Speech to Speech Translation

By Animesh Singh & Kshitij Vats

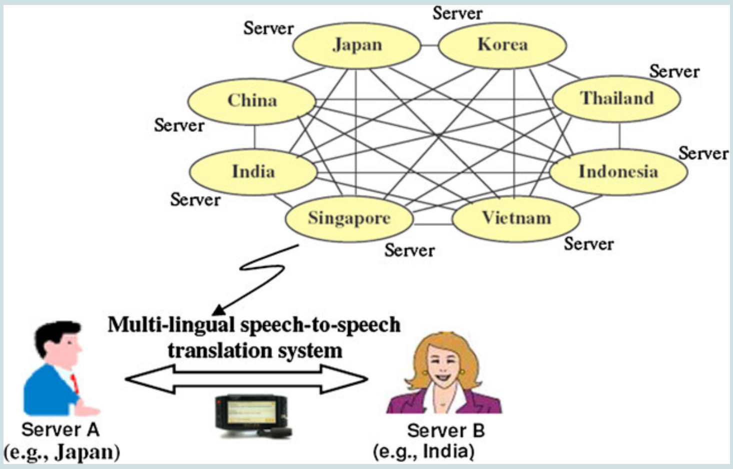
- Speech recognition: speech is recognized and transcribed.

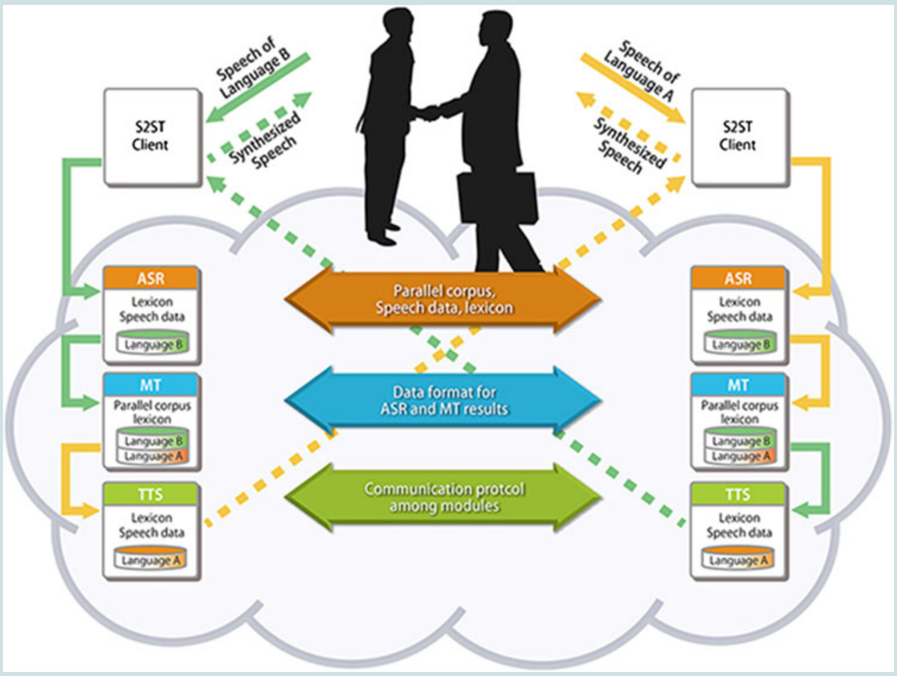
- Machine translation: text in source language is translated into text in the target language.

- Speech synthesis: speech signal is created from the text.

To construct S2ST systems, automatic speech recognition (ASR), machine translation (MT) and text-to-speech synthesis (TTS) must be built for source and target languages by collecting speech and language data, such as audio data, its manual transcriptions, pronunciation lexica for each word, parallel corpora for translation and so on.

In a loosely coupled system, these three components are connected in a sequential style: ASR converts the user's speech to text in the source language and then MT translates
the source text into the target language. Finally, TTS creates the synthesized speech from the target text.





