```
import tensorflow as tf
import tensorflow_datasets as tfds


# Loading NSynth dataset

dataset, info = tfds.load('nsynth', split='train', with_info=True)
print(info)
```

```
WARNING:absl:You use TensorFlow DType <dtype: 'string'> in tfds.features This will soon be deprecated in favor of NumPy DTypes. I
WARNING:absl:You use TensorFlow DType <dtype: 'bool'> in tfds.features This will soon be deprecated in favor of NumPy DTypes. In
Downloading and preparing dataset 73.07 GiB (download: 73.07 GiB, generated: 73.09 GiB, total: 146.16 GiB) to /root/tensorflow_da
Dl Completed...: 100% 1069/1069 [14:17<00:00, 1.59 file/s]
Dataset nsynth downloaded and prepared to /root/tensorflow_datasets/nsynth/full/2.3.3. Subsequent calls will reuse this data.
tfds.core.DatasetInfo(
    name='nsynth',
    full_name='nsynth/full/2.3.3',
    description="""
    The NSynth Dataset is an audio dataset containing ~300k musical notes, each with
    a unique pitch, timbre, and envelope. Each note is annotated with three
    additional pieces of information based on a combination of human evaluation and
    heuristic algorithms: Source, Family, and Qualities.
    """,
    config_description="""
    Full NSynth Dataset is split into train, valid, and test sets, with no
    instruments overlapping between the train set and the valid/test sets.

    """,
    homepage='https://g.co/magenta/nsynth-dataset',
    data_dir='/root/tensorflow_datasets/nsynth/full/incomplete.KESCY0_2.3.3/',
    file_format=tfrecord,
    download_size=73.07 GiB,
    dataset_size=73.09 GiB,
    features=FeaturesDict({
        'audio': Audio(shape=(64000,), dtype=float32),
        'id': string,
        'instrument': FeaturesDict({
            'family': ClassLabel(shape=(), dtype=int64, num_classes=11),
            'label': ClassLabel(shape=(), dtype=int64, num_classes=1006),
            'source': ClassLabel(shape=(), dtype=int64, num_classes=3),
        }),
        'pitch': ClassLabel(shape=(), dtype=int64, num_classes=128),
        'qualities': FeaturesDict({
            'bright': bool,
            'dark': bool,
            'distortion': bool,
            'fast_decay': bool,
            'long_release': bool,
            'multiphonic': bool,
            'nonlinear_env': bool,
            'percussive': bool,
            'reverb': bool,
            'tempo-synced': bool,
        }),
        'velocity': ClassLabel(shape=(), dtype=int64, num_classes=128),
    }),
    supervised_keys=None,
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=4096, num_shards=8>,
        'train': <SplitInfo num_examples=289205, num_shards=1024>,
        'valid': <SplitInfo num_examples=12678, num_shards=32>,
    },
    citation="""@InProceedings{pmlr-v70-engel17a,
      title =    {Neural Audio Synthesis of Musical Notes with {W}ave{N}et Autoencoders},
      author =   {Jesse Engel and Cinjon Resnick and Adam Roberts and Sander Dieleman and Mohammad Norouzi and Douglas Eck and Ka
      booktitle =        {Proceedings of the 34th International Conference on Machine Learning},
      pages =    {1068--1077},
      year =     {2017},
      editor =   {Doina Precup and Yee Whye Teh},
      volume =   {70},
      series =   {Proceedings of Machine Learning Research},
      address =          {International Convention Centre, Sydney, Australia},
      month =    {06--11 Aug},
      publisher =        {PMLR},
      pdf =      {http://proceedings.mlr.press/v70/engel17a/engel17a.pdf},
      url =      {http://proceedings.mlr.press/v70/engel17a.html},
    }""",
```

Structure of dataset :-

```
for sample in dataset.take(1):
  print("Available keys: \n")
  for key in sample.keys():
    print(key)
```

```
Available keys:

audio
id
instrument
pitch
qualities
velocity
```

