Centre of Geographic Sciences

Technical Report

Python GUI Development and Open Source Geoprocessing

Thomas Zuberbühler
thomas.zuberbuehler@gmail.com

supervised by
Mark Hebert

April 3, 2020

Contents

1 Abstract

The aim of this work was to gain a deeper insight into two new areas of the Python programming language and its toolset. On one hand, GUI development with Python and Tkinter was explored and, on the other hand, freely available spatial libraries were researched and used. Those new findings were then practiced and applied working through exercises and by developing a software to clip and project shape files.

In order to achieve this objectives, two different online courses ("GUI Development with Python and Tkinter" and "Automating GIS-Processes") were completed and various additional resources (e.g. "Geo-Python" and official documentation of the respective libraries) were consulted. The exercises, especially those of the course "Automating GIS-Processes", were particularly helpful and also challenging.

With the knowledge acquired, a script to clip shape files and optionally to project the result dataset to another projection was developed. Later, a GUI with Tkinter for the script was produced. The GUI provides a simple user interface and prevents the need to modify the script by the user. In addition, a brief tutorial explaining how to setup a Geo-Python environment with Anaconda was created.

2 Introduction

2.1 Background

Students are primary exposed to Esri's proprietary Python library arcpy and learn the basics of Python at the Centre of Geographic Sciences. However, there are other very powerful Open Source Python modules such as Shapely[1] and Geopandas[2]. Rather than limiting ourselves to proprietary solutions it is essential to know (free) alternatives as well.

Furthermore, it is important to use the right tools for the right situation. Just stringing together arcpy functions is definitely not an efficient and viable solution. Doing that in more complex scripts will lead very soon into performance issues and not maintainable code. With this approach, it is also difficult to write infallible code. In addition, arcpy is only available on Windows platforms with ArcGIS Pro[3] installed. Therefore, arcpy scripts cannot be used across platforms or on systems without a licensed and installed ArcGIS Pro instance.

While arcpy scripts can be used for customized ArcGIS Pro tools, other widely available tools do not come with this ability. It is clearly an advantage of Esri's arcpy library. However, there are free or paid

[1] `https://github.com/Toblerity/Shapely` (last accessed on March 30, 2020)

[2] `https://geopandas.org` (last accessed on March 30, 2020)

[3] Same applies with ArcGIS for Desktop. However, the report's author discourage using Python 2.7 due to its EOL.

alternative Python modules, such as Tkinter[4], PyQt[5] or EasyGUI[6], to build a user interface for any Python scripts.

But most importantly, a programming language can not be learned by utilizing libraries only. It is very critical to understand how a language works, its concepts and best practices. Like every other programming language, Python also has its patterns and anti-patterns.

2.2 Objectives

This open elective subject is an opportunity to explore other Python libraries such as Shapely, PyQGIS[7] and Geopandas. Between January and March 2020, the report's author will work through the course material of the class "Automating GIS-Processes"[8] lectured at the University of Helsinki.

The newly acquired knowledge will then applied writing a simple GIS related Python tool with a graphical user interface. In order to achieve this, a freely available Python module needs to be learned. For this open elective, Tkinter is chosen. The required expertise will be obtained by working through the Udemy online course GUI Development with Python and Tkinter[9].

Furthermore, setting up a working Python environment can be tricky. Therefore, a short tutorial how to install a free spatial Python environment will be written during the course.

2.3 Learning Outcomes

- understand and develop a tutorial to setup an Open Source Geo-Python environment
- learn how to develop basic GUIs with Tkinter
- get an overview of spatial Python libraries such as Shapely, Geopandas and PyQGIS
- develop a user interfaces with Tkinter using Open Source Geo-Python modules
- apply best practice and write pythonic code

2.4 Deliverables

- a short tutorial how to setup an Open Source Geo-Python environment
- a spatial script to clip one or more shape files and project them
- a simple user interface for that spatial script

[4] `https://wiki.python.org/moin/TkInter` (last accessed on March 30, 2020)
[5] `https://www.riverbankcomputing.com/software/pyqt/intro` (last accessed on March 30, 2020)
[6] `http://easygui.sourceforge.net` (last accessed on March 30, 2020)
[7] `https://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook` (last accessed on March 30, 2020)
[8] `https://automating-gis-processes.github.io/site` (last accessed on March 30, 2020)
[9] `https://www.udemy.com/course/desktop-gui-python-tkinter` (last accessed on March 30, 2020)

3 Methodology

This self-learning project has three parts. Firstly, learn Tkinter using an online course. Secondly, get to know open-source GIS python tools. Finally, both learning outcomes leads to the development of a basic GIS tool with an user interface.

Please note that the code produced is not overly commented. The author believes in clean code. It does not make sense to write for each label created that a label is created, or that an input field has a width of 90 if it is obvious looking at the parameters.

> «Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments. Rather than spend your time writing the comments that explain the mess you've made, spend it cleaning the mess.» — Robert C. Martin (2009)

3.1 Learning GUI Development with Tkinter

The online course "GUI Development with Python and Tkinter" by Jose Salvatierra was used to learn GUI development with Tkinter. This section will discuss a few selected learning contents which were most relevant to the final application:

- Python Refresher (Object-Oriented Programming and Type Hinting)
- Layout Manager and how to enable high-DPI in Windows 10
- Object-Oriented Programming with Tkinter
- Packaging and Distributing executables

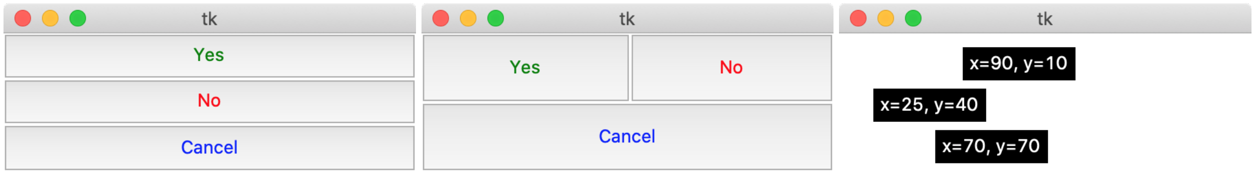
3.1.1 Object-Oriented Programming

Developers who have used object-oriented programming languages in the past are already familiar with the concept. In short, this programming paradigm uses object in order to implement data structures and other features. Without going into details, a class is a template and an object is an instance of that template. Attributes describes characteristics of an instance and methods provides functionality.

Following code shows an example using polymorphism. `Car` is an abstract class, has a two implemented methods and one abstract method. An abstract method has no implementation and its expected that the method is implemented in classes inheriting from the class `Car` (see line 24 and 28 in listing 1). Other methods may be overridden by an inheriting class, such as shown in class `Audi` on line 21. Another noteworthy methods are `__init__(self)` and `__repr__(self)`. The constructor `__init__(self)` initialize an object whereas the method `__repr__(self)` provides a string representation of the object.

```python
from abc import ABC, abstractmethod

class Car(ABC):
    def drive(self):
        print("use speed pedal in {} car".format(self.get_color()))

    def stop(self):
        print("use break in {} car".format(self.get_color()))

    @abstractmethod
    def get_color(self):
        pass

    def __repr__(self):
        return "{} {}".format(self.get_color(), type(self).__name__)

class Audi(Car):
    def __init__(self, color):
        self.color = color

    def stop(self):
        print("sorry, malfunction occurred")

    def get_color(self):
        return self.color

class Volkswagen(Car):
    def get_color(self):
        return "blue"   # looks like Volkswagen has only blue cars ;)

audi1 = Audi("red")
audi2 = Audi("blue")
volkswagen = Volkswagen()

audi1.drive()       # OUTPUT: use speed pedal in red car
audi1.stop()        # OUTPUT: sorry, malfunction occurred.
volkswagen.stop()   # OUTPUT: use break in blue car

print("{}, {} and {}".format(audi1, audi2, volkswagen))
# OUTPUT: red Audi, blue Audi and blue Volkswagen
```

Listing 1: Example for object-oriented programming



(a) pack (b) grid (c) place

Figure 1: Tkinder Layout Managers

3.1.2 Type Hinting

Unlike Java, C# and C++, Python does not come with type safety. Type safety prevents type errors. For example, a function `create_point_geom(x, y)` takes two float numbers but nobody guarantees that a caller of that function does not provide two strings. In Java, for example, type safety is always enforced by its compiler. With Python 3.5, type hinting was introduced. However, Python 3.5+ runtime does still not enforce type safety by using an appropriate IDE or linter does. Listing 2 shows an example of using type hinting. However, it was decided not to use type hiding for the final application.

```python
from typing import List
from shapely.geometry import Point, LineString

def create_point_geom(x: float, y: float) -> Point:
    return Point(x, y)

def create_line_geom(points: List[Point]) -> LineString:
    return LineString(points)

point = create_point_geom(0.0, 1.1)
line = create_line_geom([Point(45.2, 22.34), Point(100.22, -3.20)])
```

