Introduction to Natural Language Processing with Hugging Face Transformers

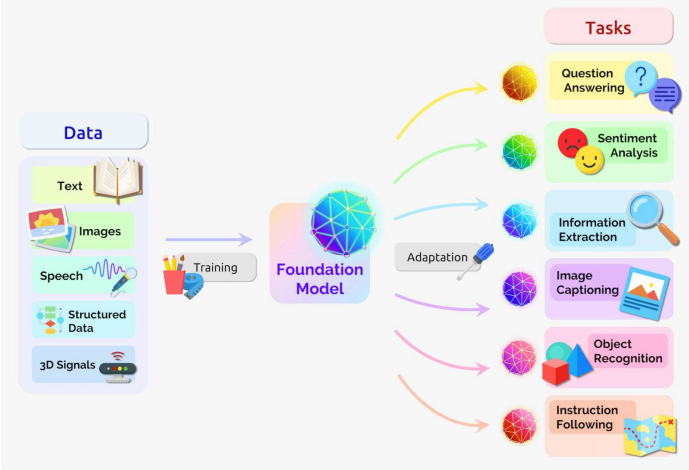


Image Source: nvidia

This Guided Project will walk you through some of the applications of Hugging Face Transformers in Natural Language Processing (NLP).

Hugging Face Transformers package is very popular and versatile Python library that provides pre-trained models for a variety of applications in NLP, as well other areas such as image analysis, audio analysis, multimodal analysis (optical character recognition, video classification, visual question answering and many more).

This Guided Project will focus on text analysis tasks, which are:

- Text Classification.
  - Sentiment Analysis. Classifies the polarity of a given text.
- Topic Classification. Classifies sequences into specified class names.
- Text Generator. Generates text from a given input.
- Token Classification.
  - Name Entity Recognition (NER). Labels each word with the entity it represents.
- Question answering. Extracts the answer from the context.
- Text Summarization. Generates a summary of a long sequence of text or document.
- Translation. Translates text into another language.

Table of Contents

Objectives

After completing this lab you will be able to:

- Use Hugging Face Transformers to do:
    - Sentiment Analysis
    - Topic Classification
    - Text Generation
    - Name Entity Recognition
    - Question Answering
    - Text Summarization
    - Text Translation

Setup

For this lab, we will be using the following libraries:

- `pandas` for managing the data.

Installing Required Libraries

The following required libraries are pre-installed in the Skills Network Labs environment. However, if you run this notebook commands in a different Jupyter environment (e.g. Watson Studio or Ananconda), you will need to install these libraries by removing the `#` sign before `!mamba` in the code cell below.

In [1]:
```
# All Libraries required for this lab are listed below. The libraries pre-installed
# !mamba install -qy pandas==1.3.4 numpy==1.21.4 seaborn==0.9.0 matplotlib==3.5.0 s
# Note: If your environment doesn't support "!mamba install", use "!pip install"
```

The following required libraries are **not** pre-installed in the Skills Network Labs environment. **You will need to run the following cell** to install them:

In [2]:
```
!pip install torch
```

```
Requirement already satisfied: torch in /home/jupyterlab/conda/envs/python/lib/pytho
n3.7/site-packages (1.13.1+cpu)
Requirement already satisfied: typing-extensions in /home/jupyterlab/conda/envs/pyth
on/lib/python3.7/site-packages (from torch) (4.5.0)
```

In [3]:
```
!pip install --upgrade torch
```

```
Requirement already satisfied: torch in /home/jupyterlab/conda/envs/python/lib/pytho
n3.7/site-packages (1.13.1+cpu)
Requirement already satisfied: typing-extensions in /home/jupyterlab/conda/envs/pyth
on/lib/python3.7/site-packages (from torch) (4.5.0)
```

