Building a GUI for Seismic Data Aquisition using Python

Name: Sankarshan Mondal
2nd Year BSMS, IISER Kolkata
Summer 2023

Instructor: Dr. Dibakar Ghoshal
Assistant Professor, Department of Earth Sciences
IIT Kanpur

Kanpur, Uttar Pradesh
India
June 22, 2023

Contents

Acknowledgements

I would like to express my sincere gratitude and appreciation to all those who have contributed to the successful completion of the project involving the development of a GUI for seismic data acquisition and processing using Python and the Tkinter module.
First and foremost, I would like to extend my heartfelt thanks to my supervisor, Dr. Dibakar Ghoshal, for his guidance, support, and valuable insights throughout the project. His expertise and constant encouragement played a crucial role in shaping the project and overcoming various challenges.
I am grateful to the institution, IIT Kanpur for providing the necessary resources, including access to seismic data, software, and computing infrastructure. These resources were vital for the development and testing of the GUI, ensuring its functionality and performance.
Furthermore, I extend my appreciation to the individuals who generously shared their knowledge and expertise during the course of this project. Their valuable insights and suggestions significantly influenced the design and functionality of the GUI, enhancing its usability and effectiveness.
In conclusion, this project has been a collaborative effort, and I am truly thankful to everyone who has been a part of it. Your contributions, guidance, and support have been invaluable, and I am grateful for the opportunity to work on such an exciting and meaningful project.

1 Introduction

The purpose of this report is to provide an overview of a project involving the development of a Graphical User Interface (GUI) for seismic data acquisition and processing. The GUI has been built using Python programming language, utilizing the Tkinter module. This report outlines the project's objectives, the methodology used, and the resulting functionalities of the GUI.

2 Objectives

The main objectives of this project were as follows: - Develop a user-friendly GUI for seismic data acquisition and processing. - Implement functionalities for data acquisition from seismic sensors. - Generate four different output files containing processed seismic data.

3 Methodology

The project utilized Python as the programming language due to its simplicity, versatility, and wide range of available libraries. Tkinter, a standard GUI toolkit for Python, was chosen for building the graphical interface. The project followed an iterative development approach, including the following steps:

3.1 GUI Design

The initial phase involved designing the graphical layout of the user interface. This included selecting appropriate widgets, such as buttons, labels, and text boxes, to facilitate user interaction. The GUI design aimed to be intuitive, ensuring ease of use for users with varying levels of technical expertise.

3.2 Data Acquisition

The next step was to implement functionalities for data acquisition from seismic sensors. The GUI provided options to select the sensor, set acquisition parameters (e.g., sample rate, duration), and initiate data capture. The acquired data was stored in memory for further processing.

3.3 File Generation

The final step involved generating four different output files containing the seismic data. These files were saved in four formats- sps, rps, xps and txt to ensure compatibility with other applications. Each file contained specific information, such as Shot data, Receiver data, relation between shot and receiver and timestamp data, depending on the user's requirements.

4 Functionality of the GUI

The developed GUI offered the following key functionalities:

1. Data Acquisition
   (a) Setting acquisition parameters (sample rate)
   (b) Initiating data capture
2. File Generation
   (a) Four output files containing seismic data in different formats
      - *.sps (Shot Data)
      - *.rps (Receiver Data)
      - *.xps (Relation Data)
      - *.txt (Time Data)

   (b) Options to generate the specific data to be included in each file

5 Conclusion

In conclusion, the project successfully developed a GUI for seismic data acquisition using Python and the Tkinter module. The GUI provided a user-friendly interface for acquiring data from seismic sensors. It allowed the generation of three output files containing the seismic data. This project serves as a valuable tool for researchers, geophysicists, and professionals working with seismic data, enabling efficient data analysis and reporting. Further enhancements and features can be added to the GUI to meet specific user requirements in the future.

6 Demonstration of the GUI

The GUI can be used when we have four files in the same directory. In other words it is of the form of a folder containing three files and a two folders. They are the following:

1. **python_3.py**
2. **header**: This is a folder containing the following files:
   (a) receiver.txt
   (b) relation.txt
   (c) shot.txt
3. **plot**: This folder is required for the graphical representation. If this folder is not present, please create an empty folder with the name - 'plot'.

The file inputs we need are the the time files in **.csv** format (preferably).

