PhonPlotter (v. 1.1)

User Guide

Ryan Gehrmann
Payap University
November 18, 2023

1 Introduction

PhonPlotter (PP) is a methodology designed to facilitate acoustic phonetic experiments for those with minimal training in data science and computer programming. This, it is hoped, will encourage and inspire students, especially students at the undergraduate or MA level, to undertake preliminary acoustic phonetic investigations, gain experience in designing and implementing such experiments, and ultimately serve as a springboard into more in-depth analysis.

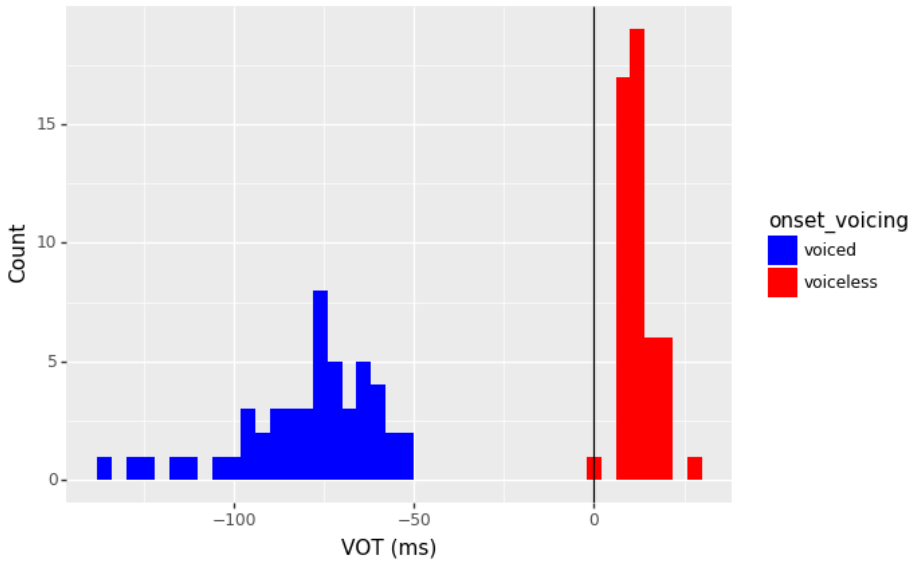
PP is designed around a folder holding audio files (the **audio folder**) and an Excel spreadsheet (the **lexical database**), which are linked by the names of the audio files in your audio folder. The lexical database has a column that holds the names of each audio file in your folder. The remaining columns hold phonological and categorical data that you are interested in exploring for your experiment.

For example, if I wanted to investigate differences of VOT in an onset stop voicing contrast, I might prepare a list of eight words as shown in Table 1 below. Here we have four words with syllables that begin with a voiceless stop /p t c k/ and four that begin with a voiced stop /b d ɟ g/. Because I am interested in exploring the differences in VOT between voiceless (vl) and voiced (vd) stops, I have added a column to my lexical database to hold this information for each word. I am also curious whether place of articulation of the stop will make a difference in VOT measurements, so I have added a column to hold information about place of articulation for onsets in each word as well.

Table 1

index	word	gloss	onset_voicing	place
1	paːŋ	palm (of hand)	voiceless	labial
2	baː	paddy rice	voiced	labial
3	ka**t**a**ː**m	crab	voiceless	alveolar
4	daː	duck	voiced	alveolar
5	caː	meat	voiceless	palatal
6	ɟaːl	casting net	voiced	palatal
7	kaːn	to chew	voiceless	velar
8	gaːl	forehead	voiced	velar

You can add as many columns as you want to your lexical database to suit the needs of your experiment. You will be able to reference these categorical variables in the data visualizations that PP generates. For example, here is a histogram showing differences in VOT (measured in milliseconds) for one speaker for the data in Table 1.



2 Project Design

2.1 Designing Around Syllables

The PP system is designed to analyze *one syllable per sound file*. The reasons for this are technical and will become clear when the system for segmenting and annotating in Praat is introduced below. If you plan to analyze a word list, you will have to think about this fact when designing your project.

For example, imagine I want to analyze the word list in Table 1. Except for number 3, each word is a monosyllable. For number 3, I will either need to (1) analyze just one of its two syllables as in Table 1, where I have bolded the final syllable or (2) make a duplicate of the audio file and re-label the index so that I have a separate audio file for each syllable of the word as in Table 2. If you go the Table 2 route, it is a good idea to add a column to the database specifying the place of the targeted syllable in the word (e.g. ultima vs. penult vs. antepenult). (More on the lexical database below).

Table 2

index	word	gloss	voicing	place	syllable
1	paːŋ	palm (of hand)	vl	labial	ultima
2	baː	paddy rice	vd	labial	ultima
3_1	**ka**taːm	crab	vl	alveolar	penult
3_2	ka**t**a:**m**	crab	vl	alveolar	ultima
4	daː	duck	vd	alveolar	ultima
5	caː	meat	vl	palatal	ultima
6	ɟaːl	casting net	vd	palatal	ultima
7	kaːn	to chew	vl	velar	ultima
8	gaːl	forehead	vd	velar	ultima

2.2 Syllable Shape

Ideally, you should stick to syllables of the shape CV(C) when using the PP system. The PP system is not designed for analyzing complex onsets. The system can track pitch and voice quality during a voiced onset and can measure VOT for onset stops, but it is not built to do both at the same time in a word like, for example, /klaːŋ/. In a word like this, you will have to decide whether to measure VOT in the /k/, vowel quality during the /aː/ or pitch and voice quality across /laːŋ/ - but you cannot do all of these at the same time.

3 Preliminaries

The instructions, tools and scripts used in PP v. 1.0 require you to have a Windows machine[1] with the following installed::

- Praat
- Python (via the *Anaconda* platform)

You do not need any prior knowledge or skills in using Praat and Python to use the PP methodology. You will be guided through what you need to know. That being said, these tools are of great utility for acoustic phonetic analysis, and it is hoped that you will be inspired to learn more about how to use them going forward. Think of this as an introduction to the kinds of things you can do.

[1] A version for Mac users is a desideratum for the future

Praat may be downloaded from the official Praat website. Note that Praat does not install like most programs. Rather, it remains a one-file executable that you need to store somewhere on your computer where you will be able to find it again in the future (I recommend Program Files).