In [4]:
```
!pip install -q transformers
```

In [5]:
```
!pip install datasets evaluate transformers[sentencepiece]
```

```
Requirement already satisfied: datasets in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (2.13.2)
Requirement already satisfied: evaluate in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (0.4.1)
Requirement already satisfied: transformers[sentencepiece] in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (4.30.2)
Requirement already satisfied: numpy>=1.17 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (1.21.6)
Requirement already satisfied: pyarrow>=8.0.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (12.0.1)
Requirement already satisfied: dill<0.3.7,>=0.3.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (0.3.6)
Requirement already satisfied: pandas in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (1.3.5)
Requirement already satisfied: requests>=2.19.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (2.29.0)
Requirement already satisfied: tqdm>=4.62.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (4.67.1)
Requirement already satisfied: xxhash in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (0.70.14)
Requirement already satisfied: fsspec[http]>=2021.11.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (2023.1.0)
Requirement already satisfied: aiohttp in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (3.8.6)
Requirement already satisfied: huggingface-hub<1.0.0,>=0.11.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (0.16.4)
Requirement already satisfied: packaging in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (6.0)
Requirement already satisfied: importlib-metadata in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from datasets) (4.11.4)
Requirement already satisfied: responses<0.19 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from evaluate) (0.18.0)
Requirement already satisfied: filelock in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from transformers[sentencepiece]) (3.12.2)
Requirement already satisfied: regex!=2019.12.17 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from transformers[sentencepiece]) (2024.4.16)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from transformers[sentencepiece]) (0.13.3)
Requirement already satisfied: safetensors>=0.3.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from transformers[sentencepiece]) (0.4.5)
Requirement already satisfied: sentencepiece!=0.1.92,>=0.1.91 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from transformers[sentencepiece]) (0.2.0)
Requirement already satisfied: protobuf<=3.20.3 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from transformers[sentencepiece]) (3.20.3)
Requirement already satisfied: attrs>=17.3.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (3.1.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (6.0.5)
```

```
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: yarl<2.0,>=1.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: frozenlist>=1.1.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (1.3.3)
Requirement already satisfied: aiosignal>=1.1.2 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: asynctest==0.13.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (0.13.0)
Requirement already satisfied: typing-extensions>=3.7.4 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from aiohttp->datasets) (4.5.0)
Requirement already satisfied: idna<4,>=2.5 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from requests>=2.19.0->datasets) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from requests>=2.19.0->datasets) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from requests>=2.19.0->datasets) (2023.5.7)
Requirement already satisfied: zipp>=0.5 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from importlib-metadata->datasets) (3.15.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from pandas->datasets) (2023.3)
Requirement already satisfied: six>=1.5 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas->datasets) (1.16.0)
```

In [6]:
```
!pip install sacremoses
```

```
Requirement already satisfied: sacremoses in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (0.0.53)
Requirement already satisfied: regex in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from sacremoses) (2024.4.16)
Requirement already satisfied: six in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from sacremoses) (1.16.0)
Requirement already satisfied: click in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from sacremoses) (8.1.3)
Requirement already satisfied: joblib in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from sacremoses) (1.3.2)
Requirement already satisfied: tqdm in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from sacremoses) (4.67.1)
Requirement already satisfied: importlib-metadata in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from click->sacremoses) (4.11.4)
Requirement already satisfied: zipp>=0.5 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from importlib-metadata->click->sacremoses) (3.15.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from importlib-metadata->click->sacremoses) (4.5.0)
```

Please **restart the kernel** after running the above installs.

Importing Required Libraries

We recommend you import all required libraries in one place (here):

In [7]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [8]:
```
from transformers import pipeline
from transformers import AutoTokenizer
from transformers import AutoModel
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/utils/validat
ion.py:37: DeprecationWarning: distutils Version classes are deprecated. Use packagi
ng.version instead.
  LARGE_SPARSE_SUPPORTED = LooseVersion(scipy_version) >= '0.14.0'
```

Background (optional)

What is a Transformer Model?

