# Florida Price Level Index (FPLI) Minimum Commute Calculator User Guide

Table of Contents

[Florida Price Level Index (FPLI) Minimum Commute Calculator User Guide 1](#_Toc64314941)

[Introduction 1](#_Toc64314942)

[Github 2](#_Toc64314943)

[Installation 2](#_Toc64314944)

[Process 6](#_Toc64314945)

[Usage 8](#_Toc64314946)

[General Usage 8](#_Toc64314947)

[Input Variables for minimum\_commute\_calculator.calculator() 10](#_Toc64314948)

[Content 14](#_Toc64314949)

[Requirements 15](#_Toc64314950)

[Packaging and Deploying Updates 17](#_Toc64314951)

## Introduction

The FPLI is the basis for adjusting Florida’s PK-12 education budget for differences in labor costs across school districts by measuring relative wage differences between districts. The possibility of commuting from a county of residence to a nearby county for employment limits the sustainable difference in wage rates between counties for an equally qualified worker doing identical jobs. Thus a measure of commute times between counties is a valuable set of data in determining wage differentials.

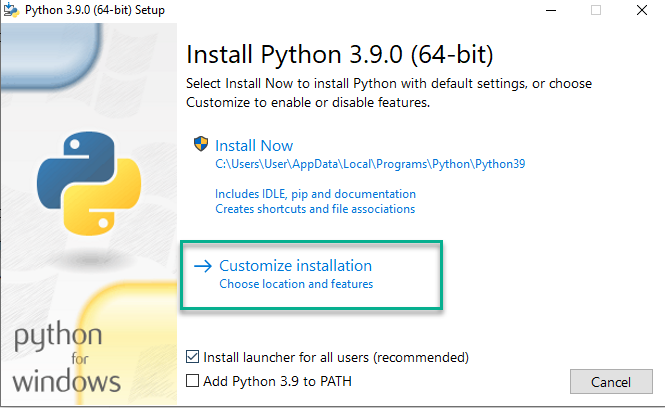
This calculator provides a reliable estimate of the inter-connectivity of county pairs across Florida, and thus provides a useful data set for determining the incremental time required for a worker to commute from one county to another. The function inputs may be used to tailor the pre-processing steps and the number of school pairs excluded from consideration due to the straight-line distance between them. The provided input parameters can thus be used to achieve the user's desired degree of optimization between price of calculation and accuracy/depth of results. The calculator identifies, for each level of school, a customizable number of school pairs which are very likely to reasonably approximate shortest commute times between schools across county borders. The calculator can easily be run with many sets of input parameters and be re-run each year with new data.

## Github

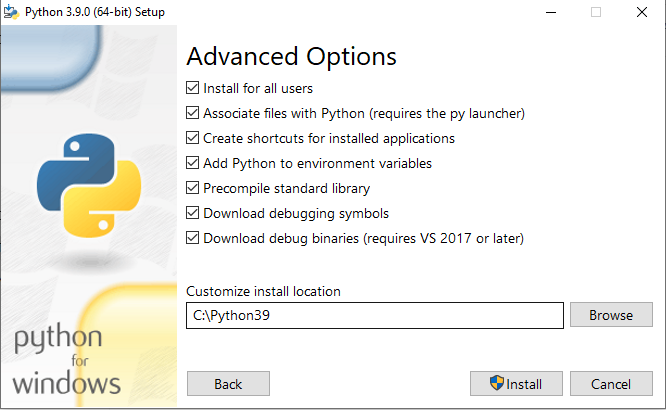
https://github.com/kdewey13/CIS4914-Minimum\_Commute\_FPLI

## Installation

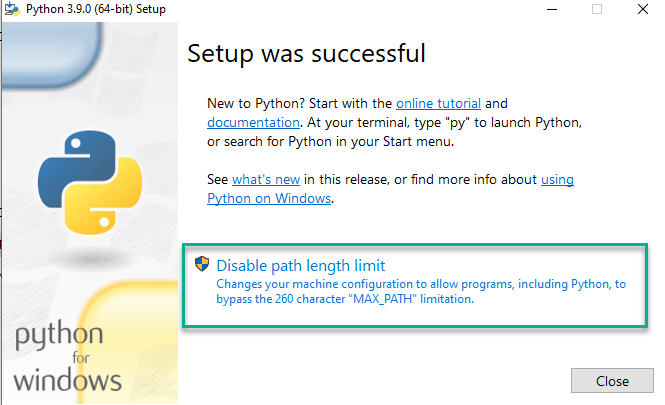
* Download the latest version of Python at <https://www.python.org/downloads/>.
  1. select ‘Custom Installation’ when you open the downloaded executable installation file



