Creating a Graphic User Interface for Roll Call Using Python

## Monica Pagán Spring 2021

**Contents**

Python Version 1

[Install](#_bookmark2) Anaconda Python 2

Configure Your Python Virtual Environment 3

[Testing the Spyder IDE](#_bookmark3) 3

Creating Function for the GUI 4

GUI Creation 5

[Notes](#_bookmark4) 7

[Resources](#_bookmark5) 7

This set of instructions will show the user how to set up a graphic user interface that can convert a .csv file into point shapefiles which can be read on a map. The instructions are intended for anyone who needs to update or change anything within the graphic user interface (GUI).

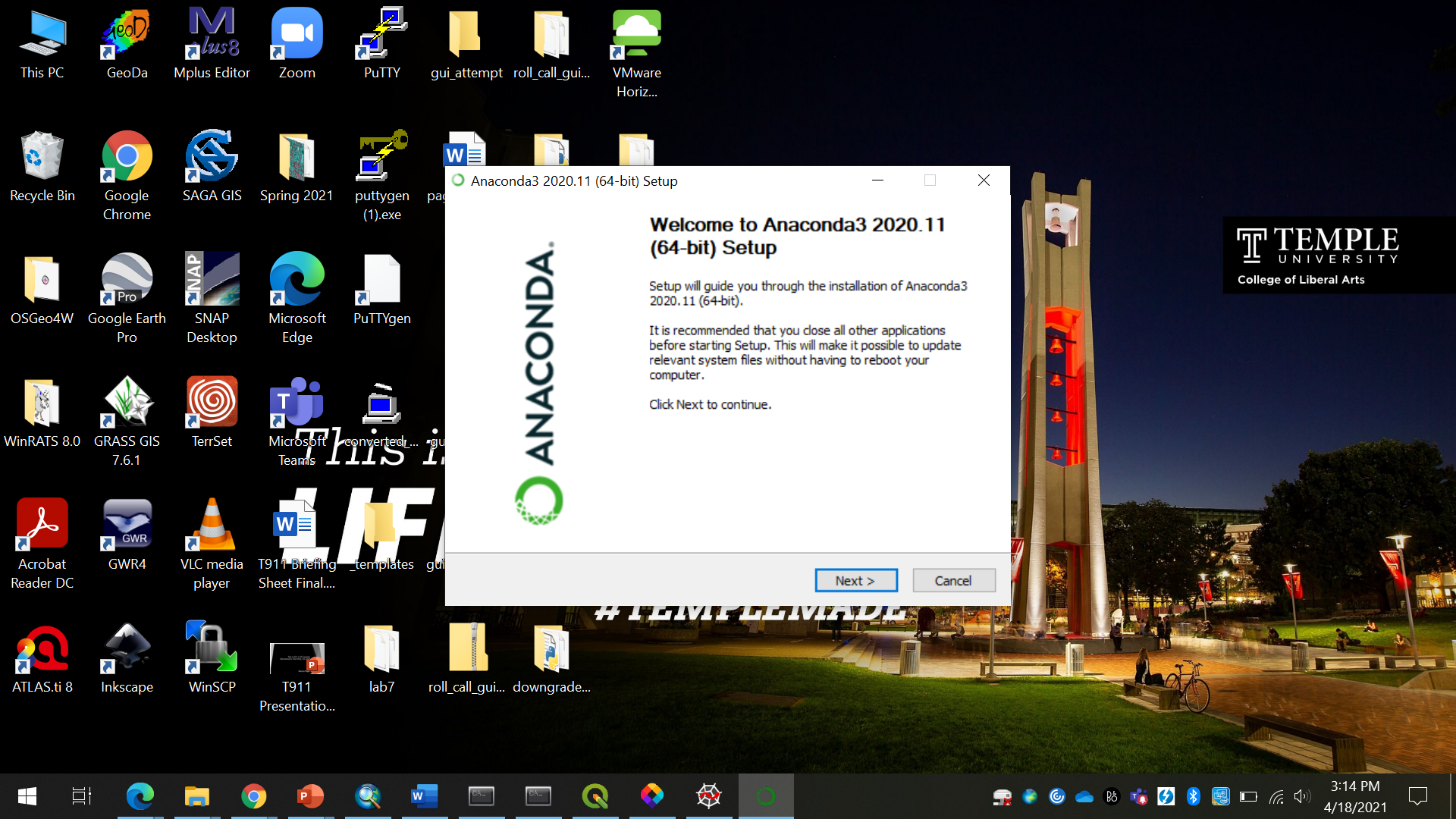
# Python Version

These instructions are written for a Python environment that is set to 3.7 and a Spyder environment of 3.3.6. If the Spyder environment is set to anything higher than 3.3.6, the script will not function, and the GUI will not launch.