A Transformer Model is a neural network that learns context and thus meaning by tracking relationships in sequential data, for example, words in a sentence. Transformer models apply an evolving set of mathematical techniques, called attention or self-attention, differently weighing the significance of each part of the input data. They are used primarily in the fields of natural language processing and computer vision. Similarly to recurrent neural networks (RNNs), transformers are designed to process sequential data, for example a body of text, with applications such as translation and text summarization. However, unlike RNNs, transformers process the entire input all at once. The attention mechanism provides context for any position in the input sequence. For example, if the input data is a natural language sentence, the transformer does not have to process one word at a time. This allows for more parallelization than RNNs, and therefore, reduces training times. The additional training parallelization allows training on larger datasets. This led to the development of pre-trained systems such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), which were trained with large language datasets, such as the Wikipedia Corpus and Common Crawl, and can be fine-tuned for specific tasks.

Transformer models were first described in a 2017 paper from Google, as the newest and the most powerful classes of models invented today, replacing some RNNs.

What do Transformer Models do?

Transformer Models have many useful applications in today world. They are widely used in near real-time translation tasks, opening communication spaces to language-diverse and hearing-impaired audiences. These models are advancing medical research by helping scientists to understand the DNA and amino acid sequences to speed up the development and improve the quality of treatments for different diseases. The models can detect anomalies to prevent fraud detection, streamline manufacturing, make recommendations and overall improve our day-to-day quality of life.

Transformer Model Architecture

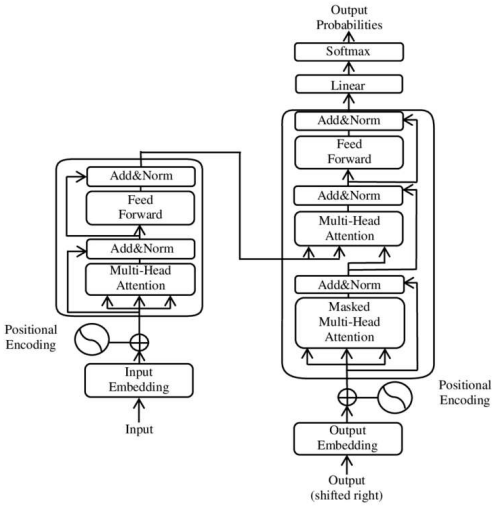


Image Source: wikipedia

The diagram above describes the pipeline between the input, output and positional encoding. For more information on Transformer Architecture visit this wikipedia page.

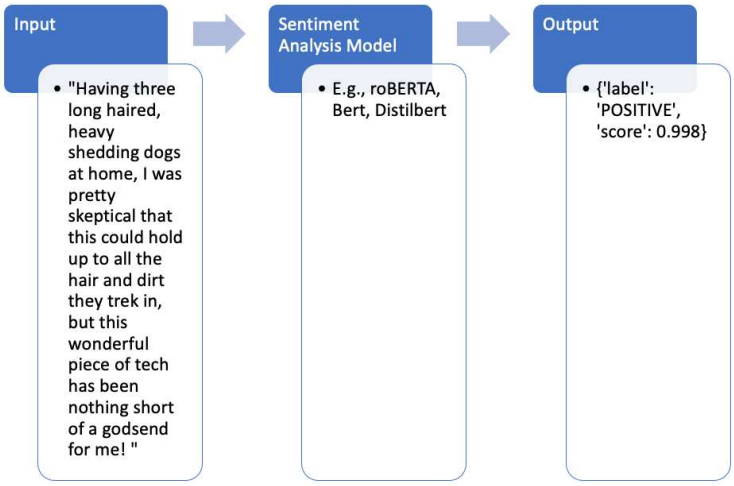
How to use Hugging Face Transformers

There is very easy way to apply Hugging Face Transformers, it is to use the `pipeline()` function. The pipeline makes it simple to use any model from the Hugging Face Hub for inference on any language, computer vision, speech, or multimodal tasks. Each task has an associated pipeline. However, it is simpler to use the general pipeline abstraction, which contains all the task-specific pipelines. The `pipeline()` automatically loads a default model and a pre-processing class capable of inference for any of your desired tasks.

The links beside each of the 7 Examples shown here, are from Hugging Face community and contain more information about the models and data they were trained on. This Hugging Face Page also contains ALL the models created by this community.