* 1. allow all the optional installation parameters
  2. save the program in the C: drive and select all the advanced features then install



* 1. disable path length limit if/when prompted after install

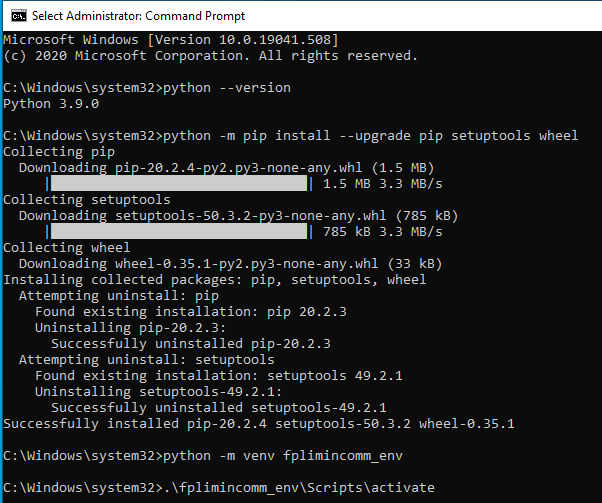


* In the command line (open as administrator):

1. verify your python install success: `python --version`
2. install pip, setuptools, and wheel: `python -m pip install --upgrade pip setuptools wheel`

* To use in Stata; after completing steps 1 and 2 on the command line:

1. Continuing on the command line:
   1. create a virtual environment named fplimincomm (or whatever name desired): `python -m venv fplimincomm\_env`
   2. activate the virtual environment:
      1. on windows: `.\fplimincomm\_env\Scripts\activate`
      2. on linux/mac: `source fplimincomm\_env/bin/activate`

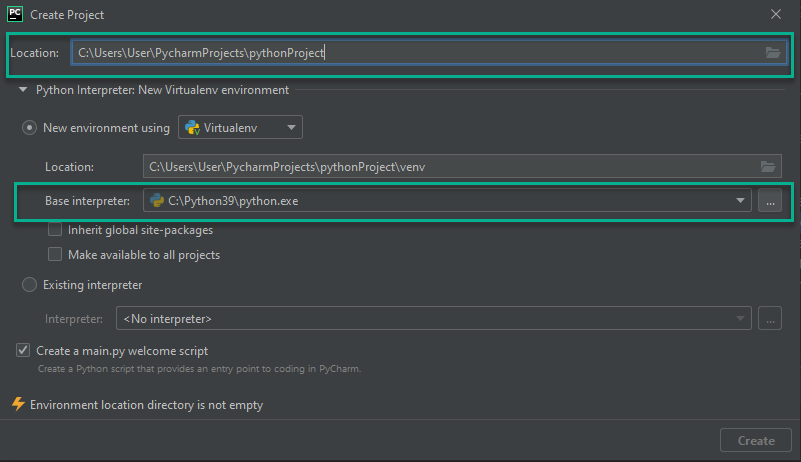


* 1. install the calculator: `pip install FPLI-Minimum-Commutes`
     1. if this does not work try: `pip install FPLI-Minimum-Commutes==X` where X is the current version number, see https://pypi.org/project/FPLI-Minimum-Commutes/