Listing 2: Example for type hinting

3.1.3 Layout Manager

There are three different layout manager in Tkinter: pack, grid and place. Pack is the most simple layout manager. By using pack, widgets will be placed relative to each other (see figure 1a and listing 3). Using the grid layout manager, widget can be placed using a table/grid and is the most flexible of all three layout managers. Figure 1b and listing 4 shows an example using grid. Last but not least, the third layout manager enables placement of the widgets precisely using x, y coordinates.

```python
import tkinter as tk

root = tk.Tk()
root.geometry("300x100+100+100")

frame = tk.Frame(root)
tk.Button(frame, text="Yes", foreground="green").pack(fill=tk.BOTH, expand=1)
tk.Button(frame, text="No", foreground="red").pack(fill=tk.BOTH, expand=1)
tk.Button(frame, text="Cancel", foreground="blue").pack(fill=tk.BOTH, expand=1)
frame.pack(fill=tk.BOTH, expand=1)

tk.mainloop()
```

Listing 3: Example using `pack`

```python
import tkinter as tk

root = tk.Tk()
root.geometry("300x100+100+100")

for i in range(2):
    root.columnconfigure(i, weight=1)
    root.rowconfigure(i, weight=1)

tk.Button(root, text="Yes", foreground="green").grid(row=0, column=0,
    sticky="NSEW")
tk.Button(root, text="No", foreground="red").grid(row=0, column=1, sticky="NSEW")
tk.Button(root, text="Cancel", foreground="blue").grid(row=1, column=0,
    columnspan=2, sticky="NSEW")

tk.mainloop()
```

Listing 4: Example using `grid`

3.1.4 Enabling High-DPI in Windows 10

Tkinter user interfaces will look blurry on a Windows system when using a high-resolution screen. However, there is a Python port to access the `SetProcessDpiAwareness`[10] function of Windows which allows to change this issue. Listing 6 shows how to access the `SetProcessDpiAwareness` function.

[10] `https://docs.microsoft.com/en-us/windows/win32/api/shellscalingapi/nf-shellscalingapi-setprocessdpiawareness` (last accessed on March 31, 2020)

```python
import tkinter as tk

root = tk.Tk()
root.geometry("300x100+100+100")

tk.Label(root, text="x=25, y=40", bg="black", fg="white").place(x=25, y=40)
tk.Label(root, text="x=70, y=70", bg="black", fg="white").place(x=70, y=70)
tk.Label(root, text="x=90, y=10", bg="black", fg="white").place(x=90, y=10)

tk.mainloop()
```

Listing 5: Example using `place`

```python
try:
    from ctypes import windll
    windll.shcore.SetProcessDpiAwareness(1)
except:
    pass
```

Listing 6: Enabling High-DPI in Windows 10

3.1.5 Object-Oriented Programming with Tkinter

The development of a large applications with Tkinter can quickly become unclear as well as difficult to extend and maintain. Therefore it is important to write structured and modular code. It is very easy to use object-oriented programming within a Tkinter application.

A complex user interface can be easily divided into several smaller parts. They do not need to know each other but provide an interface which can be accessed. Every component is its own information expert.

For example, given a component to select shape files, this component encapsulate all functions needed to fulfil this task. This could be adding additional files, removing previously selected files and discard all selected shape files. The component only provides its selection through a method, e.g. `get()`, to the outside world and the rest of the application does not have to know any details.

Above described idea is implemented in the final application, namely with the class `ShapeFileSelector` which can be found in `shape_file_clipper_app.py` (see section A.1.3).

3.1.6 Packaging and Distributing executables

There are different tools to build an executable. One of those tools is `pyinstaller` which can be installed by running `conda install pyinstaller`.

By running `pyinstaller yourpythonfile.py -onefile`, the tool analyzes the python script, collects all dependencies and creates an executable file.

3.2 Open Source Geo-Python Environment

For this project, Anaconda[11] was used. Even though the online class "Automating GIS-Processes"[12] was providing some instructions how to set up a running environment to follow their course, there were some issues to overcome. For example, it seems to be important to install python modules in the correct order especially when mixing conda channels (distribution sources).

It took quite some time to overcome those issues since two different operating systems[13] were used for this project. Because of this, an additional deliverable (an setup guide) was defined to include those finding. Please find the full guide in the appendix.

3.3 Learning Geo-Python Libraries

The online class "Automating GIS-Processes" by Henrikki Tenkanen and Vuokko Heikinheimo from the University of Helsinki was used to get an overview of available Open Source Geo-Python libraries. This report will focus on the content learned which is most relevant for the final application:

- Creating spatial features with Shapely
- Using Geopandas for data manipulation

Not discussed in this report are network analysis, static and interactive maps using mplleaflet and developing QGIS Plugins with PyQGIS.

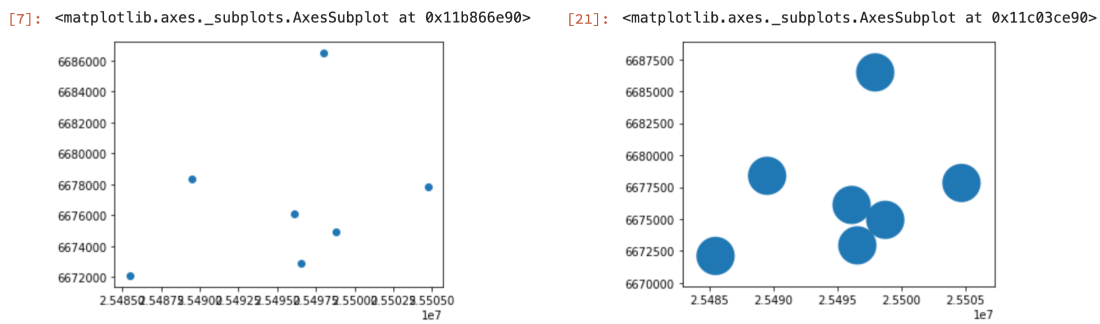
3.3.1 Shapely

According to Shapely's GitHub site[14], Shapely is used for manipulation and analysis of planar geometric objects and based on GEOS[15] and JTS[16] (Sean, 2020). An example of two functions created for exercise 1-2[17] are shown in listing 7. The listing also uses `zip(list1, list2)`, one of many additional learned Python features during the course of this project. Another example of using Shapely is shown in listing 2.

[11] `https://www.anaconda.com` (last accessed on March 31, 2020)
[12] `https://automating-gis-processes.github.io/site` (last accessed on March 30, 2020)
[13] School: Windows 10; Home: MacOS X and Ubuntu 18.04
[14] `https://github.com/Toblerity/Shapely` (last accessed on April 2, 2020)
[15] `https://trac.osgeo.org/geos` (last accessed on April 2, 2020)
[16] `https://locationtech.github.io/jts` (last accessed on April 2, 2020)
[17] `https://github.com/AutoGIS-2019/Exercise-1` (last accessed on April 2, 2020)

```python
def create_lines(start_points: List[Point], dest_points: List[Point]) -> List[LineString]:
    """ Creates lines using a list of start and destination points. """
    lines = []
    for from_, to in zip(start_points, dest_points):
        lines.append(LineString([from_, to]))
    return lines

def calculate_total_distance(lines: List[LineString]) -> float:
    """ Returns the sum of all line lengths. """
    return sum([line.length for line in lines])
```

Listing 7: Functions to create `LineString` using points and evaluate the sum of the line lengths



(a) Points created

(b) Buffers created

Figure 2: Geometries created with listing 9

3.3.2 GeoPandas and Pandas

GeoPandas combines the power of Pandas[18] and Shapely. Pandas is a powerful and freely available tool for data analysis and data manipulation.

In order to understand GeoPandas, it was also important to learn Pandas. To get an introduction in Pandas, exercises 5 and 6 of the course "Geo-Python"[19] by Dave Whipp, Henrikki Tenkanen and Vuokko Heikinheimo has been used. Listing 8 on page 13 shows the author's solution of exercise 5-5[20] from "Geo-Python".

A data frame in GeoPandas requires a column `geometry` containing the features geometry and provides methods such as `buffer(x)`. Listing 9 on page 14 shows an example how to create a buffer with GeoPandas. In addition, the example also shows how to geocode with GeoPandas using Google.