If you are new to Python and do not know how to install or use it, that is ok. You should download Anaconda, which is a data science platform that takes care of Python installation and setup, and facilitates the management of the Python *packages* that you will need.

Once you have installed Anaconda Navigator, open it and go to the Home menu. Here, you may select from various Python editors. I suggest you install Spyder and use it for running code. You can open Python scripts (.py) in Spyder, edit and run them.

4 Python Basics

You only need to know a few basic Python concepts to interact with the Python scripts included in the PP system. A quick overview is presented here. If you need additional explanation or background, the internet is full of basic Python guides and advice. Simply type your question into Google and see what comes up or watch a YouTube video or two on the basics of Python.

- **Strings** are just ordered sequences of characters (e.g. words, sentences, etc…). They are always surrounded by quotations marks, either single ‘’ or double “” (you can use either).
    - For example, “yes” and ‘hello everyone!’ are strings but yes without quotation marks is not a string (that is a variable)
- **Variables** are sequences of characters that are used to refer to information. A string may be saved as a variable and used later. Most of what you do when you edit the Python scripts is assigning strings to variables that have been provided for you.
    - For example, in the first Python script that you will use, you are asked to give your project a name. You are presented with the variable **project_name** followed by the equal sign **=** and you are asked to provide a string with no spaces in it after the equal sign.
        - project_name = ‘my_project’
        - The line of code above means “assign the string ‘my_project’ to the variable project_name”.
        - The script will use the name of your project saved in that variable to do various things like name your new project folder and label your sound files (details below in Step 2)
        - You must not change the variable name to the left of the =
            - You must only edit the string to the right of the =
- **File Paths** (i.e. addresses showing where files or folders are saved on your hard drive) are saved as special kinds of strings in Python. They are surrounded by quotation marks just like a regular string, but they also must have a lower-case **r** to the left of the opening quotation mark.
    - my_folder = r‘C:\audio_data\project_audio’
    - The line of code above means “assign the file path C:\audio_data\project_audio to the variable my_folder”.
        - The script will remember where that folder is and use it to accomplish its task
    - my_file = r‘C:\audio_data\project_audio\file1.wav’
    - The line of code above means “assign the file path C:\audio_data\project_audio\file1.wav to the variable my_file”.
        - The script will remember where that file is and use it to accomplish its task
    - **HINT**: You can easily copy file paths from a Windows Explorer window. Simply hold the shift key on your keyboard and then right click on a folder or file. You will find an option “Copy as path” in the drop down menu that pops us. Click on “Copy as path” and the file path leading to that folder or file will be saved to you clipboard. You can then paste it in to your Python editor. Note that the file path will be surrounded by double quotations marks by default (e.g. “C:\my_path”).

- **Integers** are whole numbers. You should not put quotation marks around an integer. If you do, it will save the number as a string instead. For example, 3 is an integer which can be computed as a number, but '3' is a string holding the text symbol for the number 3. The string '3' cannot be computed.
- **Floats** are like integers, but for decimal numbers. Like integers, they should not be surrounded with quotation marks. For example. 3.1 is a float which can be computed as a number, but '3.1' is a string holding the text symbols that represent the number 3.1. The string '3.1' cannot be computed.
- **Lists** can hold several items in a set order. They are always surrounded by square brackets [] and items in the list are separated by commas.
    - **my_list** = ['string1', 'string2', 'string3']
    - The line of code above means "assign the list ['string1', 'string2', 'string3'] to the variable my_list".
        - *Note: You can also hold integers, floats, other lists, etc… in a list.*
- **Dictionaries** hold several pairs of items. They are always surrounded by curly brackets {}. The two items in a pair are separated by a colon : and each pair of items in the dictionary is separated by a comma. The item to the left of the colon is called the "key" and the item to the right is called the "value". You could make an actual bilingual dictionary like this, where the source language is the key and the target language is the value (e.g. {'dog' : 'perro', 'cat' : gato'}), but in PP, we use it for things like assigning a color to a category for a plot (e.g. {'vd' : 'blue', 'vl' : 'red'})
    - **my_dictionary** = {'dog' : 'mammal', 'frog' : 'amphibian', 'snake' : 'lizard'}
    - The line of code above means "assign the dictionary {'dog' : 'mammal', 'frog' : 'amphibian', 'snake' : 'lizard'} to the variable my_dictionary".
        - Note that dictionary keys are usually strings or integers, while dictionary values can be anything at all (strings, lists, integers, floats, other dictionaries, etc…).

5 Step 1: Formatting Your Data

Formatting your data for use with PP is not difficult, but it is extremely important that you pay attention to details and get this part right. The rest of the process depends on this, and you must build a solid foundation before you begin.

In this step, you will develop the following:

- **A folder** containing your .wav audio files
- **An Excel database** that references those audio files and hold phonological information about the contents of each file that will be relevant to your analysis

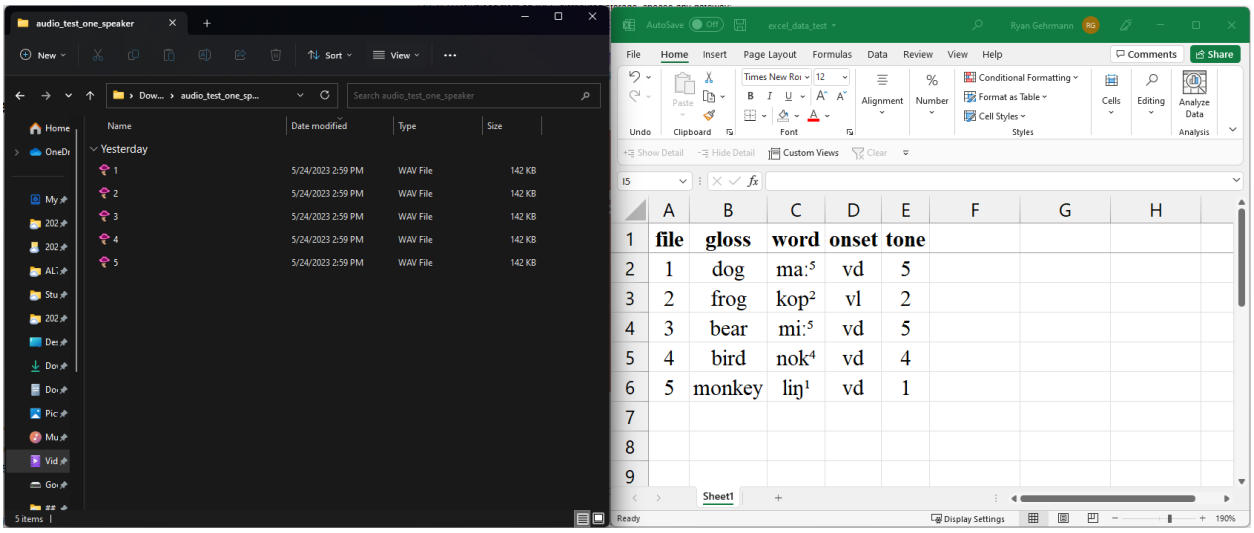
---

It is extremely important that the file names in your audio folder match the file names in your Excel database *exactly*