Example 1 - Sentiment Analysis

Sentiment analysis is a natural language processing technique that identifies the polarity of a given text. Some of the most common practical uses of sentiment analysis are tweets analysis, product reviews, support tickets classification and others. Sentiment analysis allows quick processing of large amounts, real-time data. The Diagram below, describes the process of Sentiment Analysis. As shown in the Diagram below, we can use sentiment analysis to predict a label and a score of a random amazon product review.



The evaluation metric for this type of analysis is accuracy score. The default model behind "sentiment-analysis" pipeline is "distilbert-base-uncased-finetuned-sst-2-english" model, pre-trained on a large corpus of English data in a self-supervised fashion. This means it was pre-trained on the raw texts only, with no humans labelling them in any way. For more information on this model read here.

We can start by loading text classification pipeline from `pipeline()` using "sentiment-analysis" task identifier. It uses the default, "distilbert-base-uncased-finetuned-sst-2-english" model for sentiment analysis. Note, you can skip specifying a default model, but you will recieve a warning message.

```
In [9]: classifier = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetune
```

```
Xformers is not installed correctly. If you want to use memory_efficient_attention t
o accelerate training use the following command to install Xformers
pip install xformers.
```

We can copy a random product review from amazon and input it into selected classifier. Review Source.

```
In [10]: classifier("Having three long haired, heavy shedding dogs at home, I was pretty ske
```

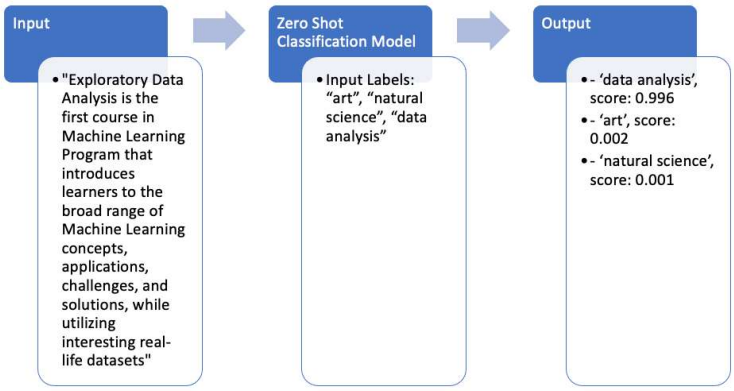
```
Out[10]: [{'label': 'POSITIVE', 'score': 0.9982457160949707}]
```

As we see, the sentiment is classified as Positive with 99.8% accuracy score.

Example 2 - Topic Classification

Topic Classification task classifies sequences into specified class names. It applies "zero-shot-classification" algorithm to perform this task. Zero-Shot Learning (ZSL) is when a classifier learns on one set of labels and then evaluates on a different set of labels, the ones that it has never seen before. This link has more information about ZSL. As shown in the Diagram below, we also need to specify some kind of descriptor (input labels) for an unseen

class (such as a set of visual attributes or simply the class name) in order for our model to be able to predict that class without training data.



First, we load a pipeline with "zero-shot-classification", pass a sequence that we want to classify and a list of candidate labels and see how the model will assign corresponding labels to the input.

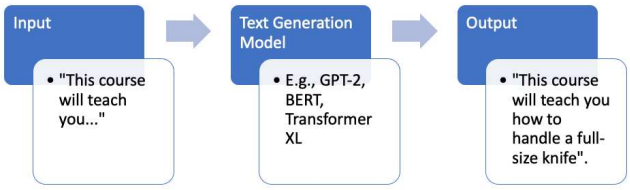
In [11]:
```
classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
classifier(
    "Exploratory Data Analysis is the first course in Machine Learning Program that
    candidate_labels=["art", "natural science", "data analysis"],
)
```

Out[11]:
```
{'sequence': 'Exploratory Data Analysis is the first course in Machine Learning Pr
ogram that introduces learners to the broad range of Machine Learning concepts, ap
plications, challenges, and solutions, while utilizing interesting real-life datas
ets',
 'labels': ['data analysis', 'art', 'natural science'],
 'scores': [0.9957792162895203, 0.0026982580311596394, 0.0015224860981106758]}
```

As we see, 'data analysis' is the most successful candidate for the topic of this input, having 99.6% score.

