User Manual:

1. After installing the program with everything downloaded(as seen in the Installation Guide), the computer will ask you 4 questions in succession before generating your desired type of output-which depends on the corpus(text source) loaded.
2. *What order would you like to generate?*: The program has the ability to go up to 4th-order and the higher the order you choose the more “human-like” the output will be. Pick a number between 0 and 4 accordingly.
3. *How long would you like the output to be?*: Enter a number. This will determine the length of the output in words/notes.
4. *Would you like to choose the start?*: Yes or No question. If you don’t choose the start of the output the program will randomly generate it for you.
5. If you say yes, the program will ask for n starting words/notes. This will allow you to enter words you would like your output to begin with and the number of words, noted as n, will depend on the order you choose(i.e. If you choose 3rd order the program will ask for 2 starting words).
6. After generating the sentence, the program will ask you if you want to generate another sentence using the same start. If you say yes, the program will run with that same start again for the output. If you say no, the program will ask you if you want to generate again at all- meaning start over from Step 2- if you say no, the program will end and if you say yes you return to Step 2.

Installation Guide:

1. Ensure you have downloaded and installed Python 3, ensuring to add Python to the OS PATH during install.
2. Download and store both the obj file and the script in the same location
3. Download the necessary dependencies. In Command Line, type:
    a. ***pip install audiolazy***
    b. ***pip install midiutil***
4. In Command Line type ***cd (your file path)***. For example, ***cd C:\Users\jorda\Desktop***
5. In Command Line call the script with ***python musicGenerator.py*** (replace musicGenerator.py with twitterGenerator.py for twitter script, etc.)
6. For the music generator, the outputted MIDI file will appear as output.mid in the location the script is being run from

Developer Manual:

**Generating New Dictionaries with a Corpus**

1. The obj file contains pickle files (stored dictionaries) for generating text from zero to fourth order (0-6 for music). These stored dictionaries are generated from plain text corpora.
    a. For example, if the corpus were “the dog jumped, but it fell down” the generated pair dictionaries that need to be generated would be…
        i. oneWordPair: “the”, “dog”, ”jumped”, “, ” , “but”, …
        ii. twoWordPair: “the dog”, “dog jumped”, “jumped ,”, …
        iii. This continues until you have as many pairs as you want orders of operation(i.e. twoWordPair for second-order probability)
    b. The keys for these dictionaries follow the above pattern, their values are how often those words (or word pairs) appear in the corpus. These dictionaries can be generated using built-in methods
        i. Create a Generator object
        ii. Using the new object, call the processWords method and store in a variable
        iii. Next, create the dictionaries using the listToDict method which must be passed the variable used to the store the pre-processed data and the order its being generated for
            1. For example oneWordPair would be passed *listToDict(words, 1)*
    c. Now that all the word pair dictionaries have been generated, the word probability dictionaries must be generated. These dictionaries are a decimal between 0 and 1 that represent the probability that the last word in key follows the previous ones.
        i. For example, if the corpus were “the dog talk the dog walk” the probability dictionaries that need to be generated would be… **(Note there is no oneWordProb)**
            1. twoWordProb:
                a. “the dog” = 1
                b. “dog talk” = .5
                c. “talk the” = 1
                d. “dog walk” = .5
    d. These dictionaries can be generated using the buildProb method. buildProb must be passed the wordPair for which it is being generated, the wordPair below it, and the order of the lower wordPair.
        i. For example, to build threeWordProb the command would be *threeWordProb = buildProb(threeWordPair, twoWordPair, 2)*
    e. **BEFORE** being used, the wordPairs must be placed in an array named pairList. The probabilities must be placed in an array named probList.

**Saving/Loading Dictionaries**

1. You could have the script pre-process and create dictionaries every time it is run, however the process can be skipped by saving the dictionaries as pickle files.
2. To save a dictionary using the Generator object use the save_obj method. To save a dictionary the command would be *save_obj(dictionary name, "fileName")*
3. To load a dictionary use *load_obj("fileName")*
4. **MAKE SURE,** to save using a unique end handle (for example, all the twitter dictionaries have oneWordPairTwitter)

**Generating Using Dictionaries**

1. To generate an output sequence use the buildSequence method. buildSequence require five items to be passed
    a. order: which order the sequence should use to build the sequence (higher = more human-like)
    b. pairList: **See Above**
    c. probList: **See Above**
    d. start: The string the generator will use to start the sequence. Can be either randomly generated from the lower order pair **OR** input by the user.
        i. start for third order = random.choice(list(twoWordPair))
    e. length: the **MINIMUM** length for the output for **TEXT** and the **MAXIMUM** length for the output of music.
        i. for text, the output will end at any period AFTER the length or after 2*length

**Difference Between the Scripts**

1. There are scripts for each type of corpus (twitter, spanish, news, music). They all have the same base methods with only SLIGHT variation.
    a. Each has a unique processWords() method that uses different regular expression commands to get the text files into plain text to be imported. This is unique to each corpus we used and may need to be reprogrammed depending on the necessities of any new corpus.
    b. For **MUSIC** the length variable is the max length. For every other script length is the minimum number words, max being length*2. It tries to stop at a period between those two points.
        i. The pariList and probList in music goes up to the sixth degree. The others only go up to four.
        ii. **NOTE FOR THE MUSIC SCRIPT**: Any input of notes containing a #, like to indicate a sharp in music, is temporarily replaced with an "s" during pre-processing to avoid errors. s is then replaced by # before output is given to the user.

**Common Errors:**

1. Memory Error: The scripts, especially for language, are very memory-inefficient and the script may not be able to load to the dictionaries on a low-RAM device.
2. Key Error: If a key error comes up during buildProb() or listToDict() it usually means that the input corpus was not sufficiently pre-processed and may have anomalies (such as unnecessary spaces, unsupported characters, etc.)