5.1 Excel Database

You should have a predetermined list of items of interest to investigate using PP. For simplicity's sake, we will just refer to this list as *the wordlist* here, since in most cases, that list will be a list of words. You will keep track of this wordlist in an Excel Spreadsheet. Every item in your wordlist should have its own row in the database, and you should have one recording corresponding to every row in the database. Best practice is to give a unique index number to each row and then label the corresponding recordings using that number. In the end, you should have a column named 'file' in your database, which *corresponds to the names of the audio files that you recorded **exactly***. There should be a one-to-one relationship between the file names in your 'file' column and the files in your audio folder.

The following screenshots provide an example with a small, five-item dataset.



	A	B	C	D	E
1	**file**	**gloss**	**word**	**onset**	**tone**
2	1	dog	ma:^{5}	vd	5
3	2	frog	kop^{2}	vl	2
4	3	bear	mi:^{5}	vd	5
5	4	bird	nok^{4}	vd	4
6	5	monkey	liŋ^{1}	vd	1

Your Excel database must include a column called ‘file’ that holds file names corresponding to the file names in your audio folder (**you do not need to include the .wav extension for each file**). Every other column is optional, and you can add as many columns as you find useful for your analysis. You will want to have separate columns for each of the categories that you want to test. In this case, we have columns for onset voicing and tone category, but you can make a column for anything at all. Some ideas: place of articulation of onset, phonation type of coda, loan-word status (native vs. loan), categories of morphological inflection, position in utterance, etc…

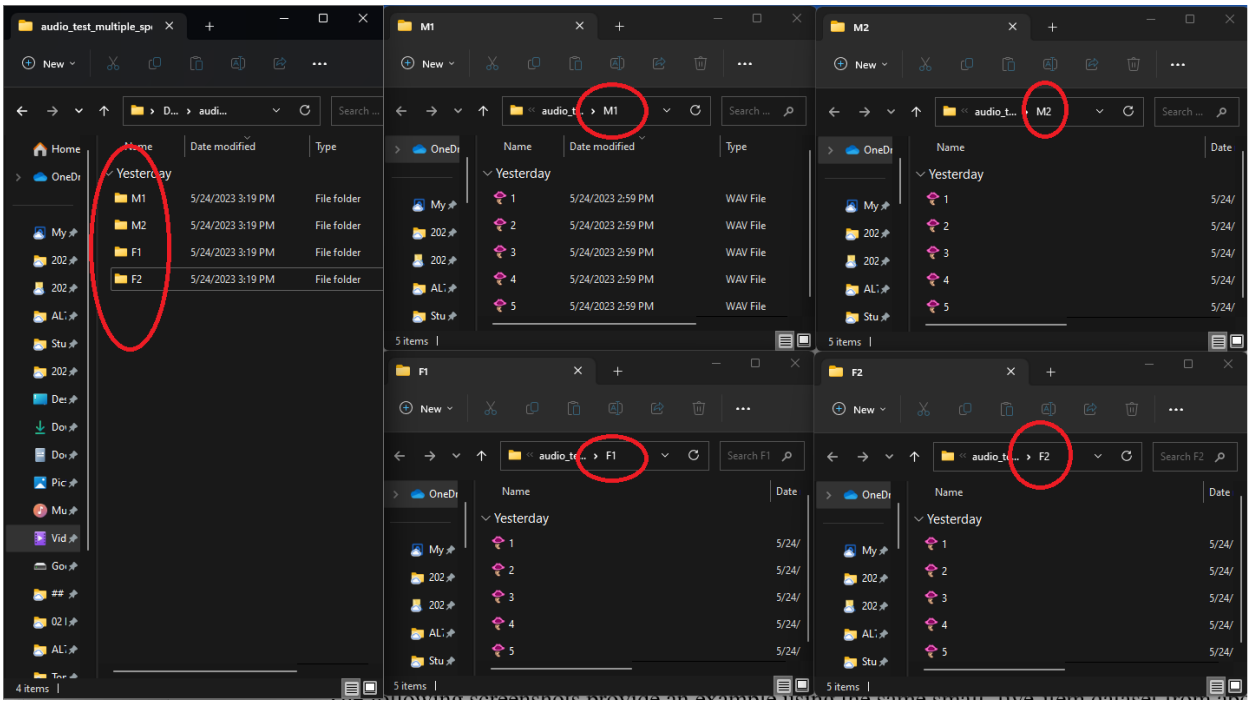
Be sure to save this file with a .xlsx file extension.

5.2 Audio Folder

If you have recorded only one speaker, then you can simply place your .wav audio files in a folder, make sure their file names correspond to the file names listed under the ‘file’ column in your Excel database, and you are done.

However, if you recorded more than one speaker, you need to take an additional step. Do not put any audio files in your audio folder yet. Instead, make a separate folder for each speaker, and name each folder in correspondence with the name or label that you want to use for that speaker in the analysis (e.g. F1, F2, M1, M2 labels). Then, put each speaker’s audio data in their respective folder. The .wav audio file names should correspond exactly to the file names listed in your Excel database. They should not reference the speaker at this point.

The following screenshots provide an example using the same small, five-item dataset from above. In this case, we have four speakers, and in each speaker’s folder, we find five audio files corresponding to the five items in the Excel database.



IMPORTANT NOTE: It is okay if there are some audio files missing in your audio folder. Maybe you were not able to record every item for one speaker, or maybe some of your recordings are un-usable and need to be thrown out. That is ok. However, ***it is not okay*** if there are audio files in your audio folder that do not occur in your Excel database. This will cause an error. In other words, if files 01.wav, 02.wav and 03.wav are found in your Excel database but 03.wav is missing from your audio folder – **no problem**. However, if 01.wav, 02.wav and 03.wav are found in your audio folder but 03.wav is missing from your Excel database – **you have a problem**.