[18] `https://pandas.pydata.org` (last accessed on April 2, 2020)
[19] `https://geo-python.github.io/site` (last accessed on April 2, 2020)
[20] `https://github.com/Geo-Python-2019/Exercise-5` (last accessed on April 2, 2020)

```python
import pandas as pd

# Problem description and CSV file description can be found here:
# https://github.com/Geo-Python-2019/Exercise-5

CSV_FILE = "data/6153237444115dat.csv"
COLUMNS = ['DAY', 'MEAN', 'MAX', 'MIN']

data = pd.read_csv(CSV_FILE)

# clean-up dataset to be usable
data['TEMP'].replace('****', pd.NaT, inplace=True)
data['TEMP'] = data['TEMP'].astype(int)
data = data.dropna(subset=['TEMP'])

# create a convenience column
data['DAY'] = data['YR--MODAHRMN'].astype(str).str[:8]

def create_daily_stats(station):
    """ Creates a statistic for given station and each day with data. """

    # filter station
    usaf = data.loc[(data['USAF'] == station)]
    # create output data frame
    output = pd.DataFrame([], columns=COLUMNS)

    # extract days
    days = usaf.drop_duplicates(subset=['DAY'])
    output['DAY'] = days['DAY']

    # calculates min, max and mean for each day
    for _, row in output.iterrows():
        day = usaf.loc[(usaf['DAY'] == row['DAY'])]
        row['MIN', 'MAX', 'MEAN'] = (day['TEMP'].min(), day['TEMP'].max(), round(day['TEMP'].mean(), 2))

    return output

kumpula = create_daily_stats(29980)
rovaniemi = create_daily_stats(28450)

print(kumpula.head())
print(rovaniemi.head())
```

Listing 8: Parsing daily temparatures (exercise 5-5 of the online course "Geo-Python")

```
import pandas as pd
import geopandas as gpd

from shapely.geometry import Point
from pyproj import CRS

from geopandas.tools import geocode

# Problem description and CSV file description can be found here:
# https://github.com/AutoGIS-2019/Exercise-3

data = pd.read_csv('shopping_centers.txt', sep=';')

# using Google geocoder for a change
geo = geocode(data['addr'], provider='GoogleV3', api_key='place your API key
↪ here', timeout=4)

# project data
geo = geo.to_crs('EPSG:3879')

# join data and drop unused column
geo = geo.join(data)
geo = v.drop(columns=['addr'])

# save shopping centres to shape file
geo.to_file(driver='ESRI Shapefile', filename='shopping_centers.shp')
# geo.plot()

# create buffer and add it to column buffer
geo['buffer'] = None
geo['buffer'] = geo['geometry'].buffer(1500)

geo['point'] = geo['geometry']
geo['geometry'] = geo['buffer']
geo.drop(['buffer'], axis=1, inplace=True)

# save buffer to a separate shape file
geo.to_file(driver='ESRI Shapefile', filename='shopping_centers_buffer.shp')
# geo.plot()
```

Listing 9: Geocode shopping centres (exercise 3-1 of the online course "Automating GIS-Processes")

3.4 Development

The development of the final application (Shape File Clipper) was straightforward. Below the steps which were involved until the final product:

1. Writing a simple (static) script
2. Rewriting script with object-oriented approach
3. Refactoring and extending script
4. Creating a mockup drawing for the user interface
5. Writing a user interface with Tkinter
6. Refactoring and extending user interface code

The Shape File Clipper was developed using PyCharm Professional 2019.3 for Anaconda[21]. It is a special version which works perfectly with an Anaconda environment. Also, GitHub was used as a version control system. Using a version control system has many advantages (GIT Tower, n.d.):

- be able to work with others (collaboration)
- manage versions (versioning)
- go back to previous version (history)
- understand changes
- side effect: have a backup

In the following sections, a few note worthy implementation details will be elaborated and discussed. The full source code can be found in the appendix.

3.4.1 Script

In order for the features to be clipped, the geometries of the polygons may have to be corrected. This can be solved by applying a buffer with a distance of 0 metres. See Listing 10. It shows the function `fix_polygons(data_frame)` extracted from `geo_util.py`.

```python
def fix_polygons(data_frame):
    """ Fixes ring self intersected polygons by applying a buffer of 0. Returns
        the data frame enriched with an additional column indicating if
        geometries are valid or not an a data frame only containing features
        with invalid geometries will be returned as well. """

    data_frame["geometry"] = data_frame["geometry"].buffer(0.0)
    return data_frame
```

Listing 10: Fix ring self intersected polygons by applying a buffer of 0.

[21] `https://www.jetbrains.com/pycharm/promo/anaconda` (last accessed on April 2, 2020)

Listing 11 shows the private method of `ShapeFileClipper` which is responsible to clip geometries of a data frame. In order to clip geometries, earthpy version 0.8.0 was used. However, in the meantime, there is a newer version of GeoPandas providing their own clip function.

```
def __clip_data(self, data_frame):
    """ Clips given data frame. """

    clip_extent = self.clip_extent

    if clip_extent.crs != data_frame.crs:
        key = str(data_frame.crs)
        if key in self.projected_clip_extents:
            clip_extent = self.projected_clip_extents[key]
        else:
            clip_extent = clip_extent.to_crs(data_frame.crs)
            self.projected_clip_extents[key] = clip_extent

    # requires earthpy version 0.8.0 !
    clipped_data_frame = ec.clip_shp(data_frame, clip_extent)
    clipped_data_frame.crs = data_frame.crs
    return clipped_data_frame[~clipped_data_frame.is_empty]
```

Listing 11: Private method of `ShapeFileClipper` to clip geometries of a data frame

3.4.2 Mockup Drawing

At the beginning of the GUI development, a mockup drawing was created. It does not only help to visualize a future user interface, it also helps to identify components. The mockup drawing is shown in figure 3.

3.4.3 User Interface

The class `ShapeFileClipperApp` operates as a coordinator between GUI and script. The class initialize the GUI using separate components such as `ShapeFileSelector` and `ClipExtentSelector`. Furthermore it gathers all input data from the GUI components and passes them on to the script. It is clearly visible in figure 4 that the script does not know any of the GUI components.

An interface called `Validator` was introduced as well. This interface is implemented by all input components (components providing input data) and, therefore, each of them has a `validate()` method. This method is called in a for-loop before the script is executed. If not all input components are valid, the script will not be executed.