Example 3 - Text Generator

Text Generation model is also known as causal language model, is a task of predicting a next word in a sentence, given some previous input. This task is very similar to the auto-correct function we have on our phones. Classification metric cannot be used in this task, as there is no single correct answer. Instead, text distribution auto-completed by the model is evaluated by the cross entropy loss and perplexity. The default model behind Text Generation is Generative Pre-trained Transformer 2, GPT-2 model. It can receive an input like "This course will teach you" and proceed to complete the sentence based on those first words, as shown in the Diagram below.



Similarly to Completion Generation Models, we also have Text-to-Text Models. These models are trained to learn the mapping between a pair of texts (e.g. translation from one language to another). The most popular variants of these models are T5, T0 and BART. Text-to-Text models are trained with multi-tasking capabilities, they can accomplish a wide range of tasks, including summarization, translation, and text classification.

Interestingly, causal language models can also be used to generate a code to help programmers in their repetitive coding tasks.

To begin our analysis, we can load a pipeline with the default "text-generation" model:

In [12]:
```
generator = pipeline("text-generation", model="gpt2")
generator("This course will teach you")
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

Out[12]:
```
[{'generated_text': 'This course will teach you how to handle and manage questions
like "Is it possible to build a web application without writing a REST API clien
t," with a focus on REST APIs. You will learn that using client-side services, tha
t is, without writing'}]
```

Alternatively, we can also use "distilgpt2" model, as well as some parameters, such length and number of the sentences needed. Distilled GPT-2 model is an English-language model pre-trained with the supervision of the smallest version of GPT-2. Like GPT-2, DistilGPT2 can be used to generate text. For more information about this model, please visit this link.

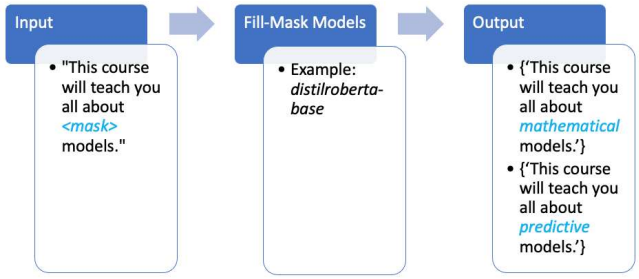
In [13]:
```
generator = pipeline("text-generation", model="distilgpt2")
generator(
    "This course will teach you",
    max_length=30,
    num_return_sequences=2,
)
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

Out[13]:
```
[{'generated_text': 'This course will teach you how to do it best and how to keep
your eyes open:\n\n\n\n\n\n\n\n\n\n\n\n'},
 {'generated_text': 'This course will teach you how to become a filmmaker and what
to do. It will be the very first course that will teach you how to be a'}]
```

Masked Language Modeling

Sometimes, it is useful to use Masked Language modeling, which also has Text Generation capabilities. **Masked language** modeling is the task of masking some of the words in a sentence and predicting which words should replace those masks. These models are useful when we want to get a statistical understanding of the language in which the model is trained in. Masked language models do not require labelled data! They are trained by masking a couple of words in sentences and the model is expected to guess the masked word. The Diagram below shows a simple representation of this concept.



For example, masked language modeling is used to train large models for domain-specific problems. If you have to work on a domain-specific task, such as retrieving information from medical research papers, you can train a masked language model using those papers.

For more information about "fill-mask" model you can read here. The Example below, shows a few options for a 'masked' word in the input sentence. Let's see 4 options in the output.