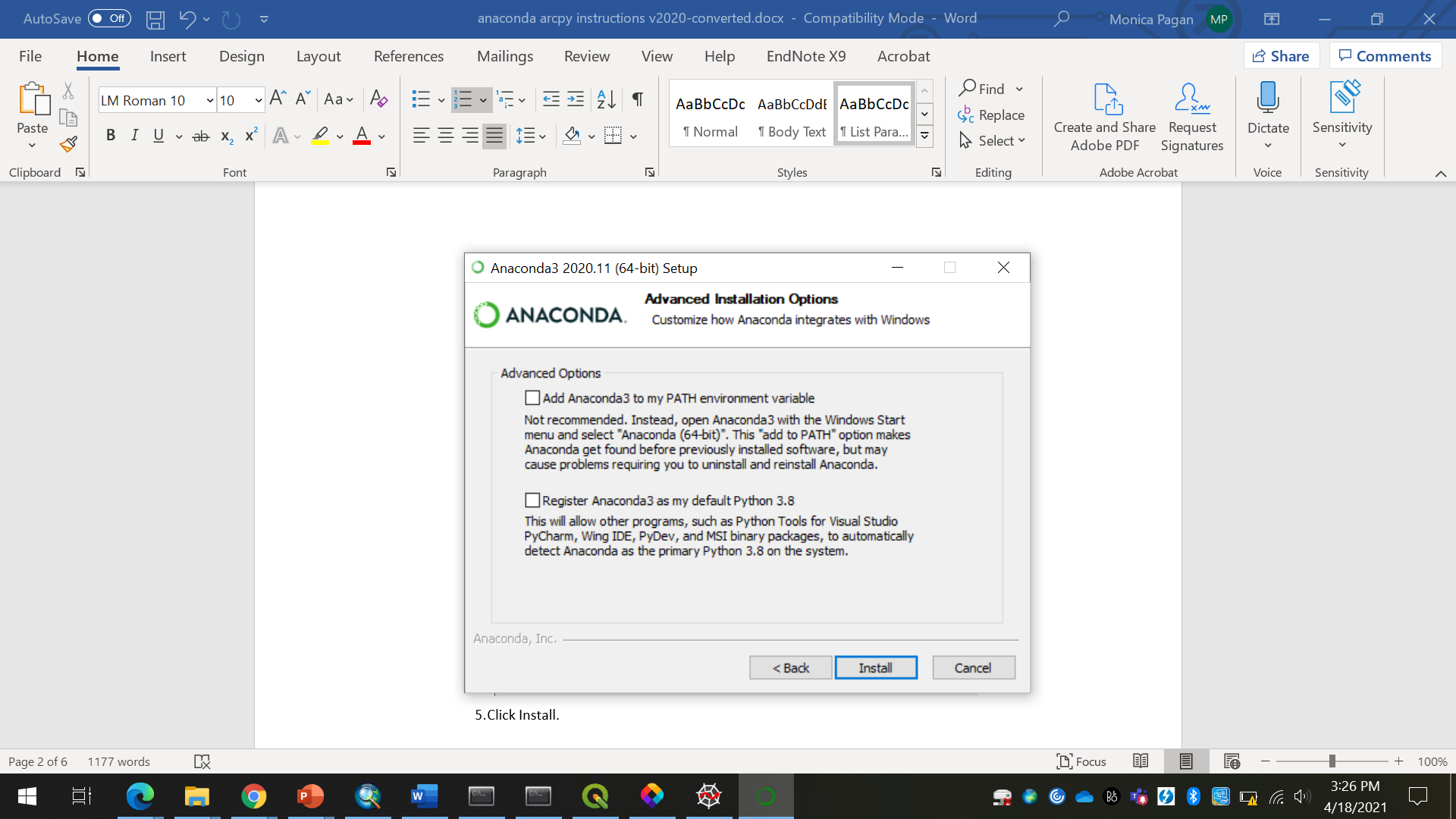
# Install Anaconda Python

1. Download the 64-bit Python 3.7 (or latest) Windows installer: https://www.anaconda.com/products/individual

**NOTE:** Download the 64-bit using the link above. This should give you a downloaded file where you can install Anaconda.