Client application

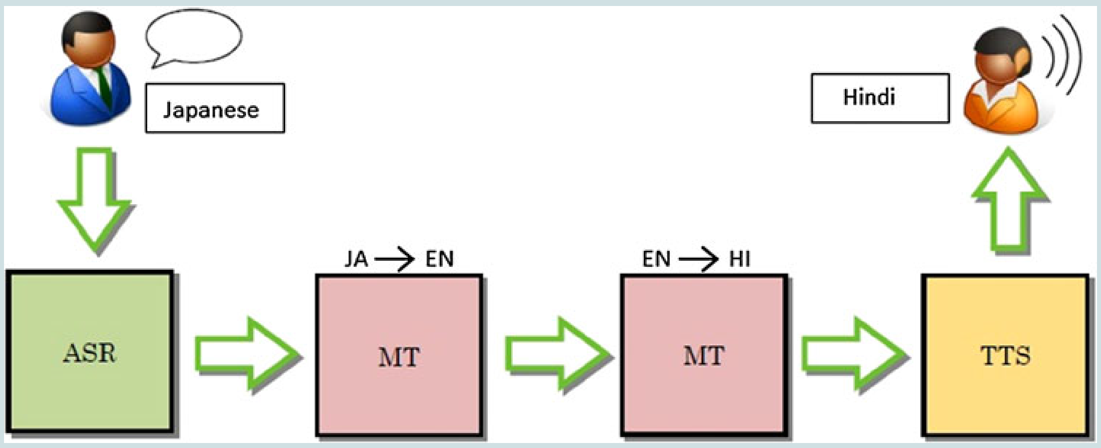
The application contributes to breaking the barriers of modalities other than language as well. For instance, it helps users to communicate with the visually-impaired via spoken word, or with the hearing-impaired via text input.

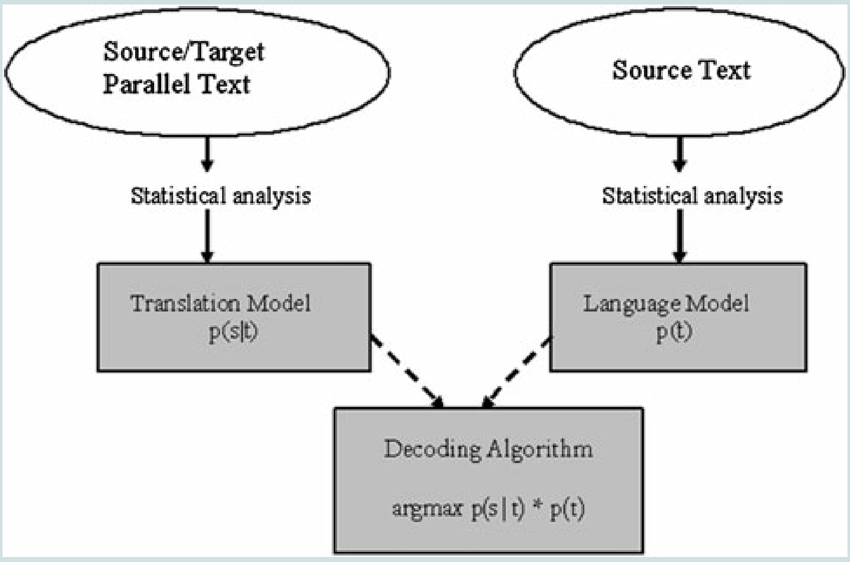
Control servers

Control server is used to relay the speech results from one user to all other users in order to enable them to perform a multiparty conversation as well as dialogue based Conversation. The Control Server at the target side decides which service(ASR, MT or TTS) to invoke and send responses back.

Communication servers
Communication Servers are designed, implemented and maintained by their respective language verticals. These servers actually perform ASR, MT and TTS service based on the "service invoke request" by the Control Server.