In [14]:
```
unmasker = pipeline("fill-mask", "distilroberta-base")
unmasker("This course will teach you all about <mask> models.", top_k=4)
```

Out[14]:
```
[{'score': 0.196197971701622,
  'token': 30412,
  'token_str': ' mathematical',
  'sequence': 'This course will teach you all about mathematical models.'},
 {'score': 0.04052729904651642,
  'token': 38163,
  'token_str': ' computational',
  'sequence': 'This course will teach you all about computational models.'},
 {'score': 0.03301800787448883,
  'token': 27930,
  'token_str': ' predictive',
  'sequence': 'This course will teach you all about predictive models.'},
 {'score': 0.03194150701165199,
  'token': 745,
  'token_str': ' building',
  'sequence': 'This course will teach you all about building models.'}]
```

Example 4 - Name Entity Recognition (NER)

NER sometimes also referred as entity chunking, extraction, or identification, is the task of identifying and categorizing key information (entities) in text. The model sorts according to

name of the person: 'PER', group: 'ORG', and location: 'LOC' with appropriate accuracy score and token location.

The default model behind NER is "camembert-ner". It was trained and fine tuned on wikiner-fr dataset (~170 634 sentences). Click here to learn more about NER.

Let's look at the specific Example, where we load a pipeline with "ner" model and some input:

In [15]:
```
ner = pipeline("ner", model="dbmdz/bert-large-cased-finetuned-conll03-english", gro
ner("My name is Roberta and I work with IBM Skills Network in Toronto")
```

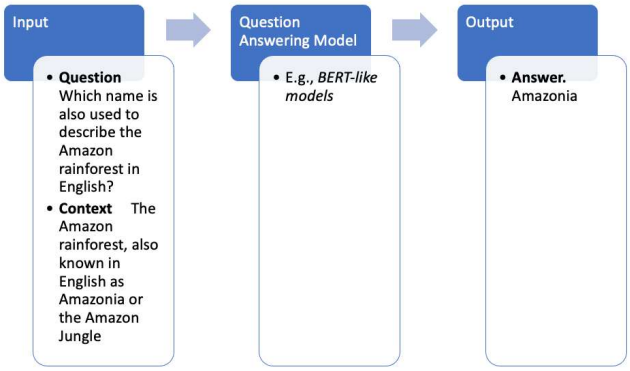
Out[15]:
```
[{'entity_group': 'PER',
  'score': 0.9993105,
  'word': 'Roberta',
  'start': 11,
  'end': 18},
 {'entity_group': 'ORG',
  'score': 0.9976597,
  'word': 'IBM Skills Network',
  'start': 35,
  'end': 53},
 {'entity_group': 'LOC',
  'score': 0.99702173,
  'word': 'Toronto',
  'start': 57,
  'end': 64}]
```

In [16]:
```
del ner
```

As we see, the model properly identifies all entities in the sentence with highest confidence scores.

Example 5 - Question Answering

Another widely used application of Hugging Face transformers is Question Answering task. **Question Answering** is the task of extracting an answer from a document. QA models take in a context parameter, which is a document in which you are searching for some information, and a question, and return an answer. The answer is being extracted, not generated. The task is evaluated on two metrics: exact match and F1-score. As discussed in Example 3, there are also other QA models that are not extractive but generative. They generate free text directly based on the context, as shown in the Diagram below.



Let's look at the Example where we have some context and we try to extract some information from it. First, we load the `pipeline()` with "question-answering" identifier. We also input our question and content. Then we apply our model to the input.

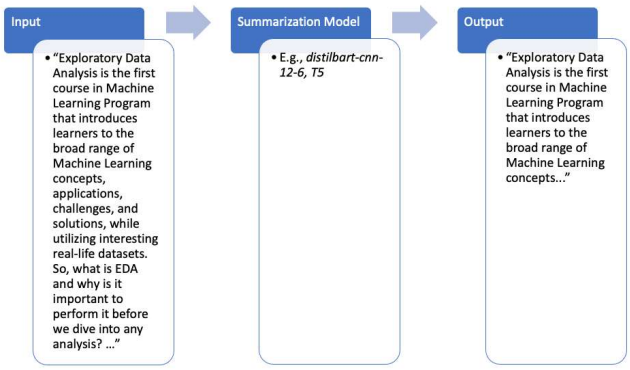
In [17]:
```
qa_model = pipeline("question-answering", model="distilbert-base-cased-distilled-sq
question = "Which name is also used to describe the Amazon rainforest in English?"
context = "The Amazon rainforest, also known in English as Amazonia or the Amazon J
qa_model(question = question, context = context)
```

Out[17]:
```
{'score': 0.8247062563896179, 'start': 48, 'end': 56, 'answer': 'Amazonia'}
```

As we see, the correct answer has been extracted with 82% confidence score.

