IMPLEMENTATION

Building mode:

```
from  tensorflow.keras.models import  Model
from  tensorflow.keras.layers import Input,  Dense,Dropout,Embedding,Add,LayerNormalization,MultiHeadAttention
, GlobalAveragePooling1D
from tensorflow.keras.optimizers  import  Adam

def transformer_encoder(inputs,  head_size,  num_heads,  ff_dim,  dropout=0):
    # Normalization and Attention
    x =  LayerNormalization(epsilon=1e-6)(inputs)
    x =  MultiHeadAttention(key_dim=head_size,  num_heads=num_heads,  dropout=dropout)(x,  x)
    x = Dropout(dropout)(x)
    res =  x  +  inputs

    # Feed Forward Part
    x =  LayerNormalization(epsilon=1e-6)(res)
    x = Dense(ff_dim, activation="relu")(x)
    x =  Dropout(dropout)(x)
    x = Dense(inputs.shape[-1])(x)
    return x + res

def build_model(input_shape, head_size,num_heads,ff_dim,num_transformer_blocks, mlp_units, dropout=0,
 mlp_dropout=0):
    inputs =  Input(shape=input_shape)
    x = inputs
    for _  in  range(num_transformer_blocks):
        x =  transformer_encoder(x,  head_size,  num_heads,  ff_dim,  dropout)

    x = GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x =  Dense(dim,  activation="relu")(x)
        x = Dropout(mlp_dropout)(x)
    outputs = Dense(y.shape[1], activation="softmax")(x)
    return Model(inputs, outputs)

input_shape  =  (X.shape[1],  X.shape[2])     # (sequence_length, input_dim)
model = build_model(
    input_shape,
    head_size=256,
    num_heads=4,
    ff_dim=4,
    num_transformer_blocks=4,
    mlp_units=[128],
    dropout=0.1,
    mlp_dropout=0.1,
  )
```

Generating Melody

```
def Malody_Generator(Note_Count):
    seed = X_seed[np.random.randint(0,len(X_seed)-1)]
    Music = ""
    Notes_Generated=[]
    for i  in  range(Note_Count):
        seed  =  seed.reshape(1,length,1)
        prediction   =   model.predict(seed,   verbose=0)[0]
        prediction   =  np.log(prediction)  /  1.0  #diversity
        exp_preds = np.exp(prediction)
        prediction = exp_preds / np.sum(exp_preds)
        index = np.argmax(prediction)
        index_N = index/ float(L_symb)
        Notes_Generated.append(index)
        Music = [reverse_mapping[char] for char in Notes_Generated]
        seed = np.insert(seed[0],len(seed[0]),index_N)
        seed =  seed[1:]
    #Now, we have music in form or a list of chords and notes and we want to be a midi file.
    Melody = chords_n_notes(Music)
    Melody_midi = stream.Stream(Melody)
    return Music,Melody_midi


#getting the Notes and Melody created by the model
Music_notes, Melody = Malody_Generator(100)
show(Melody)
```

Printing the music sheet

```
def show(music):
#     display(Image(str(music.write("lily.png"))))
    pass

def chords_n_notes(Snippet):
    Melody = []
    offset =  0  #Incremental
    for i  in  Snippet:
        #If it is chord
        if ("."  in  i  or  i.isdigit()):
            chord_notes =  i.split(".")  #Seperating the notes in chord
            notes =  []
            for j in chord_notes:
                inst_note=int(j)
                note_snip = note.Note(inst_note)
                notes.append(note_snip)
                chord_snip = chord.Chord(notes)
                chord_snip.offset = offset
```

```
                    Melody.append(chord_snip)
          # pattern is a note
          else:
               note_snip  =  note.Note(i)
               note_snip.offset  =  offset
               Melody.append(note_snip)
          # increase offset each iteration so that notes do not stack
          offset += 1
     Melody_midi  =  stream.Stream(Melody)
     return Melody_midi

Melody_Snippet = chords_n_notes(Corpus[:100])
show(Melody_Snippet)
```

Examine all the notes in the Corpus

```
count_num =  Counter(Corpus)
print("Total unique notes in the Corpus:", len(count_num))
```

```
Notes = list(count_num.keys())
Recurrence = list(count_num.values())
#Average recurrenc for a note in Corpus
def Average(lst):
     return sum(lst)  /  len(lst)
print("Average recurrenc for a note in Corpus:", Average(Recurrence))
print("Most frequent note in Corpus appeared:", max(Recurrence), "times")
print("Least frequent note in Corpus appeared:", min(Recurrence), "time")
```

```
plt.figure(figsize=(18,3),facecolor="#97BACB")
bins = np.arange(0,(max(Recurrence)), 50)
plt.hist(Recurrence, bins=bins, color="#97BACB")
plt.axvline(x=100,color="#DBACC1")
plt.title("Frequency Distribution Of Notes In The Corpus")
plt.xlabel("Frequency Of Chords in Corpus")
plt.ylabel("Number Of Chords")
plt.show()
```