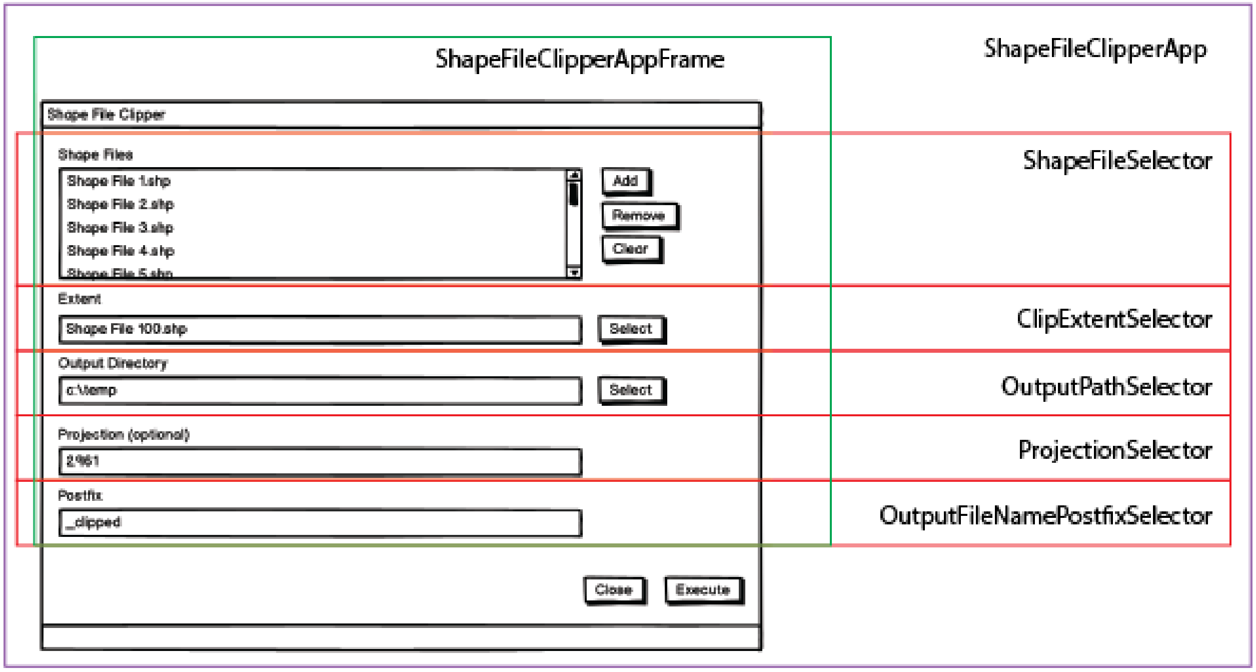


Figure 3: Mockup Drawing for Shape File Clipper

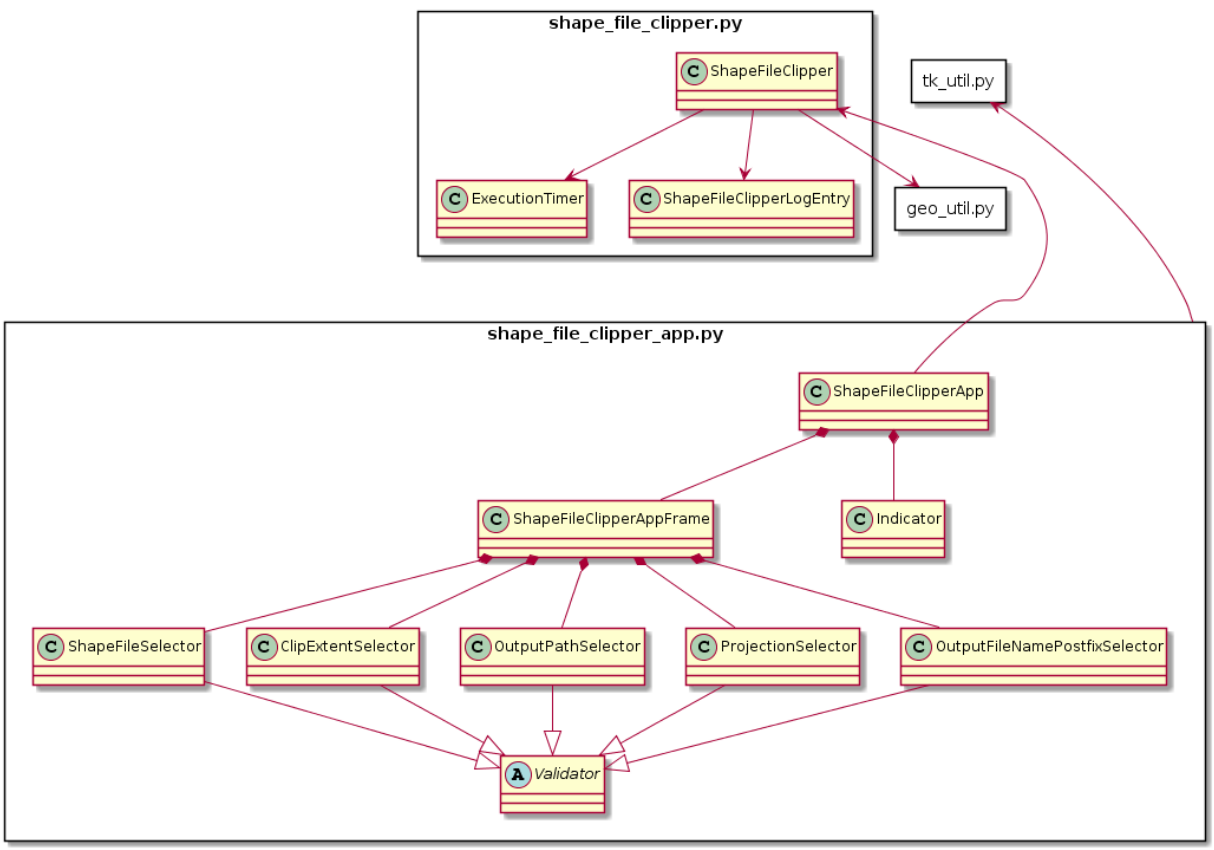


Figure 4: Class and file structure of Shape File Clipper

It is also noteworthy that each component is acting as an information expert. For example, every component knows best how to validate its own data and how to display them.

To avoid that the user interface freezes during the clip and project process, the process runs in a separate thread. See method `ShapeFileClipperApp#execute()` in `shape_file_clipper_app.py`.

```
threading.Thread(target=self.__execute).start()
```

3.4.4 Other Considerations

In order to speed up the execution, dataset will be clipped first and in a second step, using the smaller (clipped) dataset, projected.

Esri's file geodatabase is not an open file format. There is an Open Source driver for GDAL (which also can be accessed through Python) but only supports read-access [22]. However, there is another driver for GDAL which relies on the FileGDB API SDK from Esri and allows to read and write. That said, to reduce complexity, the Shape File Clipper is only working with shape files[23].

4 Results

4.1 Shape File Clipper

Shape File Clipper was developed within a few days and should be considered to be a prototype. Preparing datasets for a specific area is a very common use case. With the Shape File Clipper, cartographers and GIS technician can clip their datasets to a specific study area and optionally re-project their clipped dataset to a desired projection.

Developed with freely available libraries and a user interface built with Tkinter, Shape File Clipper is a standalone tool which can be used across different plattforms [24]. It comes with a simple interface allowing to select shape files, clip extent and output directory (see figure 5). Furthermore, input fields for an optional projection operation and to define a postfix for the output files are provided.

4.2 Learning

The course "GUI Development with Python and Tkinter" was successfully completed (see appendix) within 27 hours[25]. The second course "Automating GIS-Processes" was also successfully worked through with focus on the provided assignments. Those assignment were really helpful and most rewarding.

[22] https://gis.ucla.edu/node/53 (last accessed on April 1, 2020)
[23] see also section 6.4
[24] tested with MacOS X and Windows 10
[25] Note: Hours shown on Certificate of Completion only represent the video duration.

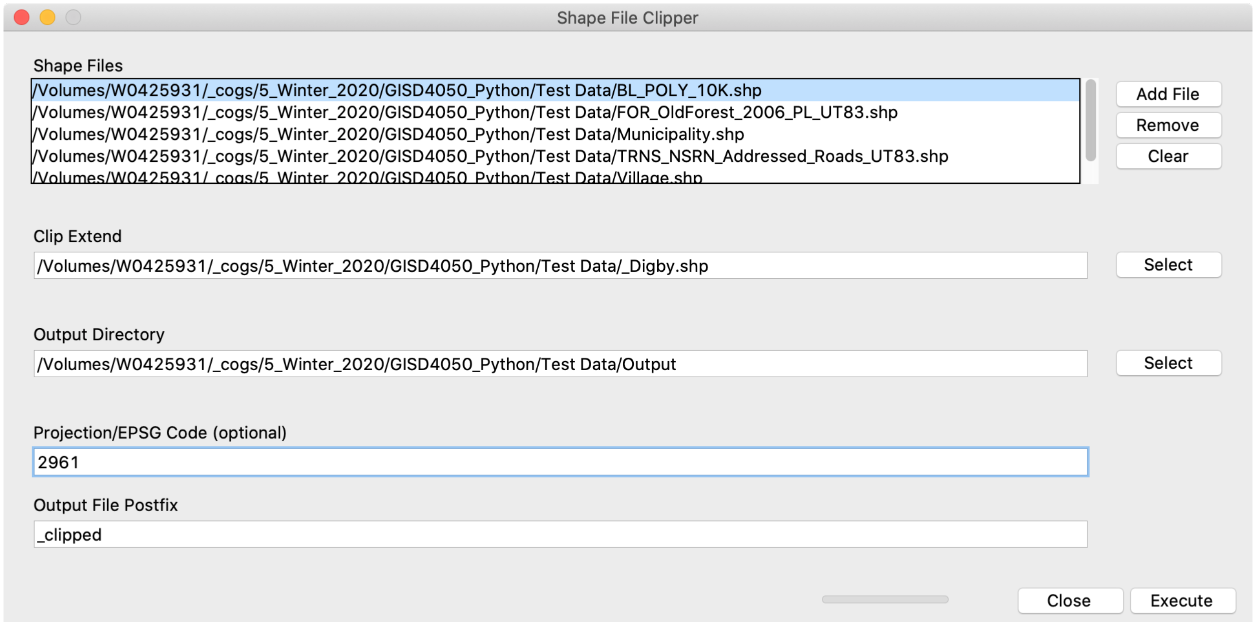


Figure 5: Shape File Clipper

4.3 Setting up a Geo-Python environment

Also, a tutorial how to set up an Open Source Geo-Python environment with Anaconda was written. The tutorial explains briefly how to install Anaconda and required Python modules in order to work through the online course "Automating GIS-Processes". However, the guide does expect prior knowledge and is not suitable for novices.

4.4 Issues and Solutions

4.4.1 Clipping

For clipping a features to an extent, the final application is using `earthpy`. However, just recently a new version of `earthpy` was released. This new version has its clipping function deprecated and a message is being shown that Geopandas new clip version should be used. To overcome this issue, an old version of `earthpy` (version 0.8.0) needs to be installed. This can be done by executing `conda install -c conda-forge earthpy=0.8.0`.

4.4.2 Deployment

Using `pyinstaller` did not work as expected and no executable were successfully built. Geopandas and `earthpy` comes with a lot of dependencies. It looks like that `pyinstaller` is not able to resolve all those dependencies. After a couple of hours trying to solve the issue, it was decided to find an alternative deployment. Therefore, Shape File Clipper can currently only be run on systems with a Anaconda Python environment which can be set up as shown in listing 12.

```
conda create -n clipper python=3.7.6
conda activate clipper
conda install gdal
conda install geopandas
conda install -c conda-forge earthpy=0.8.0
```

Listing 12: Alternative Deployment (run with `python shape_file_clipper_app.py`)

5 Discussion

The GUI framework Tkinter is not really convincing to use for future projects. It may be a good idea to look into alternatives before starting the next Python project that requires a user interface.

A completely different and very positive impression was gained using the Open Source Geo-Python libraries. Those freely available libraries are at least as good as Esri's proprietary `arcpy` library. There is a little more time required for the training and there are fewer out-of-the-box solutions. Whereas `arcpy` is easier to use, most of the time slower and costs a lot of money.

However, comparing `arcpy`'s API with other library's APIs (from a software engineer's view), `arcpy` does absolutely not provide a good API design[26], i.e. lots of inconsistency, use of wrong data types (mostly using strings) and strange naming convention (e.g. `arcpy.Clip_analysis`) to name a few issues.

5.1 Tkinter Alternative: PyQt

Tkinter has a few limitation and comes with unnecessary complexity. For example, there is no simple solution to hide a frame or other widget. To overcome this impediment an additional frame must be used and placed over the widget to be hidden.

There are other GUI libraries for Python such as PyQt. To solve the above problem PyQt provides a simple function. Any QtWidgets[27] instance implements the methods hide() and show().

Also, Tkinter has no advanced widgets such as table, scrollable list and date picker. Those advanced widget needs to be developed manually (widget behaviour and look-and-feel) which can be a very time-consuming process. On the other hand, PyQt has those widgets built-in. Furthermore, user interfaces developed with PyQt comes with a more modern look.