ANOTHER IMPORTANT NOTE: If you are using a numerical indexing system, you may well run into trouble if you have leading zeroes in your file names (e.g. 0035.wav). There are two ways to address this: (1) in your Excel database, make sure that all of the file names under the 'file' column have a file extension (e.g. make sure that all of them have .wav at the end as in 0035.wav, not 0035 on its own), or (2) remove the leading zeroes from all audio file names and from the file names under the 'file' column in your Excel database.

Once you have carefully checked that your Excel database and audio folder are properly formatted, you can move on to Step 2.

6 Step 2: Starting the Project Folder

In this step, you will create a project folder using a Python script. You should only move on to this step after your data has all been properly formatted according to the guidance in the previous step

The project folder will be created on the root of your C:\ drive. The reason for this is that you will need to run some Praat scripts in the following steps. Praat scripts do not work if there are any space characters in the file path leading to a file. The closer the project folder is to the root of the drive, the less chance of encountering an error because of this issue. (You can move the folder later, after the Praat scripts have been run).

You will use the Python script pp_project_folder.py to create your project folder. The script will ask you for just five inputs at the top.

- project_name: A name for the project with no spaces (including _ between words is ok)
- my_excel: File path to .xlsx file holding phonological data
- multiple_speakers: Did you record more than one speaker? (yes or no)

- tokens: how many tokens per .wav file (answer with an integer, e.g. 3)
- my_audio: File path to folder holding .wav files

project_name: Provide a string holding the name of your project. There can be no spaces in the string because this project name will become (1) the name of your project folder and (2) a prefix on all your .wav files. If there were a space in your project_name string, this would cause an error when running Praat scripts.

my_excel: Provide a string holding the file path leading to the Excel database that holds your file names and accompanying phonological data. This Excel file will be copied and placed in your new project folder with the file name lexical_database. The contents of your lexical_database will also be updated in accordance with the changes made to the audio files (see below). Your original Excel file will remain unaltered. Note that the file path that you provide must lead to a .xlsx Excel file, or else you will get an error.

multiple_speakers: If you recorded your word list with more than one speaker, you should provide 'yes' for this variable. Otherwise, provide 'no'.

tokens: Each sound file can be annotated just once. Consequently, if each of your sound files contains more than one token, we will need to produce multiple copies of each sound file. That way, you can annotate each token in a separate sound file. Provide an integer here (not a string, i.e. do not use ' ' or " "), to let the script know how many tokens you have per sound file.

my_audio: Provide a string holding the file path leading to the folder on your hard drive that holds your .wav audio files. Your audio files will be copied, relabeled and multiplied as needed according to your inputs for the multiple_speakers and tokens variables.

If you indicate 'yes' for multiple_speakers, the script will look for sub-folders in your my_audio folder. These sub-folders should be labeled with the names or codes assigned to each of your speakers (F1, F2, M1, M2, etc…). Each audio file will be re-labeled as follows: project_speaker_item_token. Project is taken from your project_name variable. Speaker is taken from the name of the sub-folder in which the audio file was found. Item is taken from the name of the .wav audio file itself. Token is added in accordance with the number of tokens that you indicated for the tokens variable (e.g. if you indicate 3, then the script will make three exact copies of the original audio file, labeled project_speaker_item_1, project_speaker_item_2 and project_speaker_item_3).

If you indicate 'no' for multiple_speakers, the script will not look for sub-folders in your my_audio folder. Instead, it will look for your .wav audio files in the main folder. Each audio file will be relabeled as follows: project_item_token. Project is taken from your project_name variable. Item is taken from the name of the .wav audio file itself. Token is added in accordance with the number of tokens that you indicated for the tokens variable (e.g. if you indicate 3, then the script will make three exact copies of the original audio file, labeled project_item_1, project_item_2 and project_item_3).

Your relabeled and potentially multiplied audio files (unless tokens = 1) will be copied and moved to an 'audio' sub-folder in your new project folder. Your original audio files will not be altered and will remain as they were in your my_audio folder.

Please ensure that the string that you supply for my_audio leads to a real folder on your hard drive to avoid an error. In addition, if you indicate 'yes' for multiple_speakers, please ensure that the .wav files are organized into sub-folders in your my_audio folder and that those sub-folders are labeled according to speakers' names or codes. On the other hand, if you indicate 'no' for multiple_speakers, please ensure that the .wav files are not found in any sub-folders, but rather directly in the my_audio folder.

7 Stop and Try it for Yourself (Part 1)

Let's stop now and try out Steps 1 and 2 on some pre-prepared test data. The folder contains recordings from two speakers of the Triang language: one male (M3) and one female (F1). Their recordings are found in the triang_test_audio folder in sub-folders appropriately labeled M3 and F1. Each sub-folder holds the same number of sound files, labeled to match the files listed in the 'file' column of the Excel database. For this project, we have one token per audio file, but we have two recordings per word list item. So, for example, for item number 1, under M3

with have two recordings, each suffixed with a token number: 1_1.wav and 1_2.wav. These are two different recordings of the speaker saying item 1. We find the same under the F1 folder.

Note that for each of these recordings, the word is couched in a carrier phrase:

ʔal	ka:	ʔi:	_____	ka:	maj
1SG	IRR	say	_____	for	2SG

"I will say _____ for you."

Your Task: You are given properly formatted data. Follow Step 2 and start a project folder using this data.

8 Step 3: Creating TextGrids

In this step, you will segment and annotate your audio files in Praat. You must do this in a particular way when using the PP system.

8.1 Create TextGrids

TextGrid files save annotation and segmentation information that Praat can read. We must create a properly formatted TextGrid file for each of the audio files in your project audio folder. This is easily done using the Praat script pp_textgrid_creator.praat. Download this script and open Praat. In the Praat Objects window, go to the Praat menu > Open Praat script… Navigate to the TextGrid creator script and open it. The script will open in a new window. (**NOTE**: it is a good idea to save this and other scripts that you use in your project folder. You can make a new sub-folder called 'scripts').