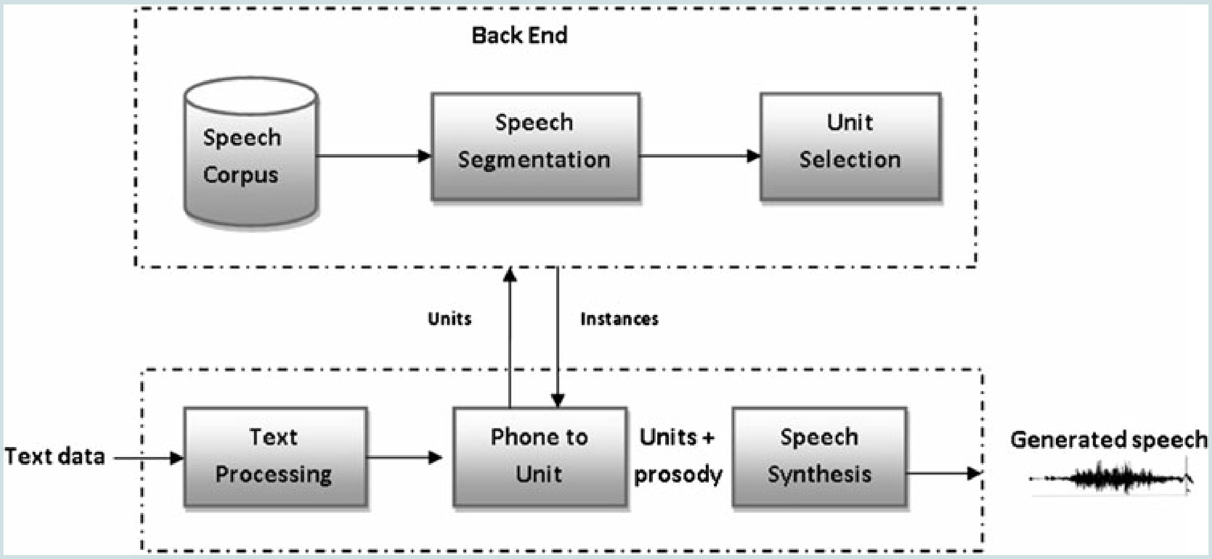
Technology
A technical definition of ASR is the building of system for mapping acoustic signals to a string of words. The general problem of automatic transcription of speech by any speaker in any environment is still far from solved. But recent years have seen ASR technology mature to the point where it is viable in certain limited domains.
Five main sub-components of an ASR system are
a. Acoustic Model (p(O|W)
b. Language Model (P(W))
c. Lexicon/Pronunciation Model (HMM)
d. Feature Extraction
e. Decoder





Standardisation

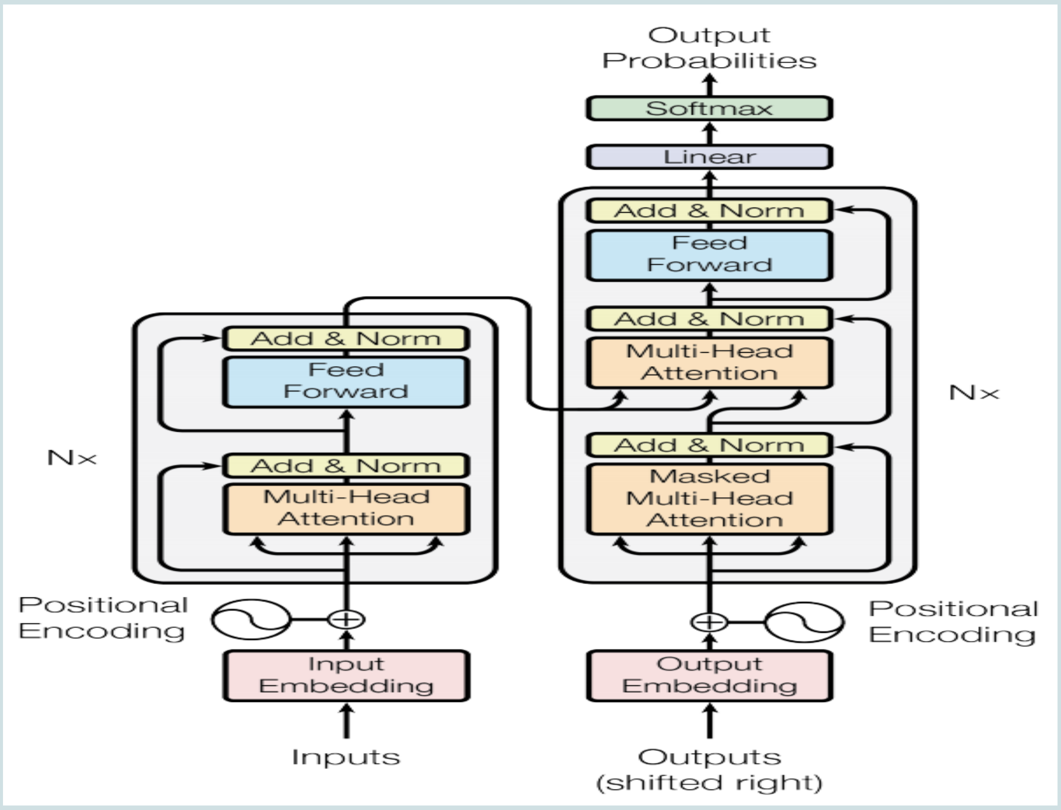
It is also necessary to create common speech-recognition and translation dictionaries, and compile standardised bilingual corpora.