[26] For an introduction see `https://www.youtube.com/watch?v=heh4OeB9A-c` (last accessed on April 1, 2020).
[27] `https://doc.qt.io/qtforpython/PySide2/QtWidgets/QWidget.html` (last accessed on March 31, 2020)

However, PyQt also comes with two possible disbenefits. While Tkinter is distributed with Python, PyQt has to be installed separately. However, this is easy to overcome by distributing an installer with the software. For commercial solutions, PyQt's licensing may be a disadvantage. PyQt is available under the GPL. However, a commercial license, if needed, can be bought.

In addition to working through the Tkinter online course, the first chapters of the e-book "Create Simple GUI Applications with Python & Qt5"[28] by Martin Fitzpatrick were explored.

6 Future Work

6.1 User Interface

Due to the time constraint given for this project, and at this point, the application presents itself more like a prototype. However, it also points out the necessity to have more than one iteration when developing an application. Among other things, look-and-feel can definitely be improved, the responsiveness of the user interface should be optimized as well as the alignment of the widgets.

There are two ways to improve the application's look-and-feel. Either to develop a customized style for `tkinter.ttk` or substitute Tkinter with another GUI framework, e.g. with PyQt. Considering all other required improvement such as the user interface's responsiveness, as well as considering the learning curve for both options, it is recommended to replace the GUI framework.

6.2 Selecting Shape Files

6.2.1 Component Behaviour

At the moment it is not possible to select more than one shape file in the `Listbox` and remove those from the list. Furthermore, the `Listbox` widget does not have a standard behaviour as a user would expect. It is suggested to tweak Tkinter's `Listbox` widget to acquire a behaviour a computer user is used to. If this is not possible, it would be best to replace the GUI framework.

6.2.2 Extension

It is possible to select one or more shape file using the file chooser component. However, an idea would be to let the user select one or more directories and all shape files in those directories will be added to the selection.

6.3 Selecting Projection

There is no option to select a projection from a list yet. It would be great if the user could open a dialog and select a projection by name or EPSG code using a filterable list or table.

[28] `https://leanpub.com/create-simple-gui-applications` (last accessed on March 31, 2020)

6.4 File Geodatabase

Esri's file geodatabase is a very common file format. In addition to be able to select shape files, the Shape File Clipper could be extented to allow selecting features from geodatabases. However, there are a few implementation consideration to take into account. For example, if a user can select features from different geodatabases or just from one. Another complex issue would be how to realize a usable and intuitive user interface if a user can choose from different sources.

6.5 Indicator

At the moment, the input fields will be hidden behind an empty frame. But this panel could be used to show a console with details about the progress.

6.6 Source Code

Presented source code requires further refactoring. In particular, `shape_file_clipper_app.py` is too long and should be split into several different source files.

7 Conclusion

This project was a great opportunity to explore new libraries and technologies. The focus of this open elective was on the two online courses (around 5/6 of the class time) while the application produced was a necessary by-product in order to have a deliverable.

The course "GUI Development with Python and Tkinter" was less hands-on delivered. However, it gave a tightly packed insight and is probably one of the best courses to learn Tkinter. Everything needed is covered. Downside of this course was that it was only delivered through videos. Therefore it is not the best source to look up information.

On the other hand, the course "Automating GIS-Processes" takes a different path and everything is delivered in text. Since this course is also delivered in person at the University of Helsinki, supporting videos were recorded and provided to the public. Furthermore, assignments at the end of every section were really helpful and probably most helpful to learn Shapely, Geopandas and other Open Source libraries.

There were some issues setting up a working Open Source Geo-Python environment using Anaconda. However, this was a great occasion to learn how to use and work with different Python environments on different systems. As part of this work, short instructions, explaining how to set up an Open Source Geo-Python environment were created.

While working and exploring new Python libraries was very rewarding, using Tkinter was rather frustrating. Tkinter is, compared with other GUI frameworks, not easy to understand, its API is not very convenient to use and its look-and-feel is to hide away. However, this newly learned experience will be very helpful for a future evaluation of GUI frameworks in Python.

The result application is a fully functional prototype. Given the time constraint, focus was rather on the source code than UX design. Source code is object-oriented and tries to follow best practice as good as possible. Sometimes it was very challenging not knowing all aspects of Python programming. That said, by developing this application, not only were new libraries explored, it was also a great opportunity to dive deeper into Python programming. However, the result needs more improvement in both areas, code and user interface.

Additional learning outcomes, such as how to use Jupyter Lab and GDAL command line commands (to name only two), were also very beneficial for future opportunities. An early script version of the Shape File Clipper was also used to process data used in another project.

8 References

GIT Tower. (n.d.). *Why Use a Version Control System?* Retrieved 2020-04-02, from `https://www.git-tower.com/learn/git/ebook/en/command-line/basics/why-use-version-control`

Martin, R. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

Sean, G. (2020, January 27). *GitHub Repository Readme File.* Retrieved 2020-04-02, from `https://github.com/Toblerity/Shapely`

Please Note: Cross references are shown as footnotes. This section contains sources effectively referenced in the report.

A Appendix

A.1 Source Code: Shape File Clipper

A.1.1 shape_file_clipper.py

```
import os
import shutil
import glob
import time

import geopandas as gpd
import earthpy.clip as ec

import logging

import geo_util

TEST_CLIP_EXTENT = r"/Users/thozub/ShapeFileClipper/test_extent.shp"
TEST_SHAPE_FILE_DIRECTORY = r"/Users/thozub/ShapeFileClipper/test_shape_files"
TEST_SHAPE_FILES = glob.glob(os.path.join(TEST_SHAPE_FILE_DIRECTORY, "*.shp"))
TEST_OUTPUT_PATH = r"/Users/thozub/ShapeFileClipper/test_output"

def init_logging():
    """ Initialize logging. """

    log_format = "%(asctime)s  %(levelname)s %(message)s"
    logging.basicConfig(level=logging.INFO, format=log_format)

class ExecutionTimer:
    """ Timer class to stop execution time. """

    def __init__(self):
        self.start_time = None
        self.reset()

    def get_running_time(self):
        """ Returns the running time since creating this timer or since its last reset. """

```

```
        return time.time() - self.start_time

    def reset(self):
        """ Resets the timer. """

        self.start_time = time.time()

    def log_running_time(self):
        """ Logs the execution time since creating this timer or since its last reset. """

        logging.info("Execution time: {0:.0f} seconds".format(self.get_running_time()))

class ShapeFileClipperLogEntry:
    """ Log Entry class of ShapeFileClipper. """

    def __init__(self, result_message, ignored_values, execution_time, do_log_message_immediately=True):
        self.result_message = result_message
        self.ignored_values = ignored_values
        self.execution_time = execution_time

        if do_log_message_immediately:
            logging.info(result_message)

    def __repr__(self):
        if self.ignored_values is None or len(self.ignored_values) == 0:
            return "{}\nExecution Time: {}".format(self.result_message, self.execution_time)

        return "{}\nIgnored (invalid) Values:\n{}\nExecution Time: {}".format(
            self.result_message, self.ignored_values, self.execution_time)

class ShapeFileClipper:
    """ Shape File Clipper script. """

    def __init__(self, clip_shape_file, output_path, output_file_postfix):
        self.clip_extent = gpd.read_file(clip_shape_file)
```

```
        self.projected_clip_extents = {}  # applying flyweight pattern
        self.output_path = output_path
        self.output_file_postfix = output_file_postfix
        self.log = []

    def __generate_clipped_output_file_path(self, file_path):
        """ Generates the output file path based on the input shape file, output path
            and output file post fix. """

        name, extension = os.path.splitext(os.path.basename(file_path))
        new_file_name = "".join([name, self.output_file_postfix, extension])
        if self.output_path is None:
            return os.path.join(os.path.dirname(file_path), new_file_name)
        return os.path.join(self.output_path, new_file_name)

    def __clip_data_frame(self, data_frame):
        """ Clips given data frame. """

        clip_extent = self.clip_extent

        if clip_extent.crs != data_frame.crs:
            key = str(data_frame.crs)
            if key in self.projected_clip_extents:
                clip_extent = self.projected_clip_extents[key]
            else:
                clip_extent = clip_extent.to_crs(data_frame.crs)
                self.projected_clip_extents[key] = clip_extent

        # requires earthpy version 0.8.0 !
        clipped_data_frame = ec.clip_shp(data_frame, clip_extent)
        clipped_data_frame.crs = data_frame.crs
        return clipped_data_frame[~clipped_data_frame.is_empty]

    def __clip_shape_file(self, shape_file, fix_invalid_polygons=True):
        """ Clips given shape file. Fixes invalid polygon when parameter is
            set to True (default). """

        # load shape file
```

```python
        data_frame = gpd.read_file(shape_file)

        shape_file_name = os.path.basename(shape_file)

        # fix invalid polygons
        if fix_invalid_polygons and geo_util.is_polygon_feature_set(data_frame):
            data_frame = geo_util.fix_polygons(data_frame)

        # check geometries and enrich with column is_valid
        valid_values, non_valid_values = geo_util.check_geometries(data_frame)

        clipped_data_frame = None
        if any(valid_values.intersects(self.clip_extent.unary_union)):
            clipped_data_frame = self.__clip_data_frame(valid_values)

        if clipped_data_frame is None or len(clipped_data_frame) == 0:
            return None, non_valid_values, "No features in {} within clipping extent. Ignored.".format(shape_file_name)

        return clipped_data_frame, non_valid_values, "{} successfully clipped.".format(shape_file_name)

    def __save_shape_file(self, data_frame, input_shape_file):
        """ Saves given data_frame to a shape file. Requires path to input_shape_file to generate
            output file path. Calls __generate_clipped_output_file_path(input_shape_file)
            internally. """

        output_file_path = self.__generate_clipped_output_file_path(input_shape_file)
        data_frame.to_file(driver="ESRI Shapefile", filename=output_file_path, encoding="UTF-8")
        return output_file_path

    def clip(self, shape_file):
        """ Clips given shape file. """

        logging.info("Processing {}...".format(os.path.basename(shape_file)))
```

```
        timer = ExecutionTimer()

        clipped_data_frame, ignored_values, result_message =
         ↪ self.__clip_shape_file(shape_file)

        self.log.append(ShapeFileClipperLogEntry(result_message, ignored_values,
         ↪ timer.get_running_time()))
        timer.reset()

        if clipped_data_frame is not None:

            output_file_path = self.__save_shape_file(clipped_data_frame,
             ↪ shape_file)

            result_message = "{} saved.".format(output_file_path)
            self.log.append(ShapeFileClipperLogEntry(result_message, None,
             ↪ timer.get_running_time()))


    def clip_and_project(self, shape_file, epsg_code):
        """ Clips given shape file and project result with given EPSG code. """

        logging.info("Processing {}...".format(os.path.basename(shape_file)))
        timer = ExecutionTimer()

        clipped_data_frame, ignored_values, result_message =
         ↪ self.__clip_shape_file(shape_file)

        self.log.append(ShapeFileClipperLogEntry(result_message, ignored_values,
         ↪ timer.get_running_time()))
        timer.reset()

        if clipped_data_frame is not None:

            clipped_data_frame = geo_util.project_data_frame(clipped_data_frame,
             ↪ epsg_code)
            output_file_path = self.__save_shape_file(clipped_data_frame,
             ↪ shape_file)