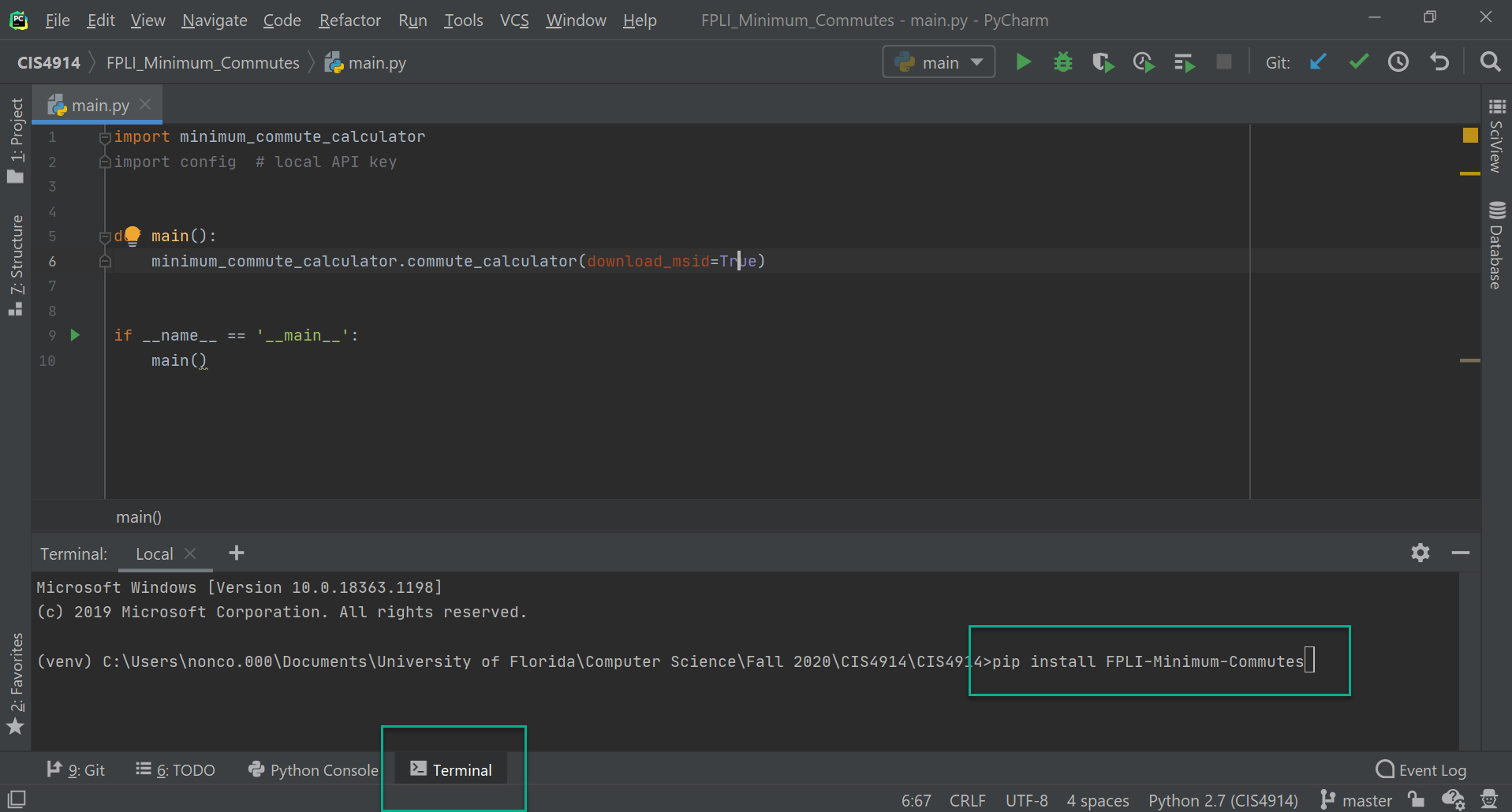
1. On the Stata command line:
   1. run `python search`
      * the above should show the path to the created virtual environment
      * set the python executable to the created virtual environment with: `set python\_exec <path\_to\_virtual\_environment>
2. Proceed as per usage section.
   * Refer to <https://blog.stata.com/2020/08/18/stata-python-integration-part-1-setting-up-stata-to-use-python/>.
   * Note: Stata implementation needs troubleshooting as at time of writing author does not have access to Stata to test these implementation steps.

* To use in PyCharm; complete steps 1 and 2 from the command line section above, then:

1. install PyCharm, community edition will be fine
2. open a new project, use the desired directory as the Location and the installed version of Python (saved in C: above) as the Base Interpreter



1. in the Terminal (tab at bottom of PyCharm window, not the Python Console) run `pip install FPLI-Minimum-Commutes`
   * + if this does not work try: `pip install FPLI-Minimum-Commutes==X` where X is the current version number, see https://pypi.org/project/FPLI-Minimum-Commutes/



1. Proceed as per usage section.

## Process

Conceptually, the process of the function can be broken down into the following high-level steps.

