer, with the final layer providing a more human-like insight. Unlike traditional ML models which require data to be labeled, deep learning models can ingest large amounts of unstructured data. They are used to perform more human-like functions such as facial recognition and natural language processing.



A simplified representation of deep learning. Source: https://www.databricks.com/discover/pages/the-democratization-of-artificial-intelligence-and-deep-learning

What is Time Series Machine Learning?

A time-series machine learning model is one in which one of the independent variables is a successive length of time minutes, days, years etc.), and has a bearing on the dependent or predicted variable. Time series machine learning models are used to predict time-bound events, for example - the weather in a future week, expected number of customers in a future month, revenue guidance for a future year, and so on.