```
!pip install librosa numpy pandas matplotlib IPython --quiet
!pip install soundfile hmmlearn --quiet
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 24.4 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 165.9/165.9 kB 10.4 MB/s eta 0:00:00
```

```
from google.colab import files
files.upload()

!mkdir -p ~/.kaggle
!mv "kaggle (1).json" ~/.kaggle/kaggle.json
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d desolationofsmaug/saraga-carnatic-music-dataset

!unzip saraga-carnatic-music-dataset.zip -d saraga
```

Show hidden output

```
import librosa
import numpy as np
import os
import matplotlib.pyplot as plt



swaras = ['Sa', 'Ri1', 'Ri2', 'Ga2', 'Ga3', 'Ma1', 'Ma2', 'Pa', 'Da1', 'Da2', 'Ni2', 'Ni3']
svara_ratios = [1.0, 256/243, 9/8, 32/27, 5/4, 4/3, 45/32, 3/2, 128/81, 5/3, 16/9, 15/8]

def freq_to_swara(frequency, tonic=261.63):
    if frequency < 50:
        return 'Silence'
    swara_freqs = np.array(svara_ratios) * tonic
    index = np.argmin(np.abs(swara_freqs - frequency))
    return swaras[index]


def extract_swaras_from_audio(file_path, tonic=261.63):
    y, sr = librosa.load(file_path)
    pitches, magnitudes = librosa.piptrack(y=y, sr=sr)

    swara_seq = []
    for i in range(pitches.shape[1]):
        index = magnitudes[:, i].argmax()
        pitch = pitches[index, i]
        swara = freq_to_swara(pitch, tonic)
        swara_seq.append(swara)
    return swara_seq


from tqdm.notebook import tqdm



from tqdm import tqdm

import glob

audio_files = glob.glob("saraga/carnatic/*/*.mp3")

all_swaras = []

for file_path in tqdm(audio_files, desc="Extracting swaras"):
    try:
        swaras_from_file = extract_swaras_from_audio(file_path)
        all_swaras.extend(swaras_from_file)
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
```

```
Extracting swaras: 100%|██████████| 197/197 [15:45<00:00,  4.80s/it]
```

```
print(all_swaras)
```

```
['Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'Ni3', 'N
```

```
import pandas as pd

df = pd.DataFrame({'Swara': all_swaras})
df.to_csv('extracted_swaras.csv', index=False)

from google.colab import files
files.download('extracted_swaras.csv')
```

```
import pandas as pd

df = pd.read_csv("extracted_swaras.csv")
swara_sequence = [s for s in df['Swara'] if s != "Silence"]  # Optional: remove silence
```

```
from collections import defaultdict
import random
import pandas as pd

class VOGUEModel:
    def __init__(self, max_order=3):
        self.max_order = max_order
        self.model = defaultdict(lambda: defaultdict(int))

    def train(self, sequence):
        self.sequence = sequence  # Save for seeding
        for order in range(1, self.max_order + 1):
            for i in range(len(sequence) - order):
                context = tuple(sequence[i:i+order])
                next_swara = sequence[i+order]
                self.model[context][next_swara] += 1

    def generate(self, length=50):
        # Use a random seed from training data
        seed = self.sequence[:self.max_order]
        generated = list(seed)
        for _ in range(length):
            for order in reversed(range(1, self.max_order + 1)):
                context = tuple(generated[-order:])
                if context in self.model:
                    next_token = random.choices(
                        list(self.model[context].keys()),
                        weights=self.model[context].values()
                    )[0]
                    generated.append(next_token)
                    break
            else:
                # fallback if no context matches
                generated.append(random.choice(self.sequence))
        return generated

# Example usage
# swara_sequence = [...]  # Replace this with your real input list

vogue = VOGUEModel(max_order=3)
vogue.train(swara_sequence)

generated_swaras = vogue.generate(length=200)

pd.DataFrame({'Generated_Swara': generated_swaras}).to_csv('generated_swaras.csv', index=False)
```

```
import numpy as np
from scipy.io.wavfile import write

def swara_to_freq(swara, tonic=261.63):
    swaras = ['Sa', 'Ri1', 'Ri2', 'Ga2', 'Ga3', 'Ma1', 'Ma2', 'Pa', 'Da1', 'Da2', 'Ni2', 'Ni3']
    ratios = [1.0, 256/243, 9/8, 32/27, 5/4, 4/3, 45/32, 3/2, 128/81, 5/3, 16/9, 15/8]
    if swara not in swaras:
        return 0
    return tonic * ratios[swaras.index(swara)]
```

```python
def synthesize_swara_sequence(sequence, duration=0.3, sr=22050):