1. Run the installer and select “All Users” if you have administration privileges. If not, select “Just Me.”
2. The installation location should be decided on a case-by-case basis. In the model example used here the program is kept in the C:\Program Files in the computer, but this may be different depending on the machine you are using.
3. Do not add Anaconda to the PATH or make it your system Python. Leave these boxes unchecked:



1. Click Install.

# Configure Your Python Virtual Environment

You may be prompted during this process to create an Anaconda Cloud account. This is not necessary and may be skipped. We will not be making use of an Anaconda Cloud account in this course.

1. Launch the Anaconda Prompt from the Windows Start Menu. Do not launch the Anaconda Powershell Prompt.
2. Create a new environment (with the Spyder IDE) to begin creating the GUI with the following command:

conda create --name roll\_call python=3.7 spyder=3.3.6 geopandas fiona matplotlib shapely pyinstaller

Hit Enter to proceed.

1. As this runs, click “yes” for any changes for the environment and let it continue to run successfully.
2. Close the terminal.
3. If the environment that is created needs to be accessed, this can be done by typing the following line into the terminal:

conda activate roll\_call

# Testing the Spyder IDE

Spyder will be used as a Python editor in this environment. This will allow the user to make any changes necessary to the scripts provided. Spyder can be opened through the start menu or through the terminal if the environment that contains it is activated. This can be done by simply using the line:

spyder

Take note that if you try to open Spyder in another environment or in your base environment, it will not open. This is specific to the ‘roll\_call’ environment that you created.

# Creating Function for the GUI

The creation of the GUI’s function should come first to ensure that the functionality works without any errors. The first step is creating the class that holds the ‘rollcallfunction’. The second next is to have the initialization function where the variables would be created and used throughout the rest of the script. This is seen in the code below:

class rollcallfunction:

def \_\_init\_\_(self, csv, direct, output):

self.direct = direct

self.csv = csv

self.output = output

The third step is to ensure that the .csv file will be converted into a shapefile through the function ‘shapefile’. This can be done by utilizing the package GeoPandas and converting the .csv into a dataframe. This dataframe is then given a crs, and a geometry column. This new shapefile will be output into the user’s directory and can be viewed later with a connection to the internet. This is achieved by the following code:

def shapefile(self):

self.df = pd.read\_csv(self.csv)

geometry = [Point(xy) for xy in zip(self.df.Longitude, self.df.Latitude)]

crs = {'init':'epsg:4326'}

geo\_df = gpd.GeoDataFrame(self.df, crs=crs, geometry=geometry)

geo\_df.to\_file(self.output + '\\' + 'roll\_call\_locations.shp', driver='ESRI Shapefile')

The fourth step in the creation of the function is to create the map that the shapefile will be output to which is created as ‘rc\_map’. The ‘roll\_call\_locations’ shapefile is read through the ‘shapefile’ function up above. It reads in the user’s output directory where the ‘roll\_call\_locations’ are. The geopandas function also reads in the ‘major\_roads’ geopackage file as well as the ‘us\_counties’ geopackage file. These files were read into the plot and ‘x’ and ‘y’ limits were set to the ‘roll\_call\_locations’ shapefile in order to zoom into those extents. Each of the points are labeled with the function below according to the ‘x’ and ‘y’ coordinates. The plot is then saved into the output location of the user’s choice as shown below:

def rc\_map(self):

self.df = geopandas.read\_file(self.output + '\\' + 'roll\_call\_locations.shp')

roll\_call = self.df

maj\_roads = geopandas.read\_file(self.direct + '\\' + 'major\_roads.gpkg')

us = geopandas.read\_file(self.direct + '\\' + 'us\_counties.gpkg')

plt.figure(figsize=(90,90), facecolor='w', edgecolor='k')

ax1 = plt.axes()

maj\_roads.plot(ax=ax1, alpha=0.25, color='grey', facecolor='white')

us.plot(ax=ax1, alpha=0.1, edgecolor='black', facecolor='white')

roll\_call.plot(ax=ax1, alpha=0.25, color='green', facecolor='green')

bar = AnchoredSizeBar(ax1.transData, 1, '50 miles', 4)

ax1.add\_artist(bar)

ax1.set\_title('Roll Call Locations', fontweight = 'bold')

ax1.margins(2,2)

xlim = ([roll\_call.total\_bounds[0], roll\_call.total\_bounds[2]])

ylim = ([roll\_call.total\_bounds[1], roll\_call.total\_bounds[3]])

ax1.set\_ylim(ylim)

ax1.set\_xlim(xlim)

for i, txt in enumerate(roll\_call['Licensee N']):

plt.annotate(txt, (roll\_call['Longitude'][i], roll\_call['Latitude'][i]))

plt.savefig(self.output + '\\' +'roll\_call.pdf')