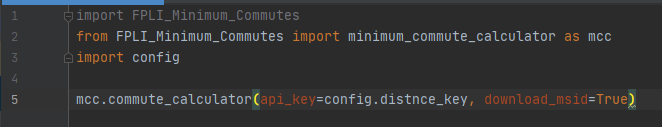
1. [optional] Download the Florida Department of Education Master School ID Database and preprocess it into the format expected by the function.
2. [optional] Preprocess provided excel file containing the Florida Department of Education Master School ID Database data into format expected by the function.
   1. Note: Only perform step 1 *OR* 2, not both. If neither step is done, an input file in the correct format must be provided. See input\_data\_example.csv for an example of the correct format or 'Requirements' below for an explanation
3. Data set-up:
   1. create a dictionary to map school levels to their corresponding input variable values
   2. create an in-memory database to store the data and query against
   3. read the input data in from \_input\_csv\_ file into a data frame
   4. remove any undesired rows from the data frame, as determined by the input variables:
      1. *charter* will keep/remove charter schools
      2. the boolean values within the *desired\_level\_details* variable will determine if each school grade level is kept/removed
   5. save the input data to the database
   6. generate a database cursor, a list of all the schools/their information, and a list of the districts
4. Perform the straight-line distance determination.
   1. for each district, find the schools with max/min latitude and longitude, consider them border schools for that district. The minimal straight-line distance obtained through comparison of the sets of border schools of each county will provide a reasonable first approximation of the minimal straight-line distance between schools in the two counties. Given that the initial elimination of school pairs on the basis of straight-line distance is a) customizable through the *optimization\_radius* input variable and b) intended to be significantly larger than the expected significant elimination threshold (*max\_radius\_to\_consider*), this should be sufficient for the initial winnowing of counties to compare.
   2. for each district pair, compare their border schools and determine if any fall within the optimization radius
   3. if a pair of border schools fall within the optimization radius, we can infer that this county pair may have additional viable pairs of schools, thus we compare all schools between the districts
   4. store schools falling under the optimization radius to the distance\_pairs table in the db
   5. output the distance\_pairs table to a csv (file name determined by *distance\_pairs\_csv* input varible), saving this allows this step to be skipped if the program is run again for any reason by re-loading the saved file
   6. output a csv with the distance pairs details for examining, called distance\_pairs\_details.csv
   * Note: this step need only be performed once, after which the csv that is output can be provided as an input parameter (*distance\_pairs\_csv*) and the distance\_pairs table will be populated from it
5. From the distance\_pairs table, select only those pairs with straight line distances between them that are less than the maximum radius to consider (*max\_radius\_to\_consider*) and store in the straight\_line\_pairs table.
6. Determine minimum distance pairs from straight\_line\_pairs, related minimums, and make API calls (if desired).
   1. For each district (considered origin), determine the other districts for which school pairs exist in the straight\_line\_pairs table (considered destination).
   2. For each destination district possessing school pairs to origin, and for each level of school, do the following:
      1. Select the desired number of pairs for the level of school (see input variable *desired\_level\_details*). Each pair will have a unique origin school, and each will be a minimal straight line distance, provided the uniqueness condition is satisfied (i.e. more minimal values may exist for another selected school, but since it has already been chosen, we do not select it again).
      2. For each of the unique origin schools selected above, select the desired number of pairs (from input variable *desired\_level\_details*) with minimal straight line distances having that school as the origin.
      3. The final step depends on if making API calls was or was not enabled when the calculator was called.
         1. If not enabled (*make\_api\_calls* set to False), the selected pairs are written to the database with their commute times and distances left blank. An output file is created for review; the output pairs represent those that would have had their real-time commutes calculated. Thus the cost of making the associated API calls can be determined from this output file and if desired, the input values can be changed to yield a different result.
         2. If making API calls was enabled (*make\_api\_calls* set to True), then the API request is built and executed, and the result is stored in the database. The complete output file is then generated.

## Usage

### General Usage

To use the module in Python:

* create a file to work in by right clicking on the topmost directory in the project structure (directory tree in left of window) and select "New" then "File". Name the file as desired, using the .py extension (this will be the script you run, so 'main.py' would be appropriate, and hit enter.
* at the top of the file import the package, type: `import FPLI\_Minimum\_Commutes`
* next import the module containing the calculator and rename for brevity by typing: `from FPLI-Minimum-Commutes import minimum\_commute\_calculator as mcc`
* write a line to call the calculator function with: `mcc.commute\_calculator(<desired input parameters>)`, being sure to update all input variables to the desired values.
* to run the program, right click the tab where your file is open (i.e. the filename) at the top of the window, then select 'Debug' (recommended) or 'Run'
* the program will provide status updates as it runs as well as when it completes.