Preprocessing the dataset :-

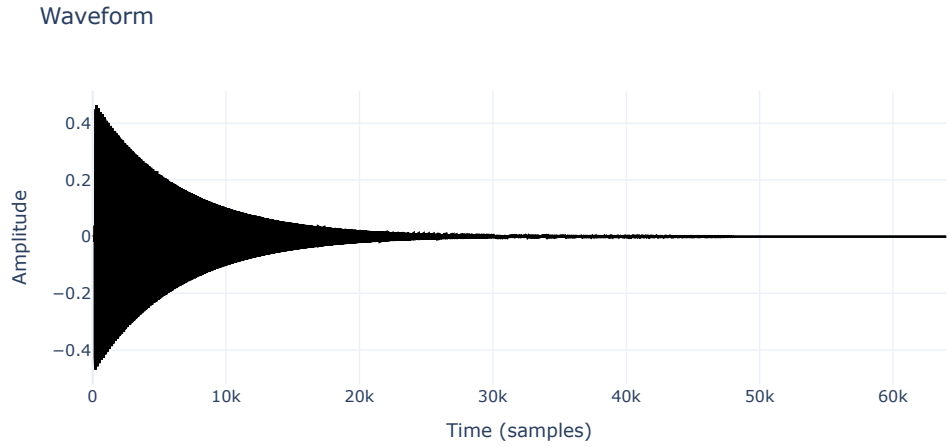
```
# Extract audio and an alternate label (ex. pitch)

def preprocess_nsynth(sample):
  audio = sample['audio']
  label = sample['pitch']
  return audio, label

# Preprocessing
processed_dataset = dataset.map(preprocess_nsynth)

for audio, label in processed_dataset.take(1):
  print(f"Audio shape: {audio.shape}")
  print(f"Label (Pitch): {label.numpy()}")
```

```
Audio shape: (64000,)
Label (Pitch): 106
```

```
# Converting the audio tensor to a NumPy array and playing it using the IPython Audio display

from IPython.display import Audio

audio_np = audio.numpy()
Audio(audio_np, rate=16000)   # Assuming a sample rate of 16kHz
```

Visualizing the dataset :-

```
import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(
    y = audio_np,
    mode ='lines',
    line = dict(color = 'black'),
    name = "Waveform"
))

fig.update_layout(
    title = "Waveform",
    xaxis_title = "Time (samples)",
    yaxis_title = "Amplitude",
    template = "plotly_white",
    width = 800,
    height = 400
)

fig.show()
```

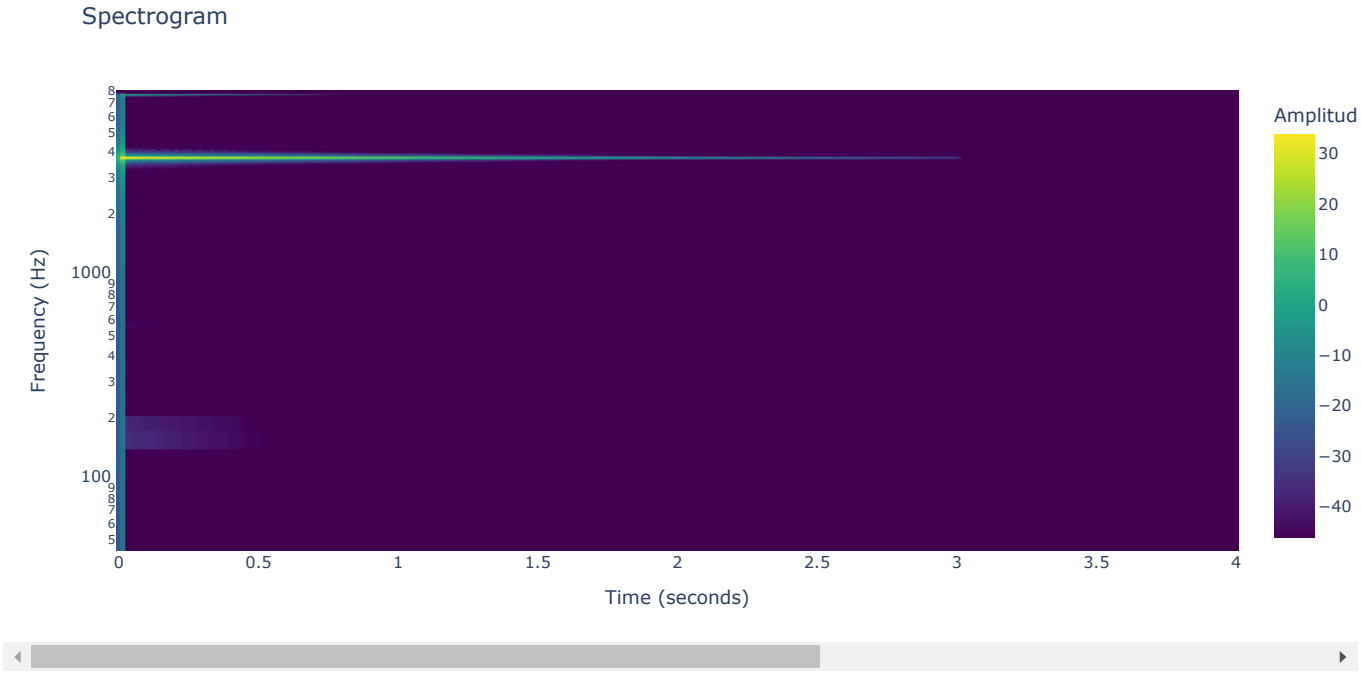


The waveform graph displays a decaying amplitude over time, starting with a high magnitude and gradually tapering off to zero. This indicates that the audio signal begins with a strong onset, followed by a rapid decay in energy.