Once the script is open, press control + r to run the script. A pop-up dialogue box will appear asking you for two file paths. The first one, soundDir, is the directory holding your audio files (i.e., your project audio folder). The second one, tgDir, is the directory where you want Praat to create TextGrid files. You should put the file path to your project audio folder here as well, since we have no reason to separate them into separate folders. Once you are ready, click "Ok" and the script will create a TextGrid file for every audio file in your project audio folder.

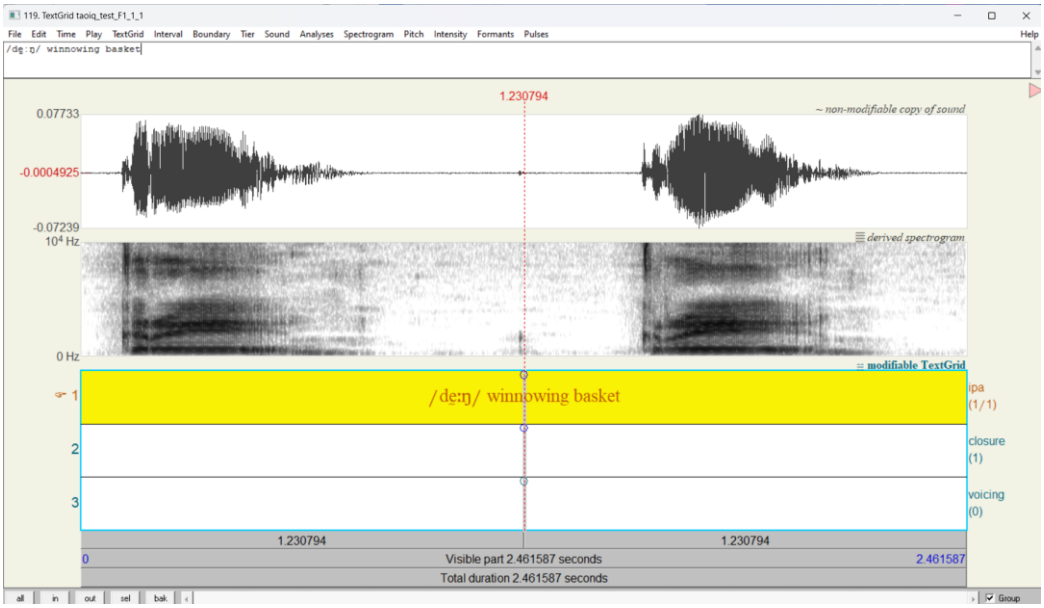
NOTE: **Do not put a final backslash** on the file path address, or the script will not work.

8.2 Add Word & Gloss to TextGrid (Optional)

You can fill in phonological information to Tier 1 of your TextGrids if you like (this is entirely optional). To do this, you must have a column labeled 'word' and a column labeled 'gloss' in your lexical database. You can put whatever info you like in these columns, but the script will add text to Tier 1 of your text grids in the format:

/word/ gloss

The screenshot below demonstrates what this will look like in Praat.



If you want to fill in this information on Tier 1, you will need to run the Python script pp_add_textgrid_word_data.py. Download the script, open it in your Python editor, edit the **lexical_database** variable, adding a file path string to lexical_database.xlsx in your project folder, and edit the **audio_folder** variable, adding a file path string to your project audio folder. Run the script and the TextGrids in your project audio folder will be updated based on the 'word' and 'gloss' columns in your database. Unedited copies of your TextGrids will be stored in your project folder in a folder called textgrids_unedited. You can get rid of these later, but it's good to keep them for a bit, in case something has gone wrong.

Note that if you have multiple speakers as sub-folders in your audio folder, you will need to run this script multiple times; once for each speaker sub-folder.

Note that you will get an error message if you run this script and (1) the lexical_database.xlsx file and/or the project audio folder that you specify do not exist on your hard drive, (2) your lexical_database.xlsx file does not include 'word' and 'gloss' columns, or (3) your project audio folder does not contain .TextGrid files.

Note that if you run this script more than once, a new textgrids_unedited folder will be created each time. The old ones will not be overwritten. Instead, an underscore _ will be appended to the end of the folder name each time to keep them separate.

9 Step 4: Segmenting and Annotating TextGrids:

You will now go through each TextGrid file to segment and annotate your data. Remember, you must segment and annotate ***just one syllable per file***.

It is very tedious and time consuming to open each sound file and then save and close it when you are done. To help with this, a script is provided: pp_textgrid_editor.praat. You just tell the script where your audio files are, and it will cycle through them for you. It will open a file and allow you to edit it. After that, you just press a button in a pop-up window, and the script will save the TextGrid file with your annotations and open the next file for you.

Open the pp_textgrid_editor.praat script using the same procedure outlined above. Run the script and a similar pop-up dialogue box will appear. Paste in the file path to your project audio folder in both fields but **be sure to include a final backslash on the file path** this time. Click Ok and the script will open the first file for you.

Your job is to annotate **one to three things on Tier 2** and **just one thing on Tier 3**. On Tier 2, create segment boundaries around the closure portion of the onset (where applicable), the open vocalic portion of the rime and any coda closure/constriction (where applicable) (see video guide for details). On Tier 3, which is a point tier, all you mark is the point in time where voicing commences. Use the following labels when annotating

- cl – Tier 2 – the onset closure/constriction (cl = closure)

- op – Tier 2 – the vocalic nucleus of the rime (op = open)
- cd – Tier 2 – the coda closure/constriction (cd = coda)
- ov – Tier 3 – the point in time where voicing begins (ov = onset of voicing)

Note: If you have a very large number of sound files to annotate, it can be useful to break them up into batches. You can do this by creating new sub-folders in your project audio folder. For example, you might have 900 sound files to annotate. That is way too many for one day! You can split them up into folders with 100 files and do 100 files per day for 9 days. That is much more manageable. **But remember two things! (1) there are two files for every one recording (**a .wav and a .TextGrid**) and (2) you need to put your files back in their original folder once you are done.**

Hint: If you click on the Praat Objects window, the dialogue box with the button to advance the script will also be brought to the front. This is a quick way to find the box when you are ready to go to the next file.

10 Step 5: Extract Acoustic Measurements

Now that you have marked up your TextGrids, it is time to extract acoustic measurements from your .wav files. You will use the Praat script praatSauce.praat to do this. This is a very complex script that comes in a package with a number of other scripts on which it relies. As a result, you must download this entire folder in order use praatSauce.praat. Once you have downloaded it, open the folder, go into the "src" sub-folder and you will find praatSauce.praat there. Open it in Praat and run it.