Example 6: Text Summarization

The next Example of this Guided Project deals with Text Summarization. **Text Summarization** is the task of creating a shorter version of a document, while preserving the relevant information and importance of the original document. The summarizer model takes in the whole document as input and outputs the summarized version. The evaluation metric used in this analysis is called Rouge. It is a benchmark based on the shared sabsequent tokens between the produced sequence and the original document.



Let's load the "summarization" pipeline, input some text that we want to summarize, and see what the output will look like.

In [18]:
```
summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
summarizer(
    """
Exploratory Data Analysis is the first course in Machine Learning Program that intr
EDA is a visual and statistical process that allows us to take a glimpse into the d
As any data scientist would agree, the most challenging part in any data analysis i
This course will teach you to ‘see’ and to ‘feel’ the data as well as to transform
The course contains videos and reading materials, as well as well as a lot of inter

"""
)
```

Out[18]:
```
[{'summary_text': ' Exploratory Data Analysis is the first course in Machine Learn
ing Program that introduces learners to the broad range of Machine Learning concep
ts, applications, challenges, and solutions . EDA is a visual and statistical proc
ess that allows us to take a glimpse into the data before the analysis . It lays f
oundation for the analysis so our results go along with our expectations .'}]
```

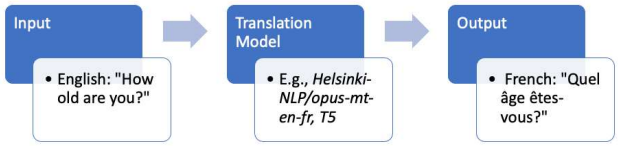
In [19]:
```
del summarizer
```

The result is a short summary of our paragraph.

Example 7 - Translation

The last Example of this Guided Project shows the Translation application. **Translation** models take an input in some source language and output the translation in a target language. The evaluation metric used for this task is called BLEU (bilingual evaluation understudy). It is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. It is on a scale from 0 to 1, 1 meaning perfect score.

The are two types of models, *monolingual* models, trained on a specific language duo data, and there are *multilingual* models, trained on multiple languages dataset.



The Example below shows a monolingual, French-English model, "translation_en_to_fr", which levereges T5-base model under the hood. These models add a task prefix, e.g., "_en_to_fr", indicating the task itself, such as translate English to French. There are also multilingual models for inference. Their inference usage differs from monolingual models. For more information on how to use multilingual models, please visit this documentation.

In [20]:
```
en_fr_translator = pipeline("translation_en_to_fr", model="t5-small")
en_fr_translator("How old are you?")
```

Out[20]:
```
[{'translation_text': 'Quel est votre âge ?'}]
```

If you would like to use a specific model that is from one specific language to another, you can also directly use the translation pipeline without specifying the model under the hood.

In [21]:
```
translator = pipeline("translation", model="Helsinki-NLP/opus-mt-fr-en")
translator("La science des données est la meilleure.")
```

Out[21]:
```
[{'translation_text': 'Data science is the best.'}]
```

Let's Practice

Exercise 1 - Sentiment Analysis

For sentiment analysis, we can also use a specific model that is better suited to our use case by providing the name of the model. For example, if we want a sentiment analysis model for tweets, we can specify the following model id: "cardiffnlp/twitter-roberta-base-sentiment". This model has been trained on ~58M tweets and fine-tuned for sentiment analysis with the "TweetEval" benchmark. The output labels for this model are: 0 -> Negative; 1 -> Neutral; 2 -> Positive.

In this Exercise, use "cardiffnlp/twitter-roberta-base-sentiment" model pre-trained on tweets data, to analyze any tweet of choice. Optionally, use the default model (used in Example 1) on the same tweet, to see if the result will change.