Transformer

A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data.

Like recurrent neural networks (RNNs), transformers are designed to handle sequential input data, such as natural language, for tasks such as translation and text summarization. However, unlike RNNs, transformers do not necessarily process the data in order. Rather, 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 need to process the beginning of the sentence before the end. Rather it identifies the context that confers meaning to each word in the sentence. This feature allows for more parallelization than RNNs and therefore reduces training times.

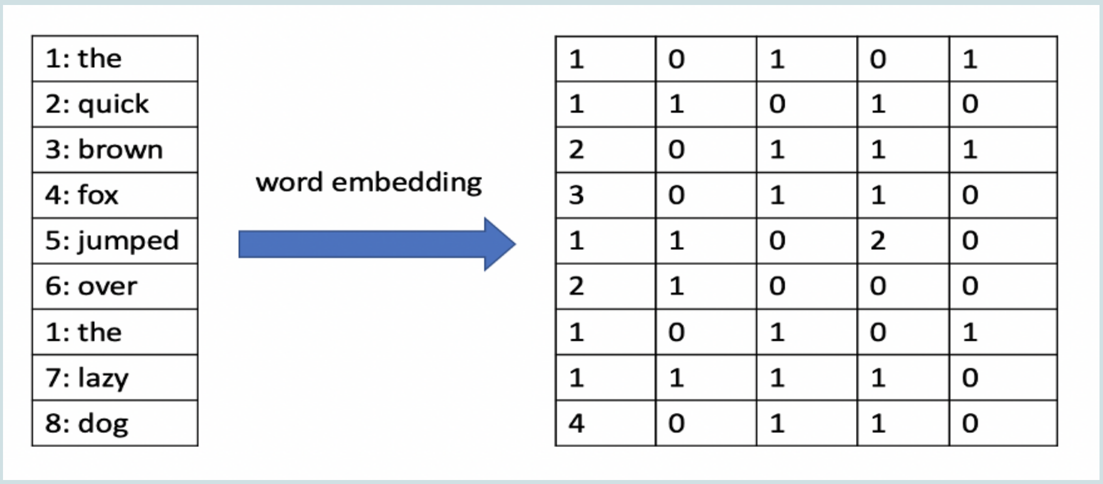


Sequential processing

Gated RNNs process tokens sequentially, maintaining a state vector that contains a representation of the data seen after every token. To process the nth token, the model combines the state representing the sentence up to token texts with the information of the new token to create a new state, representing the sentence up to token.Theoretically, the information from one token can propagate arbitrarily far down the sequence, if at every point the state continues to encode contextual information about the token. In practice this mechanism is flawed: the vanishing gradient problem leaves the model's state at the end of a long sentence without precise, extractable information about preceding tokens.

Input and Output Embedding

If you have ever worked with word embeddings using the Word2Vec algorithm, the input and output embeddings are basically just embedding layers. The embedding layer takes a sequence of words and learns a vector representation for each word.



1: the
2: quick
3: brown
4: fox
5: jumped
6: over
1: the
7: lazy
8: dog

1	0	1	0	1
1	1	0	1	0
2	0	1	1	1
3	0	1	1	0
1	1	0	2	0
2	1	0	0	0
1	0	1	0	1
1	1	1	1	0
4	0	1	1	0