    audio = np.array([], dtype=np.float32)
    for swara in sequence:
        freq = swara_to_freq(swara)
        if freq == 0:
            tone = np.zeros(int(duration * sr))
        else:
            t = np.linspace(0, duration, int(duration * sr), False)
            tone = 0.5 * np.sin(2 * np.pi * freq * t)
        audio = np.concatenate([audio, tone])
    return audio

audio = synthesize_swara_sequence(generated_swaras)
write('generated_swaras.wav', 22050, (audio * 32767).astype(np.int16))

from google.colab import files
files.download('generated_swaras.wav')
```

```python
# VOGUE class: Variable Order & Gapped HMM
class VOGUE:
    def __init__(self, order=2, gap=1):
        self.order = order
        self.gap = gap
        self.model = defaultdict(lambda: defaultdict(int))

    def train(self, sequence):
        for i in range(len(sequence) - self.order * (self.gap + 1)):
            context = tuple(sequence[i + j * (self.gap + 1)] for j in range(self.order))
            next_token = sequence[i + self.order * (self.gap + 1)]
            self.model[context][next_token] += 1

    def generate_sequence(self, length=100):
        if not self.model:
            return []

        context = random.choice(list(self.model.keys()))
        generated = list(context)

        while len(generated) < length:
            next_probs = self.model.get(tuple(context), None)
            if not next_probs:
                break
            next_token = random.choices(list(next_probs.keys()), weights=next_probs.values())[0]
            generated.append(next_token)
            context = list(context[1:]) + [next_token]
        return generated


import librosa
import numpy as np
import glob
from tqdm import tqdm
import os
import pandas as pd

swaras = ['Sa', 'Ri1', 'Ri2', 'Ga2', 'Ga3', 'Ma1', 'Ma2', 'Pa', 'Da1', 'Da2', 'Ni2', 'Ni3']
svara_ratios = [1.0, 256/243, 9/8, 32/27, 5/4, 4/3, 45/32, 3/2, 128/81, 5/3, 16/9, 15/8]

def freq_to_swara(frequency, tonic=261.63):
    if frequency < 50:
        return 'Silence'
    swara_freqs = np.array(svara_ratios) * tonic
    index = np.argmin(np.abs(swara_freqs - frequency))
    return swaras[index]

def extract_swaras_from_audio(file_path, tonic=261.63):
    y, sr = librosa.load(file_path)
    pitches, magnitudes = librosa.piptrack(y=y, sr=sr)

    swara_seq = []
    for i in range(pitches.shape[1]):
        index = magnitudes[:, i].argmax()
        pitch = pitches[index, i]
        swara = freq_to_swara(pitch, tonic)
        swara_seq.append(swara)
    return swara_seq

# Update this to reflect your real folder path
audio_files = glob.glob("/content/saraga/carnatic/*/*.mp3")

data = []

for file_path in tqdm(audio_files):
    try:
        raga = os.path.basename(os.path.dirname(file_path))
        swaras_from_file = extract_swaras_from_audio(file_path)
        for swara in swaras_from_file:
            data.append({"Raga": raga, "Swara": swara})
    except Exception as e:
        print(f"Failed on {file_path}: {e}")

df = pd.DataFrame(data)
df.to_csv("raga_swara_dataset.csv", index=False)
```

```
100%|██████████| 197/197 [15:14<00:00,  4.64s/it]
```

```
import os
import json
import pandas as pd

saraga_dir = "/content/saraga/carnatic"  # Update if needed
raga_mappings = []

for folder in os.listdir(saraga_dir):
    folder_path = os.path.join(saraga_dir, folder)
    if os.path.isdir(folder_path):
        for file in os.listdir(folder_path):
            if file.endswith(".json"):
                json_path = os.path.join(folder_path, file)
                try:
                    with open(json_path, "r") as f:
                        data = json.load(f)
                        raaga_info = data.get("raaga", {})

                        # Handle both dict and list
                        if isinstance(raaga_info, dict):
                            raga_name = raaga_info.get("name", "Unknown")
                        elif isinstance(raaga_info, list) and len(raaga_info) > 0:
                            raga_name = raaga_info[0].get("name", "Unknown")
                        else:
                            raga_name = "Unknown"

                        raga_mappings.append({"Raga": int(folder), "Raga_Name": raga_name})
                except Exception as e:
                    print(f"Error reading {json_path}: {e}")



raga_df = pd.DataFrame(raga_mappings).drop_duplicates()
raga_df
```