In [22]:
```
# TODO
specific_model = pipeline(model="cardiffnlp/twitter-roberta-base-sentiment")
data = "Artificial intelligence and automation are already causing friction in the
specific_model(data)
```

Out[22]:
```
[{'label': 'LABEL_1', 'score': 0.5272255539894104}]
```

In [23]:
```
# TODO
original_model = pipeline("sentiment-analysis")
data = "Artificial intelligence and automation are already causing friction in the
original_model(data)
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english
and revision af0f99b (https://huggingface.co/distilbert-base-uncased-finetuned-sst-2
-english).
Using a pipeline without specifying a model name and revision in production is not r
ecommended.
```

Out[23]:
```
[{'label': 'NEGATIVE', 'score': 0.9989722967147827}]
```

▶ Click here for a Hint

Exercise 2 - Topic Classification

In this Exercise, use any sentence of choice to classify it under any classes/ topics of choice. Use "zero-shot-classification" and specify the model="facebook/bart-large-mnli".

In [24]:
```
# TODO
classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
classifier(
    "I love travelling and learning new cultures",
    candidate_labels=["art", "education", "travel"],
)
```

Out[24]:
```
{'sequence': 'I love travelling and learning new cultures',
 'labels': ['travel', 'education', 'art'],
 'scores': [0.9902299642562866, 0.005778191145509481, 0.003991859499365091]}
```

Exercise 3 - Text Generation Models

In this Exercise, use 'text-generator' and 'gpt2' model to complete any sentence. Define any desirable number of returned sentences.

In [25]:
```
# TODO
generator = pipeline('text-generation', model = 'gpt2')
generator("Hello, I'm a language model", max_length = 30, num_return_sequences=3)
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

Out[25]:
```
[{'generated_text': "Hello, I'm a language modeler; I'm also a language language d
esigner, you get the general idea.\n\nWhy do you think that"},
 {'generated_text': 'Hello, I\'m a language model and a design engine," says Häns,
and it\'s been on my radar for awhile.\n\nThen'},
 {'generated_text': 'Hello, I\'m a language model developer\n\n"Can we just get ri
d of it?"\n\n- Mark Wortman\n\nWhy?'}]
```

Exercise 4 - Name Entity Recognition

In this Exercise, use any sentence of choice to extract entities: person, location and organization, using Name Entity Recognition task, specify model as "Jean-Baptiste/camembert-ner".

In [26]:
```
# TODO
nlp = pipeline("ner", model="Jean-Baptiste/camembert-ner", grouped_entities=True)
example = "Her name is Anjela and she lives in Seoul."

ner_results = nlp(example)
print(ner_results)
```

```
[{'entity_group': 'PER', 'score': 0.94814444, 'word': 'Anjela', 'start': 11, 'end':
18}, {'entity_group': 'LOC', 'score': 0.99861133, 'word': 'Seoul', 'start': 35, 'en
d': 41}]
```

Exercise 5 - Question Answering

In this Exercise, use any sentence and a question of choice to extract some information, using "distilbert-base-cased-distilled-squad" model.

In [27]:
```
# TODO
question_answerer = pipeline("question-answering", model="distilbert-base-cased-dis
question_answerer(
    question="Which lake is one of the five Great Lakes of North America?",
    context="Lake Ontario is one of the five Great Lakes of North America. It is su
)
```

Out[27]:
```
{'score': 0.9834363460540771, 'start': 0, 'end': 12, 'answer': 'Lake Ontario'}
```

Exercise 6 - Text Summarization

In this Exercise, use any document/paragraph of choice and summarize it, using "sshleifer/distilbart-cnn-12-6" model.

In [ ]:
```
# TODO
summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6",  max_
summarizer(
    """
Lake Superior in central North America is the largest freshwater lake in the world
"""
)
```

Exercise 7 - Translation

In this Exercise, use any sentence of choice to translate English to German. The translation model you can use is "translation_en_to_de".

In [ ]:
```
# TODO
translator = pipeline("translation_en_to_de", model="t5-small")
print(translator("New York is my favourite city", max_length=40))
```

Congratulations! Suraj Nate You have completed this guided project.

In [ ]:
```

```