The praatSauce interface box is very large. If your Windows display settings are zoomed out too far, you may not be able to see the bottom of the box where the Ok button is. If so, press the windows key, type in *display settings*, open the display settings window, find the *scale and layout* section, under the *Scale* option, reduce the zoom percentage until you can see the entire praatSauce interface window. (Note, you may have to close the praatSauce interface window and re-open it in order to get the window resized).

praatSauce has a complicated user interface, but you only need to edit a few fields. Only edit the following fields:

inputdir: supply file path to you project audio folder (no trailing backslash)
textgriddir: supply file path to you project audio folder (no trailing backslash)
outputdir: supply file path to the top level of your project folder (no trailing backslash)
point tier labels: ov

Click Ok then click continue four times through the next few pop-up windows without editing anything. Praat will begin a complicated series of processes that can take quite a bit of time depending on how many files you are analyzing. Just let it run until a pop-up window informs you that the analysis is complete.

Once the script has finished running, it will place a new file in your project folder: PraatSauceOutput.txt. This is a database of acoustic measurements in csv format (comma separated values). You do not need to do anything with this file, just leave it where it is.

11 Step 6: Checking Annotations

You may have made a few mistakes in your segmentation and annotation. You should now run the Python script pp_annotation_checker.py, which is designed to identify and report on typos, aberrantly long or short segments and missing annotations that are required. Open the script in your Python editor and provide info for just tone one variable:

- **project_folder** – file path to your project folder

After you run the script, a new file, annotation_error_report.txt, will appear in your project folder. Open this file and it will either list some probably/possible errors in your segmentation and annotation, or it will just say "No errors to report. Good work!". In the latter case, your work is done and you can move on. However, if the script found some problems, you should go back and manually fix your TextGrids. Remember to save them after you edit them. If you just close the TextGrid, your edits will not be saved.

Once you have made the required edits to your TextGrids, you will need to re-run PraatSauce (follow the previous step). Once you have done that, re-run pp_annoation_checker.py to see if you cleared up the issues.

Note: The checker script is designed to find suspiciously long or short segments on Tier 2. It determines this statistically. However, just because a segment is a statistical outlier in terms of its duration, that does not mean that you marked it wrong. If you check the file and the segmentation is correct, you should ignore the error.

12 Step 7: Plot VOT Distribution in a Histogram and in a Box & Whisker Plot

You are now ready to begin producing data visualizations. If you are interested in plotting VOT, you can do that using the Python script pp_plot_vot.py. This script will produce two plots, a histogram and a box & whisker plot, each showing a different representation of the distribution of VOT measurements. Open this script in your Python editor, edit the user inputs at the top of the script (see information on each variable below) and then run the script (see instructions below).

12.1 Editing User Inputs

The plotting scripts require more complicated inputs than the scripts that we have used previously. Let's go through each of the variables.

- **project_folder**: file path string to your project folder. You must provide this.
- **color_column:** you can use colors to differentiate a category in the plot (e.g. onset voicing, place of articulation, speaker, etc…). Simply give a string with the column header of a column in your lexical database that holds the relevant categories.
    - If you do not wish to use color_column, simply provide an empty string ''
    - **NOTE:** May be freely combined with facet_column for representing different kinds of categories in the same plot.
- **color_dict:** if you want to specify colors for each of the color categories in the color column, you can supply a dictionary here in the format {'category1':'color', 'category2':'color'}. Be sure to list every category in your color column or else you will get an error.
    - If you do not wish to use color_dict, simply provide an empty string ''
- **facet_column**: this works just like color_column, except that instead of giving each category its own color, each category gets its own sub-plot within the larger plot. Simply give a string with the column header of a column in your lexical database that holds the relevant categories.
    - If you do not wish to use facet_column, simply provide an empty string ''
    - **NOTE:** May be freely combined with color_column for representing different kinds of categories in the same plot.

    - **HINT**: This can be helpful to clean up the data presentation when your VOT categories are overlapping in the histogram.
- **filter_dict:** this is a useful tool to filter for only the categories that you are interested in. You can filter any column for only the categories that you want to keep. Simply provide a dictionary in the format {‘column1’:[‘items’,’to’,’keep’], ‘column2’:[‘items’,’to’,’keep’]}. For example, if you have three speakers in your database, but you just want to plot VOT for speaker1, supply a dictionary like this: {‘speaker’:[‘speaker1’]}. The filter will disregard every row in your lexical_database except for those rows where ‘speaker1’ is listed under the ‘speaker’ column.
    - If you do not wish to use filter_column, simply provide an empty string ‘’
- **bin_width:** the histogram is broken up into vertical bars of equal width. Each VOT measurement that falls into the range represented by that bar adds to that bar’s height. Ranges with more measurements get taller. In terms of VOT, the range of a bar might be 2 ms, 4 ms, 10 ms, etc… The script guesses what a good bin width would be based on your data, but it often guesses poorly. You can experiment with the bin_width to find an optimum presentation of the distribution of VOT measurements.
    - **NOTE: be sure to provide an integer, not a string here**
    - If you do not wish to use bin_width, simply provide an empty string ‘’

12.2 Running the Script

Once you have specified these variables, you can run the script (the default keyboard shortcut is F5 or you can press the ‘play’ button at the top). Check the “Console” at the bottom right of Spyder’s interface. If the script was able to run successfully, it should something look like the screenshot below.

```
In [1]: runfile('G:/My Drive/01 Projects/PhonPlotter/PhonPlotter Scripts/pp_plot_vot.py',
wdir='G:/My Drive/01 Projects/PhonPlotter/PhonPlotter Scripts')
C:\Users\ryang\anaconda3\lib\site-packages\plotnine\ggplot.py:718: PlotnineWarning: Saving
6.4 x 4.8 in image.
C:\Users\ryang\anaconda3\lib\site-packages\plotnine\ggplot.py:719: PlotnineWarning:
Filename: C:\triang_voicing\export\vot_histogram
C:\Users\ryang\anaconda3\lib\site-packages\plotnine\ggplot.py:718: PlotnineWarning: Saving
6.4 x 4.8 in image.
C:\Users\ryang\anaconda3\lib\site-packages\plotnine\ggplot.py:719: PlotnineWarning:
Filename: C:\triang_voicing\export\vot_box

In [2]:
```