To use the module in a Stata do file:

1. start a python block with `python:`
2. do the same the steps listed above for use in Python
3. end the python block with `end`

* Refer to https://www.stata.com/new-in-stata/python-integration/ for more details.
* At the time of writing the author did not have access to Stata to full test these instructions, please update as needed.

To avoid having to type the API key into the function repeatedly, place a config.py file in the same directory where the function will be run. Inside this file, place only the following text: `distance\_key = "<place\_your\_key\_here>"` where <place\_your\_key\_here> is replaced with your Google Distance Matrix API product key. Leave the quotation marks around the key. Once this file is in place, the api\_key parameter can be set by importing the config file `import config` (do in the line after importing the minimim\_commute\_calculator module). Then in the function input parameter list write: api\_key=config.distance\_key.

Obtain API keys through a Google Developer account. Consult the Google Distance Matrix API pricing sheet to determine the cost of the API calls. The Advanced features are used in this package, ensure to verify correct pricing.

### Input Variables for minimum\_commute\_calculator.calculator()

The main function of the FPLI\_Minimum\_Commutes package is minimum\_commute\_calculator.calculator(). The following input variables are available to tailor the function's actions as desired. See 'Process' for further information about how the variables are utilized. The variable value need not be specified if the default value is desired for use.

* *optimization\_radius*: Default value = 70. Type = Integer. Used as the cut off for retention of school pairs when the straight line distance between them is calculated. For instance, with the default value of 70, no pairs having a distance between them greater than 70 miles will be stored to the database for consideration as a possible minimum commute pair. Should be sufficiently large that no possible desired pair is removed at this round of elimination; the validity of the method used for the first round of elimination relies on this being a generous figure. Stricter elimination may result in leaving out desired school matches by virtue of eliminating entire country pairs. Sharpen elimination more granularly using *max\_radius\_to\_consider*.
* *max\_radius\_to\_consider*: Default value = 50. Type = Integer. Used to tailor the number of pairs selected for calculation of real-time commute time and distance.
* *distance\_pairs\_determination*: Default value = True. Type = Boolean (options: True or False). If true, step 4 in 'Process' above will be completed, i.e. the straight line distances between schools in the input file will be calculated and saved to the database, as well as an output csv file. Only set to False if the function has been run at least one time previously and you possess a csv file containing all the distance pairs.
* *distance\_pairs\_csv*: Default value = 'distance\_pairs.csv'. Type = string (enclose name of file in single or double quotation marks). If *distance\_pairs\_determination* is set to True, this will be the name of the csv file that is output containing the distance pairs that the function calculated. If *distance\_pairs\_determination* is set to False this must be set to the name (or file path if not in the directory where the function is running) of the csv file containing all the distance pairs. If *distance\_pairs\_determination* is set to False and this is not set correctly, the function execution will fail.
* *desired\_level\_details*: Default value = ([True, 3, 3], [True, 2, 2], [True, 2, 2]). Type = tuple containing 3 lists. The first list corresponds to details about the elementary school level, the second to the middle school level, and the third to the high school level. Each interior list is composed of 3 values: a boolean, an integer, and another integer, in that order. The first value in the list, the Boolean, indicates if that school level is to be considered in the calculation (options = True or False). The second value in the list is the number of unique schools to choose for that level per district pair, i.e. if 3 is used, the 3 schools with the smallest distances to schools in the other county will be selected at step 6.ii.a in 'Process' above. The third value in the list is the number of pairs to select involving each unique school per county pair, see step 6.ii.b; i.e. if 3 unique schools are chosen (second value in list = 3) and the third value in the list is 4, then a total of 12 schools will be chosen per district pair, 3 unique schools in the origin county, each matched to 4 schools in the destination county, all of which having minimal straight-line distances within their respective groupings. Taking the default input for an example, it corresponds to comparisons made on all 3 levels; elementary, middle, and high. Per district pair: 9 elementary school pairs will be selected (3 unique origin schools x 3 destination schools per origin), and 4 middle and 4 high school pairs will be selected (2 unique origin schools x 2 destinations schools per origin each) for a total of 17 pairs per district pair (if as many exist and fall within the distance radius restrictions applied). Relevant syntax to pass in the variable: tuples should be enclosed in parentheses with commas separating their contents: (contents1, contents2, contents3, ...etc), and lists should be enclosed in square brackets, again with commas separating their contents: [1, 2, 3, 4,...etc].