```

```python
            result_message = "{} projected and saved.".format(os.path.basename(output_file_path))
            self.log.append(ShapeFileClipperLogEntry(result_message, None, timer.get_running_time()))

    def print_log(self):
        """ Prints log to console. """

        for log_entry in self.log:
            print(log_entry)

def clip_and_project(shape_files, clip_shape_file, output_path, output_file_postfix, epsg_code):
    """ Clip and project given shape files. """
    clipper = ShapeFileClipper(clip_shape_file, output_path, output_file_postfix)
    for shape_file in shape_files:
        clipper.clip_and_project(shape_file, epsg_code)
    # clipper.print_log()

def run_test():
    """ Runs test with given test data. """

    # clear output directory
    shutil.rmtree(TEST_OUTPUT_PATH, ignore_errors=True)
    os.mkdir(TEST_OUTPUT_PATH)

    init_logging()
    timer = ExecutionTimer()
    clip_and_project(TEST_SHAPE_FILES, TEST_CLIP_EXTENT, TEST_OUTPUT_PATH, "_clipped", 2961)
    timer.log_running_time()

if __name__ == '__main__':
    run_test()
```

A.1.2 geo_util.py

```python
from pyproj import CRS
from shapely.geometry.polygon import Polygon
from shapely.geometry.multipolygon import MultiPolygon

def check_geometries(data_frame):
    """ Checks the geometry of each feature and returns two data frames, one with all valid values
        and a secondone with the invalid features. """

    # enrich data_frame with an is_valid column to avoid a second, redundant validation
    # for both data_frames
    data_frame["is_valid"] = data_frame["geometry"].is_valid

    valid_values = data_frame.loc[data_frame["is_valid"]]
    non_valid_values = data_frame.loc[data_frame["is_valid"] == False]

    return valid_values, non_valid_values

def fix_polygons(data_frame):
    """ Fixes ring self intersected polygons by applying a buffer of 0. Returns the data frame
        enriched with an additional column indicating if geometries are valid or not an a data
        frame only containing features with invalid geometries will be returned as well. """

    data_frame["geometry"] = data_frame["geometry"].buffer(0.0)
    return data_frame

def is_polygon_feature_set(data_frame, include_multipolygons=True):
    """ Returns true if the given data frame contains polygons or multi-polygons. """

    if include_multipolygons and isinstance(data_frame.loc[0, "geometry"], MultiPolygon):
        return True
```

```
    return isinstance(data_frame.loc[0, "geometry"], Polygon)


def project_data_frame(data_frame, epsg_code):
    """ Re-project given data frame using given EPSG code. Returns the
        reprojected data frame. """

    crs = CRS.from_epsg(epsg_code).to_wkt()
    data_frame = data_frame.to_crs(crs)
    return data_frame
```

A.1.3 shape_file_clipper_app.py

```
from pyproj import CRS
from shapely.geometry.polygon import Polygon
from shapely.geometry.multipolygon import MultiPolygon

def check_geometries(data_frame):
    """ Checks the geometry of each feature and returns two data frames, one with all valid values
        and a second one with the invalid features. """

    # enrich data_frame with an is_valid column to avoid a second, redundant validation
    # for both data_frames
    data_frame["is_valid"] = data_frame["geometry"].is_valid

    valid_values = data_frame.loc[data_frame["is_valid"]]
    non_valid_values = data_frame.loc[data_frame["is_valid"] == False]

    return valid_values, non_valid_values

def fix_polygons(data_frame):
    """ Fixes ring self intersected polygons by applying a buffer of 0. Returns
        the data frame enriched with an additional column indicating if
        geometries are valid or not an a data frame only containing features
        with invalid geometries will be returned as well. """

```

```python
    data_frame["geometry"] = data_frame["geometry"].buffer(0.0)
    return data_frame

def is_polygon_feature_set(data_frame, include_multipolygons=True):
    """ Returns true if the given data frame contains polygons or multi-polygons.
     ↪ """

    if include_multipolygons and isinstance(data_frame.loc[0, "geometry"],
     ↪ MultiPolygon):
        return True
    return isinstance(data_frame.loc[0, "geometry"], Polygon)


def project_data_frame(data_frame, epsg_code):
    """ Re-project given data frame using given EPSG code. Returns the
        reprojected data frame. """

    crs = CRS.from_epsg(epsg_code).to_wkt()
    data_frame = data_frame.to_crs(crs)
    return data_frame
```

A.1.4 tk_util.py

```python
import tkinter as tk

def set_dpi_awareness():
    """ Enables DPI Awareness on Windows. Ignores any errors. """

    try:
        from ctypes import windll
        windll.shcore.SetProcessDpiAwareness(1)
    except:
        pass

NS = tk.N + tk.S
NW = tk.N + tk.W
NE = tk.N + tk.E
SW = tk.S + tk.W
```

```
SE = tk.S + tk.E
EW = tk.E + tk.W

NSW = tk.N + tk.S + tk.W
NSE = tk.N + tk.S + tk.E
NEW = tk.N + tk.E + tk.W
SEW = tk.S + tk.E + tk.W

NSEW = tk.N + tk.S + tk.E + tk.W
```

A.2 Certificate of Completion

The Certificate of Completion of the course "GUI Development with Python and Tkinter" is shown in figure 6 on page 34. Please note that the listed hours on the certificate only represent the duration of the video duration.


Certificate of Completion

This is to certify that **Thomas Zuberbuehler** successfully completed 11 total hours of **GUI Development with Python and Tkinter** online course on March 7, 2020

Jose Salvatierra
Jose Salvatierra, Instructor

Teclado by Jose Salvatierra
Teclado by Jose Salvatierra, Instructor

& Udemy

#BeAble

Certificate no: **UC-4a9ae8fe-e0c4-40f6-a84b-59731d0a6baf**
Certificate url: **ude.my/UC-4a9ae8fe-e0c4-40f6-a84b-59731d0a6baf**

Figure 6: Certificate of Completion: GUI Development with Python and Tkinter

A.3 Project Management

A.3.1 Logged Hours

Total logged time between December 23, 2019 and April 2, 2020 are 138 hours.