If the script was unable to run because it hit an error, you will see an error message in the Console. For example, if there is a problem with your file path (e.g. a typo or the indicated folder doesn’t exist), you will get the error. Follow the guidance in the error message and try to fix the problem. If you cannot, just ask for help; it may be that you have encountered a bug and you will not be able to fix it yourself.

```
In [4]: runfile('G:/My Drive/01 Projects/PhonPlotter/PhonPlotter Scripts/pp_plot_vot.py',
wdir='G:/My Drive/01 Projects/PhonPlotter/PhonPlotter Scripts')
Traceback (most recent call last):

  File ~\anaconda3\lib\site-packages\spyder_kernels\py3compat.py:356 in compat_exec
    exec(code, globals, locals)

  File g:\my drive\01 projects\phonplotter\phonplotter scripts\pp_plot_vot.py:237
    raise ValueError(msg1 + msg2)

ValueError: project_folder must be a valid file path on your hard drive. Please check.
Hint: be sure to format the file path as r'FILEPATH'


In [5]:
```

If your plots were successfully generated and exported, you will find them in your project folder under the "export" folder.

13 Step 8: Plot Pitch, Voice Quality or Vowel Quality Over Time Using a Line Plot

If you would like to investigate pitch, voice quality or vowel quality over time, you can use the script pp_line_plot.py for this. Open this script in your Python editor.

13.1 Editing User Inputs

The plotting scripts require more complicated inputs than the scripts that we have used previously. Let's go through each of the variables.

- **project_folder**: file path string to your project folder. You must provide this.
- **y_data:** this script is designed to plot various different acoustic measurements over time. As a result, you must specify here which measurement you are interested in. Provide one of the following strings:
    - **'F0'**
        - fundamental frequency – correlates with vocal pitch
    - **'F1'**
        - frequency of the first formant – correlates with tongue height (greater F1, lower vowel)
    - **'F2'**
        - frequency of the second formant – correlates with tongue backness (greater F2, fronter vowel)
    - **'H1H2'**
        - *H1-*H2 (or Spectral Tilt) is the difference in the amplitude of the first and second harmonics – correlates with voice quality (greater *H1-*H2, laxer voice quality)
    - **'CPP'**
        - Cepstral Peak Prominence is a measurement of periodicity in the speech signal (i.e. regularity of vocal fold vibration) – correlates with voice quality (lower CPP, less modal voice quality (laxer or tenser))
- **y_unit:** if you have chosen one of the three y_data options that are frequencies (F0, F1, F2), you must choose to present the results in one of two units: Hertz ('hz') or Semitones ('st'). If you chose Spectral Tilt or CPP, these are measured in Decibels only, and so you do not need to choose.
- **normalize_y:** Normalizing a list of data points simply means to rescale it to fit into a different range. A typical way to do this is to compute the mean (average) of the list of data points, and then redefine every data point in the list relative to the mean. For example, if my data points were 1 2 3 4 5, the mean would be calculated as 3, and every n in the list could be recalculated as n-3. As a result, the data would be transformed (normalized) to -2 -1 0 1 2. Now imagine that those numbers were measurements of F0 for an

adulat man and imagine that we had another list of F0 measurements for a child, which were higher numbers because of the child's higher-pitched voice. We could not compare the two voices meaningfully without first normalizing the lists of data points. So the child's F0 measurements, 3 4 5 6 7, could be normalized to -2 -1 0 1 2, and even though the two speakers have different mean F0 values, we can directly compare the effect of things like onset phonation, coda phonation or tone category on each speaker's realization of pitch, because we are working on the same scale.
    - If you respond 'yes' to normalize_y, all measurements in your y_data column will be normalized relative to the mean measurement value.
    - If you respond 'no', nothing will be done to the measurements
- **normalize_y_by:** In the scenario sketched out above, we would not want all F0 measurements normalized to the mean of all speakers. Instead, we would want each individual speaker's F0 measurements normalized based on each individual speaker's mean F0 measurement. To do this, we would simply need to provide a column in our lexical database specifying the speaker for each sound file. We could then supply the column header of that column here for normalize_y_by.
    - If you supply a column header as a string, the script will normalize all y_data points relative to the categories specified under that column (e.g. speaker, tone, etc…)
    - If you do not want to do this, simply supply an empty string ''
- **include_cl:** Pitch and voice quality can be measured during voiced onsets.
    - Supply 'yes' here if you would like to include measurements of pitch and voice quality during voiced onsets (i.e., 'cl' segments annotated in Praat) in your plot.
    - Otherwise, supply 'no'
    - **NOTE:** if you are measuring F1 or F2, you are not interested including measurements of the onset, because F1 and F2 are relevant to vowel quality.
- **include_cd:** Pitch and voice quality can be measured during voiced codas.
    - Supply 'yes' here if you would like to include measurements of pitch and voice quality during voiced codas (i.e., 'cd' segments annotated in Praat) in your plot.
    - Otherwise, supply 'no'
    - **NOTE:** if you are measuring F1 or F2, you are not interested including measurements of the coda, because F1 and F2 are relevant to vowel quality.
- **normalize_x:** The x-axis in your line plot will show time. It is often useful to rescale time for each syllable, so that the results can be compared directly, regardless of how quickly or slowly each word was produced.
    - Supply 'yes' here if you would like to normalize time. The rime (vowel + coda (where applicable)) will be rescaled proportionally to fit between 0 and 1 for all syllables.
    - Otherwise, supply 'no'
- **cd_time:** If you have responded 'yes' for normalize_x, you can specify what proportion of the rime to fit vowels into and what proportion to fit codas into for word that have codas. Supply a number between 0 and 1 here, which will serve as the end of the vowel and beginning of the coda. For example, if you supply 0.5, for any word with a coda (i.e. files that have a 'cd' segment annotated), the vowel will be rescaled to fit between 0 and 0.5 and the coda will be rescaled to fit between 0.5 and 1. For any word without an annotated coda, the open syllable vowel will be rescaled to fit between 0 and 1.
    - Supply a float between 0 and 1 to serve as the transition point between the vowel and the coda
    - If you do not wish to do this, simply supply an empty string ''
    - **NOTE:** If you ask the script to **include_cd** and to **normalize_x**, but do not provide a float for **cd_time**, the script will disregard the 'op' and 'cd' labels in your Praat annotation and just combine them into 'rime' and rescale the whole thing between 0 and 1. This is good for measuring pitch and voice quality across the entire rime.
- **cl_time:** If you have chosen to normalize time (normalize_x = 'yes'), then you can also normalize the 'cl' closure segment of the onset to a specific time. Time is rescaled to 0 = vowel onset, so the 'cl' segment will