The step by step way to use the GUI is as follows:

Step 1

Open the terminal. Change the directory to the folder by using the following command:

```
cd ~/Desktop/sankarshan_2023
```

First run the python file in the terminal using python command:

```
python3 project_3.py
```

You will observe a GUI which looks like this:

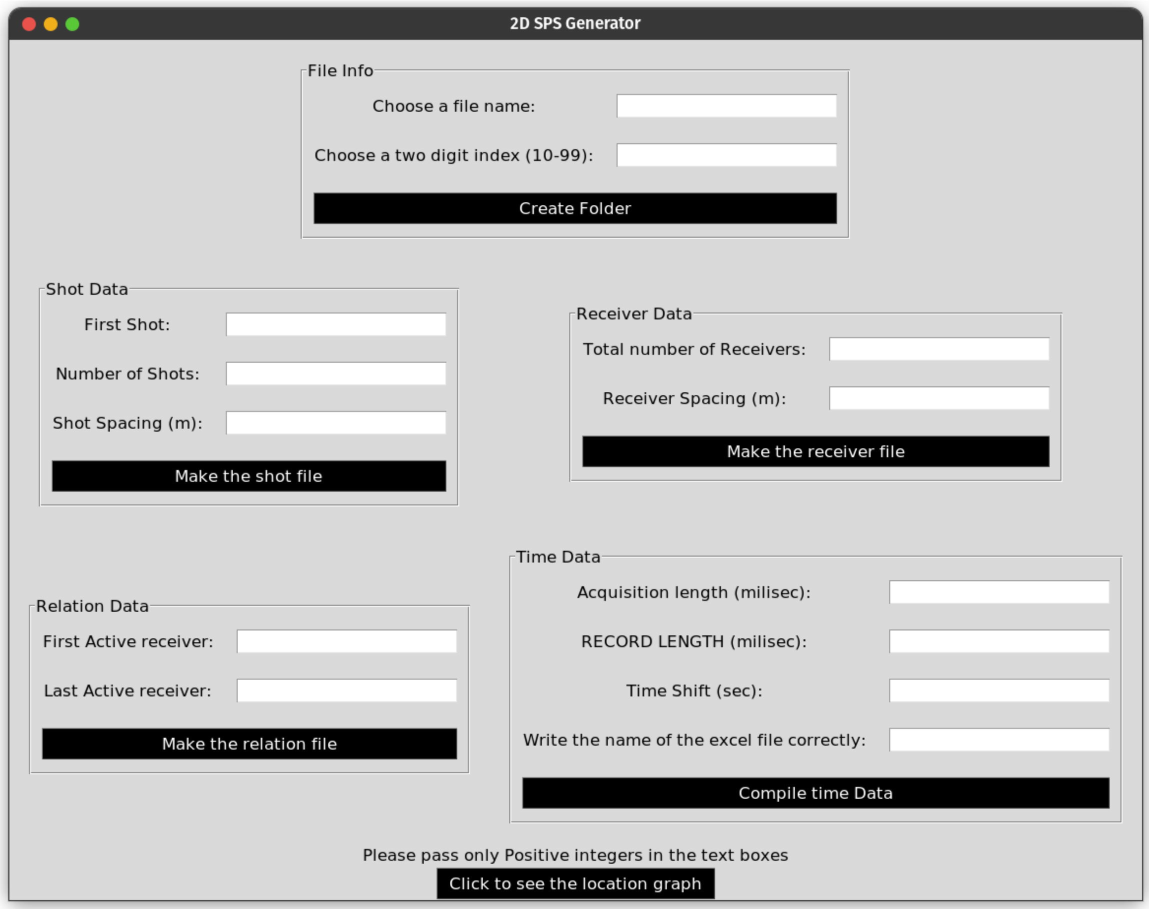

Figure 1: The GUI

Step 2

Start filling up the data in the following order:

1. File Info
2. Shot Data
3. Receiver Data
4. Relation Data
5. Time Data

Step 3

part 1

Fill up the 'File Info' box and click the **Create Folder** button. This will create a folder with the filename entered. It will contain an empty folder named **csv**.

The time files should be of the form *timestamp*.csv where *timestamp* is the name of the file with **csv** extension. After making the folder with the 'File Info' box, carefully move the *timestamp*.csv file corresponding to that dataset to a folder called **csv**. This folder will be formed inside the folder formed with the name, **filename**.

part 2

Click the respective button in every box after filling up the information in a box. Do not worry if you fill up any wrong information by mistake. Correct the value in the textbox and click the button. Your wrong value will be overwritten.

part 3

Fill up the name of the csv file carefully (complete with the extension). Also make sure that the time file corresponds to the number/index of the shot and receiver combination used i.e. be careful in moving the respective csv files to their respective folders.

part 4

For the filename, you can not create folder with the same file name multiple times. To use the same file name, you have to manually remove the existing folder with the same name. Or else the data inside the folder will be overwritten with the new data input. If you are willing to change any data in a folder you have already created, just type the name of the folder, and the index in the 'File Info' box and start filling out rest of the boxes, this would do the job.