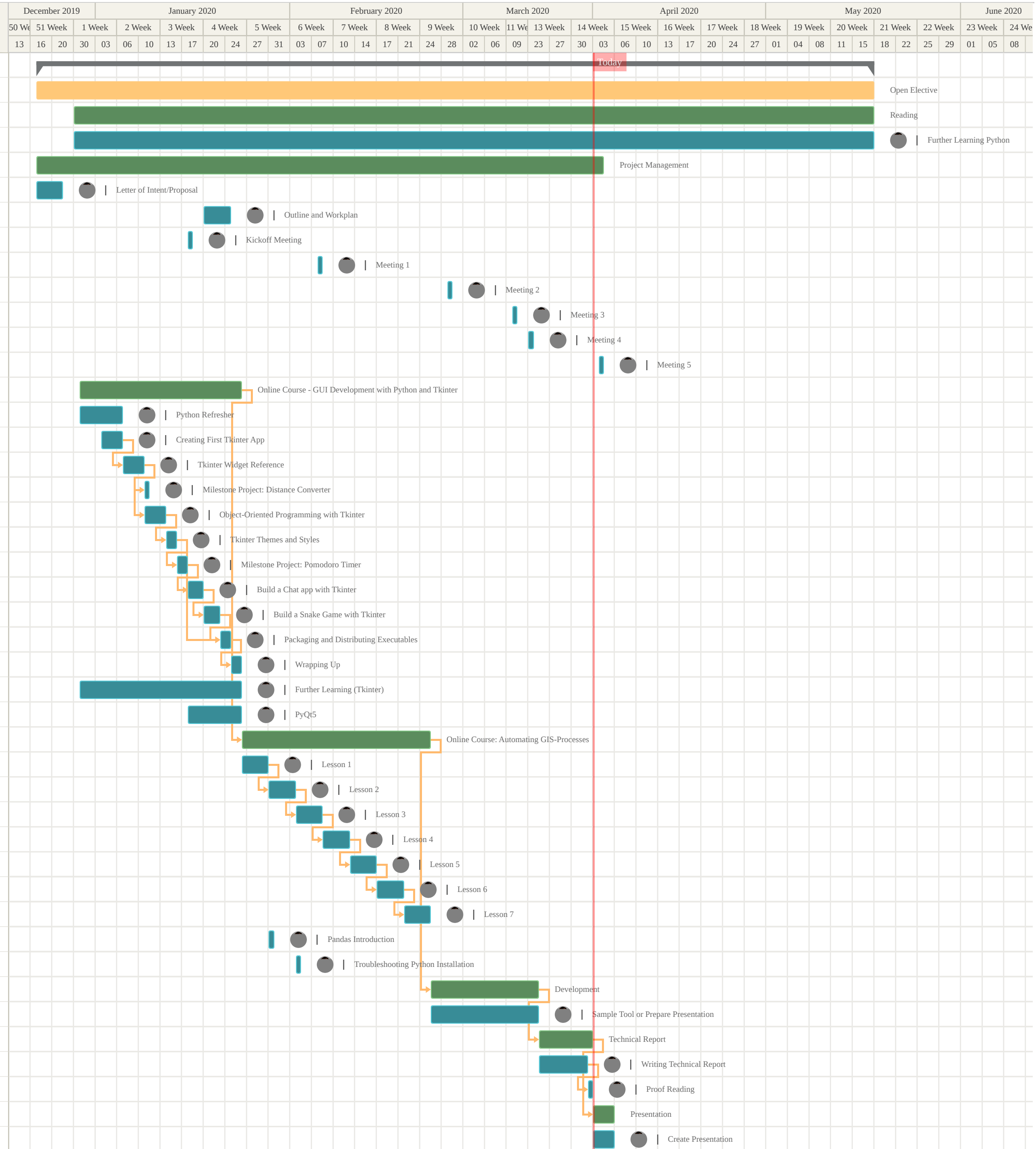
Task Name	**Project**	**Date**	**Time (min)**	**Comment**
Writing Technical Report	Open Elective	2020-04-02	480	
Proof Reading	Open Elective	2020-04-02	60	
Create Presentation	Open Elective	2020-04-02	180	
Writing Technical Report	Open Elective	2020-04-01	360	
Writing Technical Report	Open Elective	2020-03-31	360	
Writing Technical Report	Open Elective	2020-03-30	360	
Sample Tool or Prepare Presentation	Open Elective	2020-03-29	480	
Sample Tool or Prepare Presentation	Open Elective	2020-03-28	480	
Sample Tool or Prepare Presentation	Open Elective	2020-03-20	180	
Lesson 6	Open Elective	2020-03-08	60	
Lesson 7	Open Elective	2020-03-08	60	
Lesson 4	Open Elective	2020-03-07	180	
Lesson 5	Open Elective	2020-03-07	180	
Lesson 6	Open Elective	2020-03-07	180	
Lesson 3	Open Elective	2020-03-01	360	
Further Learning (Tkinter)	Open Elective	2020-03-01	60	Recherche Cairo
Meeting 2	Open Elective	2020-02-28	30	
Lesson 2	Open Elective	2020-02-23	120	
Pandas Introduction	Open Elective	2020-02-23	240	
Troubleshooting Python Installation	Open Elective	2020-02-15	240	
Troubleshooting Python Installation	Open Elective	2020-02-14	420	
Meeting 1	Open Elective	2020-02-07	30	
Troubleshooting Python Installation	Open Elective	2020-02-03	120	
Lesson 2	Open Elective	2020-02-02	180	
Pandas Introduction	Open Elective	2020-02-02	120	
Lesson 1	Open Elective	2020-02-01	420	
Troubleshooting Python Installation	Open Elective	2020-02-01	60	
Build a Chat app with Tkinter	Open Elective	2020-01-27	40	
Packaging and Distributing Executables	Open Elective	2020-01-27	30	
Wrapping Up	Open Elective	2020-01-27	10	
Build a Chat app with Tkinter	Open Elective	2020-01-26	30	
Build a Snake Game with Tkinter	Open Elective	2020-01-26	40	
Milestone Project: Pomodoro Timer	Open Elective	2020-01-25	80	

Task Name	Project	Date	Time (min)	Comment
Outline and Workplan	Open Elective	2020-01-23	90	
PyQt5	Open Elective	2020-01-20	75	
Tkinter Themes and Styles	Open Elective	2020-01-19	90	
PyQt5	Open Elective	2020-01-19	75	
PyQt5	Open Elective	2020-01-20	75	
Tkinter Themes and Styles	Open Elective	2020-01-19	90	
PyQt5	Open Elective	2020-01-19	75	
Further Learning (Tkinter)	Open Elective	2020-01-18	120	Tkinter and Canvas
Outline and Workplan	Open Elective	2020-01-17	30	
Kickoff Meeting	Open Elective	2020-01-17	30	
Tkinter Widget Reference	Open Elective	2020-01-12	120	
Milestone Project: Distance Converter	Open Elective	2020-01-12	60	
Object-Oriented Programming with Tkinter	Open Elective	2020-01-12	120	
Outline and Workplan	Open Elective	2020-01-11	90	
Letter of Intent/Proposal	Open Elective	2020-01-10	120	
Creating First Tkinter App	Open Elective	2020-01-02	180	Christmas Break
Tkinter Widget Reference	Open Elective	2020-01-02	60	Christmas Break
Further Learning Python	Open Elective	2019-12-30	300	Christmas Break
Python Refresher	Open Elective	2019-12-30	300	Christmas Break
Letter of Intent/Proposal	Open Elective	2019-12-23	180	Christmas Break

A.3.2 Project Plan

GANTTPRO Open Elective

	Task name	Assigned	Estimation (Hours)	Time log	Progress
			113h	137h 50m	
1	Open Elective		113h	SUM 137h 5/ 0	100%
1.1	Reading		0	SUM 5h 0	100%
1.1.1	Further Learning Py…	Thoma…	0	5h	100%
1.2	Project Management		11h	SUM 10h 0	100%
1.2.1	Letter of Intent/Pro…	Thoma…	5h	5h	100%
1.2.2	Outline and Workplan	Thoma…	3h	3h 30m	100%
1.2.3	Kickoff Meeting	Thoma…	30m	30m	100%
1.2.4	Meeting 1	Thoma…	30m	30m	100%
1.2.5	Meeting 2	Thoma…	30m	30m	100%
1.2.6	Meeting 3	Thoma…	30m	0	100%
1.2.7	Meeting 4	Thoma…	30m	0	100%
1.2.8	Meeting 5	Thoma…	30m	0	100%
1.3	Online Course - GU…		27h	SUM 24h 50/ 0	100%
1.3.1	Python Refresher	Thoma…	4h	5h	100%
1.3.2	Creating First Tkint…	Thoma…	2h	3h	100%
1.3.3	Tkinter Widget Ref…	Thoma…	2h	3h	100%
1.3.4	Milestone Project: …	Thoma…	1h	1h	100%
1.3.5	Object-Oriented Pro…	Thoma…	4h	2h	100%
1.3.6	Tkinter Themes and…	Thoma…	2h	1h 30m	100%
1.3.7	Milestone Project: P…	Thoma…	2h	1h 20m	100%
1.3.8	Build a Chat app wi…	Thoma…	3h	1h 10m	100%
1.3.9	Build a Snake Gam…	Thoma…	3h	40m	100%
1.3.10	Packaging and Distr…	Thoma…	2h	30m	100%
1.3.11	Wrapping Up	Thoma…	2h	10m	100%
1.3.12	Further Learning (T…	Thoma…	0	3h	100%
1.3.13	PyQt5	Thoma…	0	2h 30m	100%
1.4	Online Course: Aut…		42h	SUM 49h 0	100%
1.4.1	Lesson 1	Thoma…	5h	7h	100%
1.4.2	Lesson 2	Thoma…	5h	5h	100%
1.4.3	Lesson 3	Thoma…	5h	6h	100%
1.4.4	Lesson 4	Thoma…	5h	3h	100%
1.4.5	Lesson 5	Thoma…	5h	3h	100%
1.4.6	Lesson 6	Thoma…	5h	4h	100%
1.4.7	Lesson 7	Thoma…	5h	1h	100%
1.4.8	Pandas Introduction	Thoma…	6h	6h	100%
1.4.9	Troubleshooting Pyt…	Thoma…	1h	14h	100%
1.5	Development		20h	SUM 19h 0	100%
1.5.1	Sample Tool or Pre…	Thoma…	20h	19h	100%
1.6	Technical Report		9h	SUM 27h 0	100%
1.6.1	Writing Technical R…	Thoma…	8h	26h	100%
1.6.2	Proof Reading	Thoma…	1h	1h	100%
1.7	Presentation		4h	SUM 3h 0	100%
1.7.1	Create Presentation	Thoma…	4h	3h	100%



B How to setup a Geo-Python Environment for Windows 10

B.1 Install Anaconda

Download Anaconda 3 from the internet and install Anaconda to `c:/apps/anaconda3`.