Encoder–decoder architecture

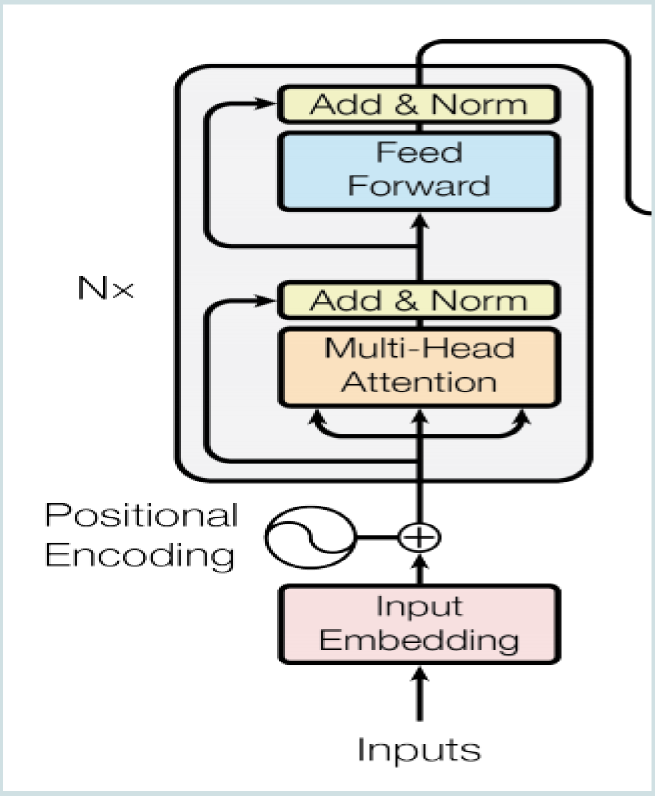
The original Transformer model used an encoder–decoder architecture. The encoder consists of encoding layers that process the input iteratively one layer after another, while the decoder consists of decoding layers that do the same thing to the encoder's output.

The function of each encoder layer is to generate encodings that contain information about which parts of the inputs are relevant to each other. It passes its encodings to the next encoder layer as inputs. Each decoder layer does the opposite, taking all the encodings and using their incorporated contextual information to generate an output sequence.To achieve this, each encoder and decoder layer makes use of an attention mechanism.

For each input, attention weighs the relevance of every other input and draws from them to produce the output. Each decoder layer has an additional attention mechanism that draws information from the outputs of previous decoders, before the decoder layer draws information from the encodings.

Encoder

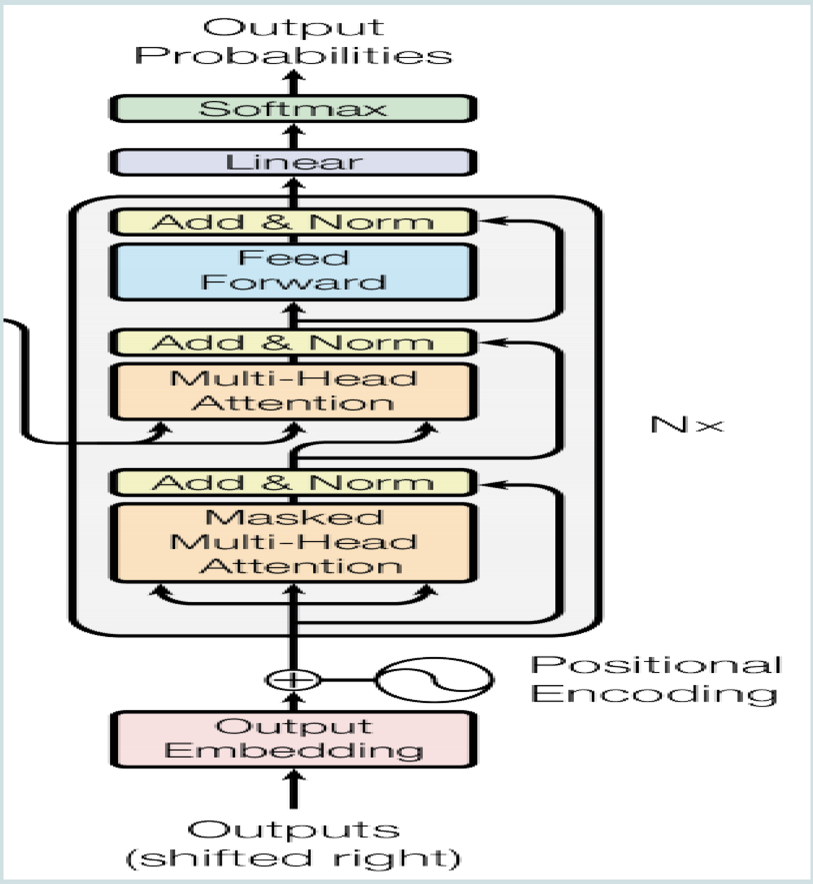
Each encoder consists of two major components: a self-attention mechanism and a feed-forward neural network. The self-attention mechanism accepts input encodings from the previous encoder and weighs their relevance to each other to generate output encodings. The feed-forward neural network further processes each output encoding individually. These output encodings are then passed to the next encoder as its input, as well as to the decoders.