Step 4

Once you have filled up all the boxes you will see a folder (by the name entered as input) in the directory. This folder consists of four new files of the following formats:

1. *.sps (Shot Data)
2. *.rps (Receiver Data)
3. *.xps (Relation Data)
4. *.txt (Time Data)

It would also have a folder named **csv**. Inside **csv**, we would have the original *timestamp*.csv file.

We can use this folder (which has the same name as the files) for analysis. It is good idea to transfer this folder to a different directory.

Step 5

There is a button at the very end which says : **Click to see the location graph**. Clicking this will show you a graph like the following:

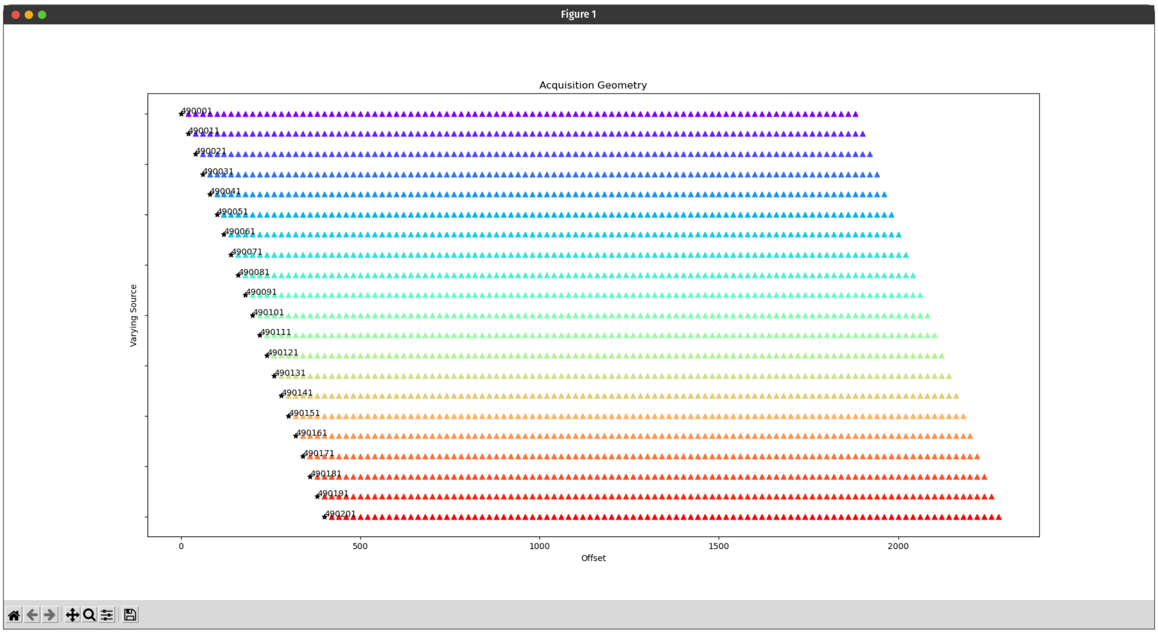

Figure 2: The interactive plot

This interactive plot shows the position of source and receiver during a single measurement. The **star** sign represents the source, the **triangle** represents the receivers. Each measurement is represented by different colours. The source is also indexed according to the Shot Data File.