* *charter*: Default value = True. Type = Boolean (options: True or False). Used to select whether or not charter schools are considered by the calculator.
* *input\_csv*: Default value = 'fpli\_min\_com\_input\_data.csv'. Type = string (enclose name of file in single or double quotation marks). If allowing the program to download or preprocess the FL DOE Master School ID Database file, this will be the name the program gives to the input file it creates. If providing an already prepared input file, provide the file name here (or file path if the file is not saved in the same directory where the function is being called from). See 'Requirements' below for description of required format.
* *output\_csv*: Default value ='fpli\_min\_commute\_pairs.csv'. Type = string (enclose name of file in single or double quotation marks). This is the file name (or desired file path) of the output of the calculator.
* *download\_msid*: Default value = False. Type = Boolean (options: True or False). Indicates if the program should download FL DOE Master School ID Database file and preprocess it for use as the input file. If providing a file to *input\_csv* do not set this to True. Additionally, if you have already downloaded the FL DOE MSID data and wish to enable \_preprocess\_ and are providing your download as \_unprocessed\_excel\_file\_ set this to False.
* *preprocess*: Default value = False. Type = Boolean (options: True or False). Indicates if the program should preprocess the file provided in the variable *unprocessed\_excel\_file*. Set to True if you have manually downloaded the FL DOE MSID data and wish the program to preprocess it into the correct format. Provide the name of the file you downloaded in *unprocessed\_excel\_file*. Set to False if you are providing your own input file in the *input\_csv* variable, or if you have set *download\_msid* to True, or if you do not possess a copy of the FL DOE MSID data to pass in as *unprocessed\_excel\_file*.
* *unprocessed\_excel\_file*: Default value = None. Type = string (enclose name of file in single or double quotation marks). This is the file name (or file path if the file is not saved in the same directory where the function is being called from) of the FL DOE MSID data manually downloaded from the web. Before passing this file into the program, be sure to open it in excel and verify that it is not corrupt. If it is corrupt, re-save it as an xlsx file. As of Dec. 2020, the downloaded file does not contain correct Beginning of File markers and is corrupt and must be re-saved to be used.
* *api\_key*: Default value = None. Type = string (enclose name of file in single or double quotation marks). The Google Distance Matrix API product key to be used when making the API calls to calculate commute times and distances. See 'General Usage' for suggestion of how to pass this variable into the function.
* *make\_api\_calls*: Default value = False. Type = Boolean (options: True or False). Indicates if the program should make API calls to calculate commute times and distances. If enabled, all pairs selected in step 6 of 'Process' above will have their real time commute times/distances calculated (cost as of Dec. 2020, $10 per 1000 calculations). If disabled, the pairs will be picked, but no calculation made, thus with this variable set to False, the other input parameters can be altered and the function re-run until the most satisfactory number of pairs are achieved. Then the function can be ran with those parameters, along with this variable set to True to obtain the commute data for those pairs.

### Content

The FPLI-Minimum-Commutes package contains three modules.

1. minimum\_commute\_calculator, functions provided:
   1. commute\_calculator(): Main function of use. Process and input parameters described above.
   2. find\_school(): Used by calculator to find a school tuple from a provided list of schools. Not intended for stand-alone use.
   3. get\_time(): Provides the string time stamp corresponding to 7 am the day after the program is being run, in the time zone it is being run from, in the format of seconds from the epoch. Used to make the commute time request for commutes departing at that time. Not intended for stand-alone use.
2. process\_data, functions provided:
   1. download\_data(input\_csv=None): Will download the FL DOE MSID data and preprocess it into the correct format for use in minimum\_commute\_calculator.calculator(); the variable input\_csv is the name or file path to be given to the created input file, its type is a string.
   2. preprocess\_fl\_msid\_data(data\_excel\_file=None, input\_csv=None): Will preprocess the FL DOE MSID data provided in the file referenced by \_data\_excel\_file\_; the variable data\_excel\_file is the name or file path of the excel file containing the FL DOE MSID data, its type is a string, and the variable input\_csv is the name or file path to be given to the created input file, its type is a string.
3. main, functions provided:
   1. main(); Driver function for running the program as a stand-alone. Given that the program is solely intended for use as an imported package, this function is not intended for use, however if in the future changes are made, the driver function can be used for debugging and if needed it can be used to run the program on its own. All it does is call commute\_calculator() so in order to use it you will need to be able to access the file and edit the input parameters.