	Raga	Raga_Name
0	63	Bhairavi
1	115	Kedāraṁ
2	90	Saurāṣṭraṁ
3	51	Ābhōgi
4	104	Nādanāmakriya
...	...	...
192	114	Saurāṣṭraṁ
193	171	Latāngi
194	33	Kamās
195	72	Gaṁbhīra nāṭa
196	173	Lalita pancamaṁ

197 rows × 2 columns

```
# Load the extracted swaras file
swaras_df = pd.read_csv("/content/raga_swara_dataset.csv")
merged_df = swaras_df.merge(raga_df, how="left", left_on="Raga", right_on="Raga")
merged_df.head()
```

	Raga	Swara	Raga_Name
0	63	Ni3	Bhairavi
1	63	Ni3	Bhairavi
2	63	Ni3	Bhairavi
3	63	Ni3	Bhairavi
4	63	Ni3	Bhairavi

```
swaras_df.columns
```

```
Index(['Raga', 'Swara'], dtype='object')
```

```
# Assume you have a DataFrame `merged_df` with columns: Swara, Raga, Raga_Name
raga_groups = merged_df.groupby('Raga')
```

```
from collections import defaultdict
import random

class VOGUEModel:
    def __init__(self, max_order=3):
        self.max_order = max_order
        self.model = defaultdict(lambda: defaultdict(int))

    def train(self, sequence):
        for order in range(1, self.max_order + 1):
            for i in range(len(sequence) - order):
                context = tuple(sequence[i:i+order])
                next_swara = sequence[i+order]
                self.model[context][next_swara] += 1

    def predict_next(self, context):
        if context not in self.model:
            return random.choice(list(set([k for ctx in self.model.keys() for k in self.model[ctx]])))
        next_sw_freq = self.model[context]
        return max(next_sw_freq, key=next_sw_freq.get)

    def generate_sequence(self, seed, length):
        result = list(seed)
        for _ in range(length):
            context = tuple(result[-self.max_order:])
            next_swara = self.predict_next(context)
            result.append(next_swara)
        return result

# Train models per raga
vogue_models = {}
for raga, group in raga_groups:
    swara_sequence = group['Swara'].tolist()
    model = VOGUEModel(max_order=4)
    model.train(swara_sequence)
    vogue_models[raga] = model



generated_sequences = []

for raga, model in vogue_models.items():
    try:
        group = raga_groups.get_group(raga)
        swara_list = group['Swara'].tolist()

        if len(swara_list) < 4:
            print(f"Not enough swaras for Raga {raga}, skipping...")
            continue

        seed = swara_list[:3]  # Initial context swaras
        generated = model.generate_sequence(seed, length=50)

        generated_sequences.append({
            'Raga': raga,
            'Seed': " ".join(seed),
            'Generated_Swaras': " ".join(generated)
        })

        print(f"✅ Generated for Raga {raga}: {' '.join(generated)}")

    except Exception as e:
        print(f"❌ Error generating for Raga {raga}: {e}")
```

```
✅ Generated for Raga 182: Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni
✅ Generated for Raga 183: Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri2 Ri
✅ Generated for Raga 184: Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni
✅ Generated for Raga 185: Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni
✅ Generated for Raga 186: Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni
✅ Generated for Raga 187: Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni
✅ Generated for Raga 188: Ni3 Ma1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri
✅ Generated for Raga 189: Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni
✅ Generated for Raga 190: Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri
✅ Generated for Raga 191: Sa Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1
✅ Generated for Raga 192: Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri
✅ Generated for Raga 193: Sa Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1
✅ Generated for Raga 194: Ri2 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri
✅ Generated for Raga 195: Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri
✅ Generated for Raga 196: Sa Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1 Ri1
```

```
gen_df = pd.DataFrame(generated_sequences)
gen_df.to_csv("generated_swaras.csv", index=False)
gen_df.head()
```

	Raga	Seed	Generated_Swaras
0	0	Ni3 Ni3 Ni3	Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni...
1	1	Ni3 Ni3 Ni3	Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni...
2	2	Ni3 Ni3 Ni3	Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni...
3	3	Da2 Ni3 Ni3	Da2 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni3 Ni...
4	4	Ni3 Ni3 Ni2	Ni3 Ni3 Ni2 Ni2 Ni2 Ni2 Ni2 Ni2 Ni2 Ni2 Ni2 Ni...

```
for idx, row in gen_df.iterrows():
    raga = row['Raga']
    swara_sequence = row['Generated_Swaras'].split()
    output_file = f"{raga}_generated.wav"
    swaras_to_audio(swara_sequence, file_name=output_file)
```