Decoder

Each decoder consists of three major components: a self-attention mechanism, an attention mechanism over the encodings, and a feed-forward neural network. The decoder functions in a similar fashion to the encoder, but an additional attention mechanism is inserted which instead draws relevant information from the encodings generated by the encoders.

Like the first encoder, the first decoder takes positional information and embeddings of the output sequence as its input, rather than encodings. The transformer must not use the current or future output to predict an output, so the output sequence must be partially masked to prevent this reverse information flow.[1] The last decoder is followed by a final linear transformation and softmax layer, to produce the output probabilities over the vocabulary.



Jiwer

Jiwercan be used to transform multiple sentences into a a single sentence. The sentences can be given as a string (one sentence) or a list of strings (one or more sentences). This operation can be useful when the number of ground truth sentences and hypothesis sentences differ, and you want to do an minimal alignment over these lists. Note that this creates an invariance: wer([a, b], [a, b]) might not be equal to wer([b, a], [b, a]).

Example:

```
sentences = ["hi", "this is an example"]
print(jiwer.ReduceToSingleSentence()(sentences))
# prints: ['hi this is an example']
```

Torch-

Torch is an open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It provides a wide range of algorithms for deep learning

```
import torchaudio

def speech_file_to_array_fn(batch):
    speech_array, sampling_rate = torchaudio.load(batch["path"])
    batch["speech"] = speech_array[0].numpy()
    batch["sampling_rate"] = sampling_rate
    batch["target_text"] = batch["sentence"]
    return batch
```

Torch Audio

Torchaudio is a library for audio and signal processing with PyTorch. It provides I/O, signal and data processing functions, datasets, model implementations and application components

```
import torch

from dataclasses import dataclass, field
from typing import Any, Dict, List, Optional, Union

@dataclass
class DataCollatorCTCWithPadding:

    processor: Wav2Vec2Processor
    padding: Union[bool, str] = True
    max_length: Optional[int] = None
    max_length_labels: Optional[int] = None
    pad_to_multiple_of: Optional[int] = None
    pad_to_multiple_of_labels: Optional[int] = None
    def __call__(self, features: List[Dict[str, Union[List[int],
torch.Tensor]]]) -> Dict[str, torch.Tensor]:
        # split inputs and labels since they have to be of different
lenghts and need
        # different padding methods
        input_features = [{"input_values": feature["input_values"]} for
feature in features]
        label_features = [{"input_ids": feature["labels"]} for feature
in features]

        batch = self.processor.pad(
            input_features,
            padding=self.padding,
            max_length=self.max_length,
            pad_to_multiple_of=self.pad_to_multiple_of,
            return_tensors="pt",
        )
        with self.processor.as_target_processor():
            labels_batch = self.processor.pad(
                label_features,
                padding=self.padding,
```

Librosa

Librosa is a Python package for music and audio analysis. Librosa is basically used when we work with audio data like in music generation(using LSTM's), Automatic Speech Recognition. It provides the building blocks necessary to create the music information retrieval systems.

```
import librosa
import numpy as np

def resample(batch):
    batch["speech"] = librosa.resample(np.asarray(batch["speech"]),
48_000, 16_000)
    batch["sampling_rate"] = 16_000
    return batch
```

Wav2Vec-

```
from transformers import Wav2Vec2ForCTC
```

Wav2Vec2 is a speech model that accepts a float array corresponding to the raw waveform of the speech signal.

Datasets-

Dataset in Python has a lot of significance and is mostly used for dealing with a huge amount of data. These datasets have a certain resemblance with the packages present as part of Python 3.6 and more. Python datasets consist of dataset object which in turn comprises metadata as part of the dataset. Querying to these datasets may include dataset objects to return the required index based on rows and columns. The dataset object comes into the picture when the data gets loaded initially that also comprise the metadata consisting of other important information.