The fifth step is to output a textfile that is concise and easy to read for the user. By creating the ‘rc\_textfile’ function, the shapefile is read in and only certain columns are selected to make the textfile pertinent to the reader. These can be changed by the user in the future if more information is needed or if excess information needs to be removed. The textfile is output to the users’ file of choice through the ‘self.output’ function:

def rc\_textfile(self):

self.df = geopandas.read\_file(self.output + "\\" + 'roll\_call\_locations.shp')

roll\_call\_text\_file = pd.DataFrame(self.df, columns =['Frequency', 'Call Sign',

'Service Co', 'Phone Numb', 'Licensee N', 'Power ERP', 'Licensee P', 'Latitude',

'Longitude'])

base\_filename = 'roll\_call\_text\_file.txt'

with open(os.path.join(self.output + '\\' + base\_filename),'w') as outfile:

roll\_call\_text\_file.to\_string(outfile)

The last function that is created for the GUI is the ‘runrc’. This runs the functions created up above to be used in the GUI.

def runrc(self):

self.shapefile()

self.rc\_map()

self.rc\_textfile()

# GUI Creation

After setting all of the functions, the GUI is created using a MainWindow and a QWidget. The QWidget is initialized, and the slots for user input are created below through the ‘qtw.QLineEdit()’ and signaled by the ‘self.username\_input’. Three of these slots are created as well as the ‘cancel’ and ‘run’ button. The ‘layout.addRow’ is created and connected to each of the ‘self.username\_input’. Each of these rows are labeled and can be changed at any time. The submit button is then linked to the ‘rc\_func’ which is shown below. This links each of the input boxes to each of the functions that were created in the ‘roll\_call\_function’. After the GUI runs completely, the final portion of the code will create a window that pops up saying that the task is completed and then terminate the window once ‘Ok’ is clicked.

class MainWindow(qtw.QWidget):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.username\_input = qtw.QLineEdit()

self.username\_input2 = qtw.QLineEdit()

self.username\_input3 = qtw.QLineEdit()

self.cancel\_button = qtw.QPushButton('Cancel')

self.submit\_button = qtw.QPushButton('Run')

layout = qtw.QFormLayout()

layout.addRow('Working Directory Including CSV File', self.username\_input)

layout.addRow('Working Directory with Geopackage Files', self.username\_input2)

layout.addRow('Output Directory', self.username\_input3)

button\_widget = qtw.QWidget()

button\_widget.setLayout(qtw.QHBoxLayout())

button\_widget.layout().addWidget(self.cancel\_button)

button\_widget.layout().addWidget(self.submit\_button)

layout.addRow('', button\_widget)

self.setLayout(layout)

self.cancel\_button.clicked.connect(self.close)

self.submit\_button.clicked.connect(self.rc\_func)

def rc\_func(self):

self.csv = self.username\_input.text()

self.direct = self.username\_input2.text()

self.output = self.username\_input3.text()

outp = gui\_without\_contextily.rollcallfunction(self.csv, self.direct, self.output)

outp.runrc()

qtw.QMessageBox.information(self, 'Task completed.', 'Check Directory.')

if \_\_name\_\_ == '\_\_main\_\_':

app = qtw.QApplication(sys.argv)

w = MainWindow(windowTitle='Roll Call')

w.show()

sys.exit(app.exec\_())

This concludes the creation of the GUI.

# Notes

# The current roll\_call\_gui.py file takes about 20 minutes to run fully as there are over

# 600 locations that need labels as well as their text adjusted.

# Resources

* [Anaconda Individual Edition](https://www.anaconda.com/products/individual)
* [Alan D. Moore](https://www.youtube.com/watch?v=M7UdAX77kpY) GUI Creation Videos
* [TIGER/LINE](https://www.census.gov/geographies/mapping-files/time-series/geo/tiger-line-file.html) Datasets
* [Natural Earth](https://www.naturalearthdata.com/downloads/10m-cultural-vectors/roads/) Datasets