Such behaviour is typical in audio signals like percussive notes or short instrumental sounds, where the initial strike produces high energy that dissipates quickly.

Analyzing the Spectrogram :-

```
import librosa
import numpy as np

# Compute the STFT

spectrogram = librosa.stft(audio_np, n_fft=512, hop_length=256)
spectrogram_db = librosa.amplitude_to_db(abs(spectrogram))

time = np.linspace(0, len(audio_np)/16000, spectrogram_db.shape[1])
frequencies = np.linspace(0, 16000 / 2, spectrogram_db.shape[0])

fig = go.Figure(data=go.Heatmap(
      z=spectrogram_db,
      x=time,
      y=frequencies,
      colorscale='Viridis',
      colorbar=dict(title='Amplitude (dB)'),
      ))

fig.update_layout(
    title="Spectrogram",
    xaxis_title="Time (seconds)",
    yaxis_title="Frequency (Hz)",
    yaxis = dict(type='log'),
    template="plotly"
)

fig.show()
```

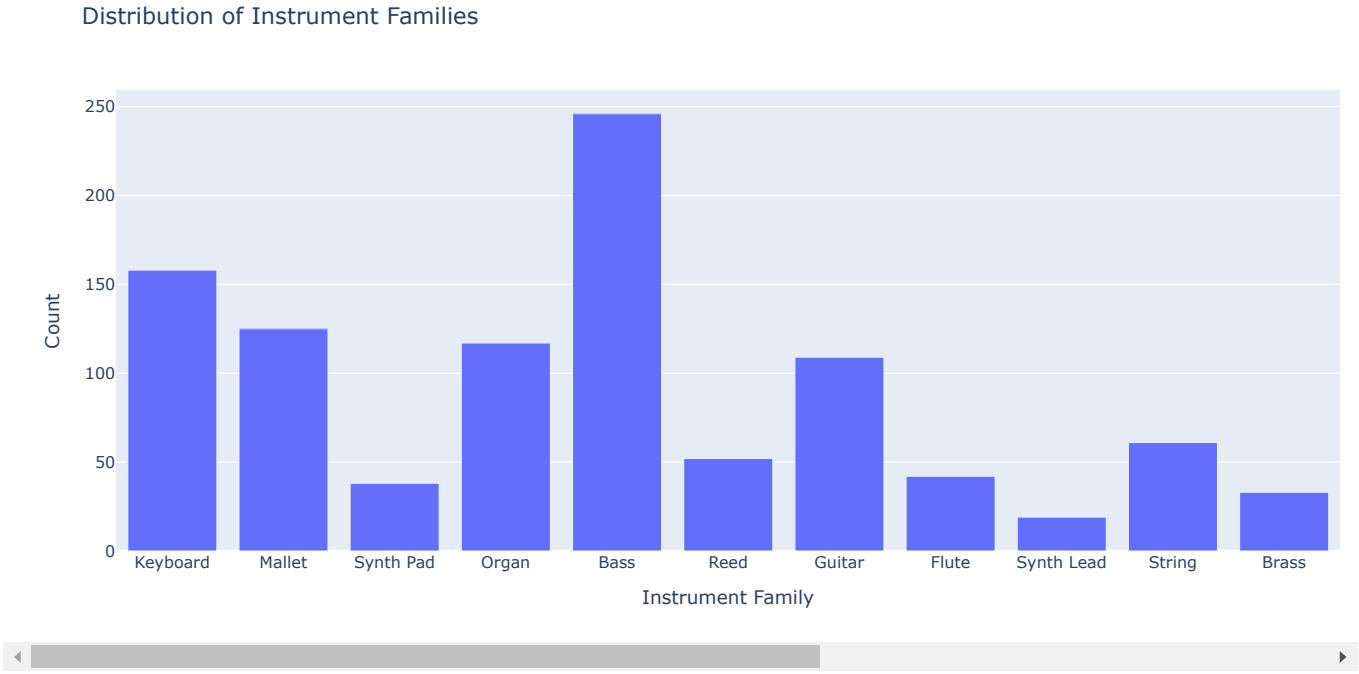


The spectrogram reveals that the audio signal primarily contains a single prominent frequency component around 400 Hz, which remains consistent throughout its duration.