- Keep path simple
- Do not add Python to your 'PATH' environment variable (if already set by another application, remove all other Python versions from the 'PATH' environment variable)
- Register Anaconda as the system Python 3.7 (not 2.7!)

Sources

- `https://www.anaconda.com/distribution`
- `https://medium.com/@GalarnykMichael/install-python-on-windows-anaconda-c63c7c3d1444`

B.2 Install PyCharm for Anaconda

Note: You will need a licence for PyCharm. If you are student, sign up for the GitHub Student Pack and get a free Education licence.

Download (`https://www.jetbrains.com/pycharm/promo/anaconda`) and install PyCharm for Anaconda.

Sources

- `https://www.jetbrains.com/pycharm/promo/anaconda`
- `https://education.github.com/pack`

B.3 Install R (optional)

Download R 3.6.2 from the internet and install Anaconda to `c:/apps/R-3.6.2`.

Sources

- `https://cran.r-project.org/bin/windows/base`

B.4 Install QGIS (optional)

Download OSGeo4W Network Installer (64 bit) and install the tools with the Express Desktop Install to `c:/apps/OSGeo4W`.

- Select all packages (QGIS, GDAL, GRASS GIS)

Sources

- https://qgis.org/en/site/forusers/download.html

B.5 Prepare Conda Environment

B.5.1 Build Tools for Visual Studio

Note: This is required for certain packages.

Download and Install Build Tools for Visual Studio 2019. (Maybe an older version does work too.)

- Select "C++ build tools"

Sources

- https://visualstudio.microsoft.com/downloads

B.5.2 Setup Base Environment

Open the Anaconda Prompt (via Start Menu) and use following commands:

```
cd %userprofile%
conda config
conda config --add channels defaults
conda install geopandas jupyterlab
```

B.5.3 Setup GIS-Environment

Open the Anaconda Prompt (via Start Menu) and use following commands:

```
conda create -n gis python=3.7
conda activate gis

conda install jupyterlab
conda install gdal

conda install geopandas matplotlib mapclassify
conda install -c conda-forge geojson contextily folium mplleaflet osmnx
conda install pysal rasterio
conda install dill
conda install -c conda-forge geoplot rasterstats
```

```
conda install psycopg2
conda install sqlalchemy
conda install -c conda-forge geoalchemy2

pip install urbanaccess pandana
pip install dash==0.19.0
pip install dash-renderer==0.11.1
pip install dash-html-components==0.8.0
pip install dash-core-components==0.14.0
pip install plotly --upgrade
```

B.5.4 Test your Environment

Open the Anaconda Prompt (via Start Menu) type in:

```
python -c 'import gdal; print(dir(gdal))'
```

Start Python REPL in the Anaconda Prompt and execute following lines. No error should occur.

```python
import geopandas as gpd
import pysal
import cartopy
import geoplot
import osmnx
import folium
import dash
import rasterio
import osmnx
import contextily
import psycopg2
import sqlalchemy
import geoalchemy2
```

Sources

- https://automating-gis-processes.github.io/site/course-info/Installing_Anacondas_GIS.html
- https://github.com/ContinuumIO/anaconda-issues/issues/10351#issuecomment-528378258

B.5.5 Portable Conda

1. copy `c:/apps/anaconda3` to your USB stick (keep same path structure on USB stick, otherwise you'll need to update path in batch file)
2. create `conda-console.bat` (see below) and save it to the USB stick root directory
3. start your portable conda by double clicking `conda-console.bat`

Create a batch file with following content:

```
@echo off
title Conda Console
echo --------------------------------------------------------------
echo Welcome to your Conda Console
echo --------------------------------------------------------------
set /p lw="What's your drive letter? "
set /p env="Which conda environment do you want to use?"
set PATH=%lw%:\apps\anaconda3\condabin\;%PATH%
cd /D %userprofile%
cmd /K "conda activate %env%"
```

C How to setup a Geo-Python Environment for Ubuntu 18.04

C.1 Install ZSH Console (optional)

Follow instructions on:

- `https://kifarunix.com/how-to-install-and-setup-zsh-and-oh-my-zsh-on-ubuntu-18-04`

C.2 Install Anaconda

Change to your console and type in:

```
cd ~/Downloads
wget https://repo.anaconda.com/archive/Anaconda3-2019.10-Linux-x86_64.sh
bash Anaconda3-2019.10-Linux-x86_64.sh
```

Follow the instructions and install Anaconda to `~/anaconda3`. If you are asked to initialize Conda do it. After the installation you need to restart your console.

Check your Conda version with:

```
conda --version
```

C.2.1 Use Anaconda in a ZSH Console (optional)

Open `~/.zshrc` in a text editor, for example with nano:

```
nano ~/.zshrc
```

Add add the end of the file following two lines and restart your console.

```
.  ~/anaconda3/etc/profile.d/conda.sh
conda activate base
```

Add at the end of the file following two lines and restart your console.

Check your Conda version with:

```
conda --version
```

Sources

- `https://www.anaconda.com`
- `https://stackoverflow.com/a/52029602/42659`

C.3 Install PyCharm for Anaconda

Note: You will need a licence for PyCharm. If you are student, sign up for the GitHub Student Pack and get a free Education licence.

Change to your terminal and execute:

```
cd ~/Downloads
wget
→  https://download.jetbrains.com/python/pycharm-professional-anaconda-2019.3.3.tar.gz
sudo tar xfz pycharm-*.tar.gz -C /opt/
```

Start PyCharm with:

```
sh /opt/pycharm-anaconda-2019.3.3/bin/pycharm.sh
```

After PyCharm is started, go to `Tools -> Create Desktop Entry...` to create a link in your Application list.

Sources

- `https://www.jetbrains.com/pycharm/promo/anaconda`
- `https://www.jetbrains.com/help/pycharm/installation-guide.html`
- `https://askubuntu.com/a/108794`
- `https://education.github.com/pack`

C.4 Install R (optional)

Go to your terminal and type in:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
↪ E298A3A825C0D65DFD57CBB651716619E084DAB9
sudo add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu
↪ bionic-cran35/'
sudo apt update
sudo apt install r-base
```

Test R with: `sudo -i R` (you might need to type in your sudo password)

Sources

- `https://www.digitalocean.com/community/tutorials/how-to-install-r-on-ubuntu-18-04`

C.5 Install QGIS (optional)

Add following lines to `/etc/apt/sources.list`:

```
deb     https://qgis.org/debian bionic main
deb-src https://qgis.org/debian bionic main
```

Then execute following commands in your terminal:

```
wget -O - https://qgis.org/downloads/qgis-2019.gpg.key | gpg --import
gpg --fingerprint 51F523511C7028C3
gpg --export --armor 51F523511C7028C3 | sudo apt-key add -
sudo apt-get update
sudo apt-get install qgis qgis-plugin-grass
```

C.6 Prepare Conda Environment

C.6.1 Setup Base Environment

Go to your terminal and type in:

```
cd ~
rm .condarc
conda config
conda config --add channels defaults
conda install geopandas jupyterlab
```

C.6.2 Setup GIS Environment

Go to your terminal and type in:

```
conda create -n gis python=3.7
conda activate gis

conda install jupyterlab
conda install gdal

conda install geopandas matplotlib mapclassify
conda install -c conda-forge geojson contextily folium mplleaflet osmnx
conda install pysal rasterio
conda install dill
conda install -c conda-forge geoplot rasterstats
conda install psycopg2
conda install sqlalchemy
conda install -c conda-forge geoalchemy2

```

```
pip install urbanaccess pandana
pip install dash==0.19.0
pip install dash-renderer==0.11.1
pip install dash-html-components==0.8.0
pip install dash-core-components==0.14.0
pip install plotly --upgrade
```

C.6.3 Test your Environment

Go to your terminal and type in:

```
python -c 'import gdal; print(dir(gdal))'
```

Start Python REPL in your terminal and execute following lines. No error should occur.

```
import geopandas as gpd
import pysal
import cartopy
import geoplot
import osmnx
import folium
import dash
import rasterio
import osmnx
import contextily
import psycopg2
import sqlalchemy
import geoalchemy2
```

Sources

- https://automating-gis-processes.github.io/site/course-info/Installing_Anacondas_GIS.html
- https://github.com/ContinuumIO/anaconda-issues/issues/10351#issuecomment-528378258