Additionally, the package contains 'input\_data\_example.csv', which is a csv file in the format that commute\_calculator() expects. It also contains 'FLDOE MSID Information.pdf', which is a pdf describing all the data available in the FL DOE MSID database, along with appendices of all codes contained therein (as of 2020, may need replaced in future). This word document version of the user guide is also contained in the package.

Finally, it contains the file old\_implementation.py, the contents of which are thoroughly explained therein. However to summarize, it contains an implementation of minimum pairs selection based on minimum straight line distance and best and worst case commute time estimation. It was abandoned because even with relatively stringent pair removal, the optimized pairs to run through the API were still over 700,000 pairs, i.e. over $7000 to calculate. It was unknown how much that figure would be reduced by continuing the optimization as the API calculations were done (based on real vs worst case commute estimates), however the implemented plan was devised as a sufficient work-around, as the absolute true minimum commute time is not strictly necessary, and the provided measure of inter-connectivity between counties will serve the purpose of estimating commute costs in determining wage differentials.

### Requirements

* Python version 2.7 or newer.
* For package installation: pip, setuptools, and wheel.
* Python packages required for the calculator (should be automatically installed with the FPLI-Minimum-Commutes package):
  + geopy.distance (straight line distance calculation using Vincenty's formulae)
  + pandas (provides spreadsheet/csv read/write & dataframe manipulation)
  + sqlite3 (in-app database to store data and query against)
  + sqlite3's Error (allows outputs database errors)
  + datetime (manipulates times)
  + requests (make HTTP requests, for commute API requests)
  + pytz (handles time zone manipulation)
  + tzlocal (gets local system time zone
  + xlrd (reads excel files, only needed if the process\_data module is implemented (i.e. if the input data file is not provided))
  + xlsxwriter (writes xlsx files, only needed if the process\_data module is implemented (i.e. if the input data file is not provided))
  + os (used to delete a created file, only needed if the process\_data module is implemented (i.e. if the input data file is not provided))
* For package deployment:
  + packages: setuptools, wheel, and twine
  + pypi account credentials ([fplimincomm@gmail.com](mailto:fplimincomm@gmail.com))
  + pypi API token for FPLI Minimum Commutes
  + contact the repository owner or project lead for credentials
* To make the API calls: valid Google Distance Matrix API product key.

#### Input Data File Format Requirements

The commute\_calculator() function requires the input data file (corresponding to input variable *input\_csv*) to be a csv file with the following columns each with the specified data type:

* district\_name: string value of the district the school is in
* school\_name: string value of the school name
* level: must be the string value of the school grade levels served. In the case of combinations, the levels must be separated by a comma followed by a space. Recommended values:
  + elementary
  + elementary, middle
  + elementary, high
  + elementary, middle, high
  + middle
  + middle, high
  + high
* street\_address: string value of the physical address
* city: string value of the city
* state: string value of the state
* zip: the zip code, may either be the 5 digit zip, or the complete hyphenated zip
* latitude: float
* longitude: float
* charter: must be the Boolean: TRUE or FALSE

An example is provided, see 'input\_data\_example.csv'.

## Packaging and Deploying Updates

To update the package, contact the repository owner, then:

1. Make the desired updates.
2. Increment the version number in setup.py.
3. Push to the remote repository.
4. Place the .pypirc file in your home directory (contact repository owner or project lead for access)
5. In the command line (run as administrator):
   1. navigate to the directory where the package is located
   2. update setuptools and wheel: `python -m pip install --user --upgrade setuptools wheel`
   3. run `python setup.py sdist bdist\_wheel`
   4. update twine: `python -m pip install --user --upgrade twine`
   5. run `python -m twine upload dist/\*` (credentials for the fplimincomm pypi account automatically provided by .pypirc file)