The amplitude of this frequency is high, as indicated by the bright color, while other frequencies show minimal or no energy. The faint low-frequency components near the start of the signal suggest a brief presence of low-pitched content. This pattern suggests a sustained note, likely from a single instrument, with little harmonic variation or timbral complexity.

Analyzing Instrument Distribution :-

```
from collections import Counter

# Counting instrument occurrences
instrument_counts = Counter()

for sample in dataset.take(1000):
  instrument = sample['instrument']['family'].numpy()
  instrument_counts[instrument] += 1

# Mapping numeric IDs to instrument family names

instrument_families = ["Bass", "Brass", "Flute", "Guitar", "Keyboard", "Mallet", "Organ", "Reed", "String", "Synth Lead", "Synth Pad", '
mapped_family_counts = {instrument_families[family_id]: count for family_id, count in instrument_counts.items()}


import plotly.express as px
fig = px.bar(
    x=list(mapped_family_counts.keys()),
    y=list(mapped_family_counts.values()),
    labels={'x': 'Instrument Family', 'y': 'Count'},
    title="Distribution of Instrument Families",
    template="plotly"
)

fig.show()
```



The Bass family has the highest count, with around 250 occurrences, followed by the Keyboard and Mallet families, which have moderate representation.

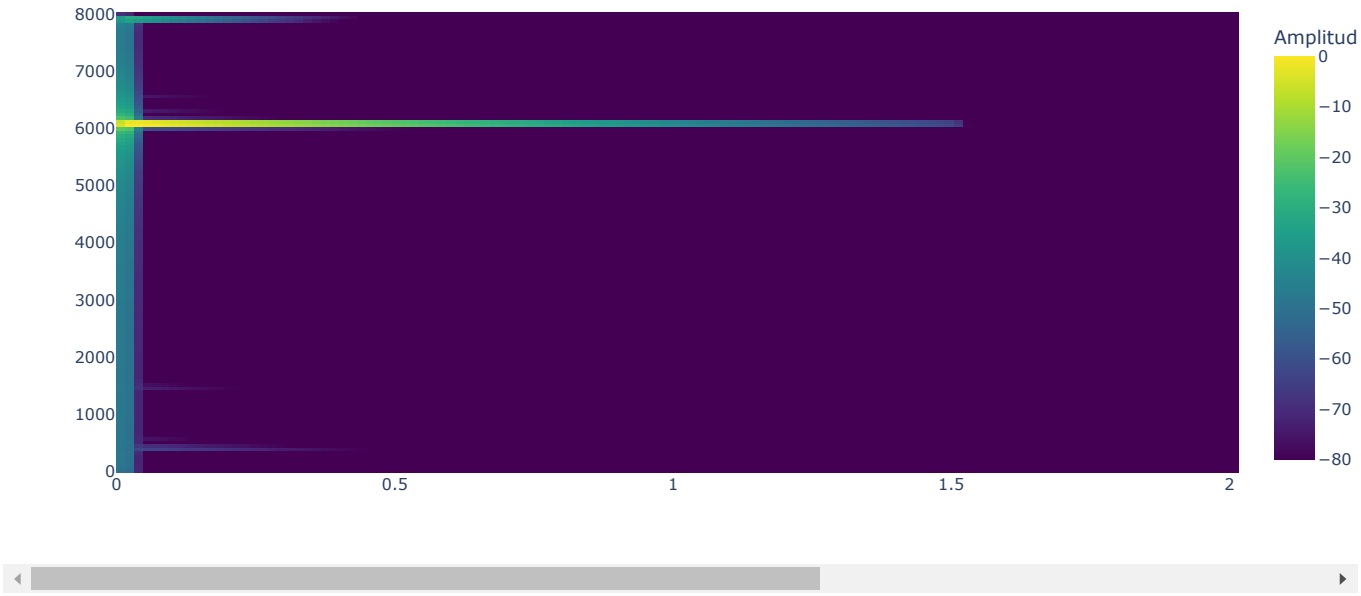
In contrast, instrument families like Synth Lead, Flute, and Brass have the lowest counts.

This distribution suggests that the dataset emphasizes bass and keyboard instruments, while certain families are underrepresented, which could impact tasks like classification or model training.