be in negative time. Simply supply a negative float (e.g. -0.2) and all 'cl' segments will be normalized to between -0.2 and 0.
    - Supply a float less than 0 (i.e. a negative number) to serve as the onset for normalized 'cl' segments
    - If you do not wish to do this, simply supply an empty string ''
    - **NOTE:** If you choose normalize_x and include_cl, but do not supply cl_time, the script will automatically scale all 'cl' segments to between -0.3 and 0.
- **time_zoom:** You can zoom in on a specific interval of time in your plot if you like. This will not effect the calculations involved in determining outlier values, normalizing y-axis values or applying regression techniques to the y-values, it simply "zooms in" on a specific part of the plot.
    - Supply a list containing two numbers: [start, end].
    - **NOTE**: This will work for normalized time or regular time. Just be aware that you need to think in terms of proportions for normalized time and in terms of milliseconds for regular time.
- **color_column:** you can use colors to differentiate a category in the plot (e.g. onset voicing, place of articulation, speaker, etc…).
    - Supply a string with the column header of a column in your lexical database that holds the relevant categories.
    - If you do not wish to use color_column, simply provide an empty string ''
    - **NOTE:** May be freely combined with line_column and facet_column for representing different kinds of categories in the same plot.
- **color_dict:** if you want to specify colors for each of the color categories in the color column, you can supply a dictionary here in the format {'category1':'color', 'category2':'color'}. Be sure to list every category in your color column or else you will get an error.
    - If you do not wish to use color_dict, simply provide an empty string ''
- **line_column**: just like for color_column (see above), you can use different line types (solid, dashed, dotted, etc…) to differentiate categories in the plot.
    - Supply a string with the column header of a column in your lexical database that holds relevant categories
    - If you do not wish to use color_column, simply provide an empty string ''
    - **NOTE:** May be freely combined with color_column and facet_column for representing different kinds of categories in the same plot.
- **facet_column**: this works just like color_column and line_column, except that instead of giving each category its own color or line type, each category gets its own sub-plot within the larger plot.
    - Supply a string with the column header of a column in your lexical database that holds the relevant categories.
    - If you do not wish to use facet_column, simply provide an empty string ''
    - **NOTE:** May be freely combined with color_column and line_column for representing different kinds of categories in the same plot.
- **filter_dict:** this is a useful tool to filter for only the categories that you are interested in. You can filter any column for only the categories that you want to keep. Simply provide a dictionary in the format {'column1':['items','to','keep'], 'column2':['items','to','keep']}. For example, if you have three speakers in your database, but you just want to plot VOT for speaker1, supply a dictionary like this: {'speaker':['speaker1']}. The filter will disregard every row in your lexical_database except for those rows where 'speaker1' is listed under the 'speaker' column.
    - If you do not wish to use filter_column, simply provide an empty string ''
    - **NOTE:** Unlike zoom_time (see above), using filter_dict will affect the calculation of outliers, normalization and regression, because all of rows that are filtered out are filtered before any of those calculations are made.

13.2 Running the Script

Once you have specified these variables, you can run the script (the default keyboard shortcut is F5 or you can press the 'play' button at the top). Check the "Console" at the bottom right of Spyder's interface. If the script was able to run successfully, it should something look like the screenshot below.

```
In [1]: runfile('G:/My Drive/01 Projects/PhonPlotter/PhonPlotter Scripts/pp_plot_vot.py',
wdir='G:/My Drive/01 Projects/PhonPlotter/PhonPlotter Scripts')
C:\Users\ryang\anaconda3\lib\site-packages\plotnine\ggplot.py:718: PlotnineWarning: Saving
6.4 x 4.8 in image.
C:\Users\ryang\anaconda3\lib\site-packages\plotnine\ggplot.py:719: PlotnineWarning:
Filename: C:\triang_voicing\export\vot_plot___

In [2]:
```

If the script was unable to run because it hit an error, you will see an error message in the Console. For example, if there is a problem with your file path (e.g. a typo or the indicated folder doesn't exist), you will get the error. Follow the guidance in the error message and try to fix the problem. If you cannot, just ask for help; it may be that you have encountered a bug and you will not be able to fix it yourself.

```
In [4]: runfile('G:/My Drive/01 Projects/PhonPlotter/PhonPlotter Scripts/pp_plot_vot.py',
wdir='G:/My Drive/01 Projects/PhonPlotter/PhonPlotter Scripts')
Traceback (most recent call last):

  File ~\anaconda3\lib\site-packages\spyder_kernels\py3compat.py:356 in compat_exec
    exec(code, globals, locals)

  File g:\my drive\01 projects\phonplotter\phonplotter scripts\pp_plot_vot.py:237
    raise ValueError(msg1 + msg2)

ValueError: project_folder must be a valid file path on your hard drive. Please check.
Hint: be sure to format the file path as r'FILEPATH'


In [5]:
```

If your plot was successfully generated and exported, you will find it your project folder under the "export" folder.

14 Summary & Outlook

The PhonPlotter system is designed to equip and enable anyone who is interested in acoustic phonetic analysis to undertake an acoustic analysis project of limited scope. What PhonPlotter can do is only a taste of what is possible, however. Users are encouraged to deconstruct the PhonPlotter system and its associated scripts to figure out how they work. For those who wish to edit the scripts to suit their own purposes, but do not have the Python coding skills required, there is ample training materials on the basics of Python available online. Users are encouraged to learn specifically about Pandas for holding the relevant data and Plotnine for producing plots. Many linguists prefer to use R instead of Python for this kind of work, and there are many wonderful materials available on that topic as well.

If users find this first version of PhonPlotter useful, more versions with increased functionality will be developed. Please do get in touch (ryan_gehrmann@gmail.com) with feedback, especially where it relates to…

- usability / comprehensibility of the training materials
- bugs and things that do not work as they should

- the technical implementation of the system from a programming or statistical perspective
- recommendations for expanded functionality