Mel Spectrogram Analysis :-

The Mel spectrogram translates audio frequencies into the Mel scale, to simulate human perception of sound.

```
mel_spectrogram = librosa.feature.melspectrogram(y=audio_np, sr=16000, n_mels=128)
mel_spectrogram_db = librosa.power_to_db(mel_spectrogram, ref=np.max)

fig = go.Figure(data = go.Heatmap(
    z = mel_spectrogram_db,
    x = time,
    y = np.linspace(0, 16000 / 2, mel_spectrogram_db.shape[0]),
    colorscale = 'Viridis',
    colorbar = dict(title="Amplitude (dB)")
))

fig.show()
```

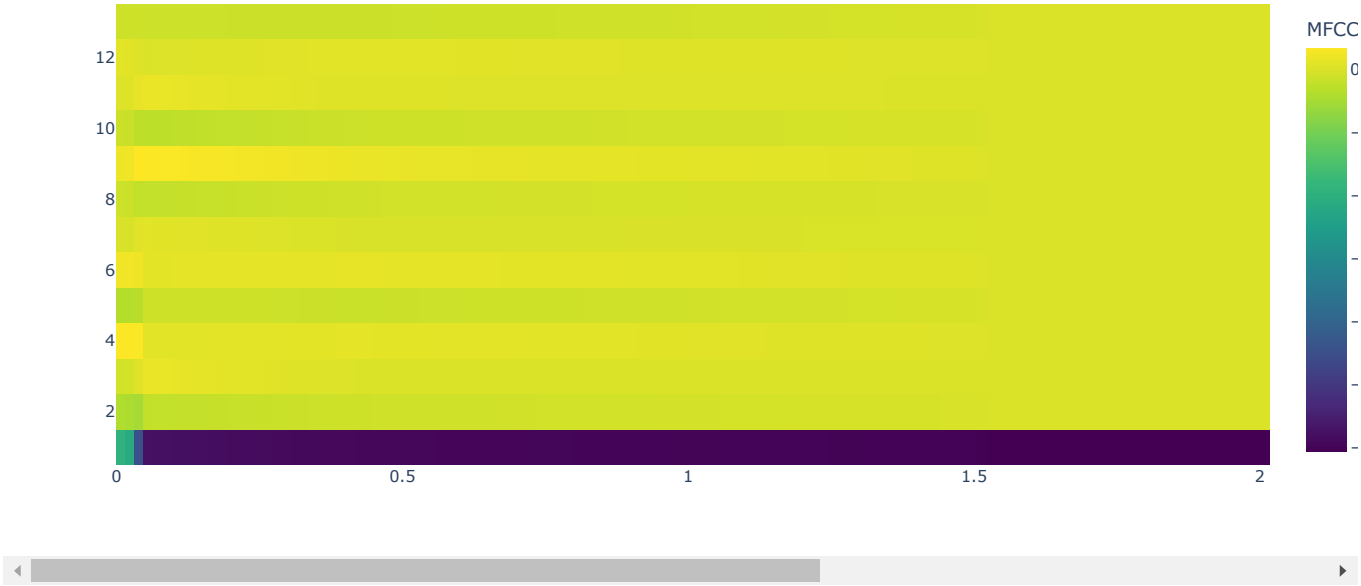


The Mel spectrogram highlights that the audio signal has a prominent frequency component (yellow-green band) at 6,000 Hz, which remains sustained throughout the clip. It indicates a stable tone with high energy.

Lower frequencies below 1,000 Hz display much weaker energy, which suggests minimal low-pitched content.

Mel-Frequency Cepstral Coefficients (MFCC) Analysis :-

```
mfccs = librosa.feature.mfcc(y=audio_np, sr=16000, n_mfcc=13)

fig = go.Figure(data=go.Heatmap(
    z=mfccs,
    x=time,
    y=np.arange(1, mfccs.shape[0] + 1),
    colorscale='Viridis',
    colorbar=dict(title="MFCC Value")
))

fig.show()
```



The MFCC graph highlights the spectral features of the audio signal over time.

The first MFCC coefficient shows a significantly lower magnitude compared to the others, indicating it captures the signal's overall energy. The remaining coefficients exhibit relatively uniform values across time, which suggests the audio signal has a stable frequency content without major timbral variations.

Transforming Audio Data :-

```
# Pitch shifting (+2 semitones)

audio_pitch_shifted = librosa.effects.pitch_shift(audio_np, sr=16000, n_steps=2)

# Time-stretching (speed up by 1.5x)

audio_time_stretched = librosa.effects.time_stretch(audio_np, rate=1.5)

# plot waveforms
```