7 Source Code

The total code used for the GUI is as follows:

```
from tkinter import *
from PIL import Image, ImageTk
import pandas as pd
import os
import numpy as np
from datetime import datetime
import time
import matplotlib.pyplot as plt
import matplotlib.cm as cm



root = Tk()
root.title("2D SPS Generator")
root.iconphoto(False, PhotoImage(file = 'seismo.png'))


frame0 = LabelFrame(master=root, text="File Info")
frame0.grid(row=0, column=0, columnspan=2, pady=20, padx=20)

label_name = Label(frame0, text="Choose a file name:")
label_name.grid(row=0, column=0, pady=12, padx=10, sticky='ew')

l_n_e = Entry(frame0)
l_n_e.grid(row=0, column=1, pady=12, padx=10, sticky='ew')

index = Label(frame0, text="Choose a two digit index (10-99):")
index.grid(row=1, column=0, pady=12, padx=10, sticky='ew')

index = Entry(frame0)
index.grid(row=1, column=1, pady=12, padx=10, sticky='ew')

def f():
    os.mkdir("{}".format(l_n_e.get()))
    os.mkdir("{}/csv".format(l_n_e.get()))


global dir
dir = os.getcwd()

button0 = Button(frame0, text="Create Folder", command=f, bg="black", fg="white")
button0.grid(row=2, column=0, columnspan=2, pady=12, padx=10, sticky='ew')





frame1 = LabelFrame(master=root, text="Shot Data")
frame1.grid(row=1, column=0, pady=20, padx=20)

frame2 = LabelFrame(master=root, text="Receiver Data")
frame2.grid(row=1, column=1, pady=20, padx=20)

frame3 = LabelFrame(master=root, text="Relation Data")
frame3.grid(row=2, column=0, pady=20, padx=20)

frame4 = LabelFrame(master=root, text="Time Data")
frame4.grid(row=2, column=1, pady=20, padx=20)


label = Label(root, text="Please pass only Positive integers in the text boxes")
label.grid(row=3, column=0, columnspan=2, sticky='ew')

f_s = Label(frame1, text="First Shot:")
```

```
f_s.grid(row=0, column=0, pady=12, padx=10)

f_s_e = Entry(frame1)
f_s_e.grid(row=0, column=1, pady=12, padx=10)


n_s = Label(frame1, text="Number of Shots:")
n_s.grid(row=1, column=0, pady=12, padx=10)

n_s_e = Entry(frame1)
n_s_e.grid(row=1, column=1, pady=12, padx=10)


d_s = Label(frame1, text="Shot Spacing (m):")
d_s.grid(row=2, column=0, pady=12, padx=10)

d_s_e = Entry(frame1)
d_s_e.grid(row=2, column=1, pady=12, padx=10)


n_r = Label(frame2, text="Total number of Receivers:")
n_r.grid(row=0, column=0, pady=12, padx=10)

n_r_e = Entry(frame2)
n_r_e.grid(row=0, column=1, pady=12, padx=10)


d_r = Label(frame2, text="Receiver Spacing (m):")
d_r.grid(row=1, column=0, pady=12, padx=10)

d_r_e = Entry(frame2)
d_r_e.grid(row=1, column=1, pady=12, padx=10)

#=====================================================================================

def shot_f(x,y):


    #Take into input Shots
    shot = int(x)
    shot+=1
    #We form a list to index the shots. This way we would be able to track the data

    global shot_n
    shot_n=[]
    for i in range(shot):
        num = str(int(index.get())*1000+i+int(f_s_e.get()))
        shot_n.append(num + "1E1")


    #Lattitude
    #We don't take the lattitude from a file and store it into a list. Ordered values are preferred
    lat = [0.0 for i in range(shot)]


    #Longitude
    #We don't take the longitude from a file and store it into a list. Ordered values are preferred
    long = [0.0 for i in range(shot)]




    #Elevation
    #We don't take the elevation from a file and store it into a list. Ordered values are preferred
    elev = [0.0 for i in range(shot)]
```

```
    #We also obtain the shot value from a file
    #example file

    #shot offset due to shot spacing

    shot_v = float(y)


    #We convert it into float as this is the last entry in the row and we want to
    move to the next row.
    so = []
    for x in range(shot):
        n = 0 + x*shot_v
        so.append(n)

    #thing for formatting
    S1 = ["S1" for i in range(shot)]
    space1 = [" "*16 for i in range(shot)]
    #index
    space2 = [" "*1 for i in range(shot)]
    zero1 = ["0" for i in range(shot)]
    #lattitude
    space3 = [" "*1 for i in range(shot)]
    zero2 = ["0 0" for i in range(shot)]
    #longitude
    space4 = [" "*2 for i in range(shot)]
    #offset
    space5 = [" "*5 for i in range(shot)]
    #elevation
    space6 = [" " for i in range(shot)]
    #some zeroes
    number = [" 1235959" for i in range(shot)]



    df = pd.DataFrame({"S1":S1, "Space1":space1,"Shot_no": shot_n, "space2":space2, "
    zero1":zero1, "Lattitude": lat, "space3":space3, "zero2":zero2, "Longitude": long
    , "space4":space4, "Shot_offset": so, "space5":space5, "Elevation": elev, "space6
    ":space6, "zeroes":elev, "number":number})


    with open("header/shot.txt") as f:
        os.chdir("{}".format(l_n_e.get()))
        with open("{}.sps".format(l_n_e.get()), "w") as f1:
            for line in f:
                f1.write(line)
            f1.write("\n")
            df_string = df.to_string(header=False, index=False)
            f1.write(df_string)
        os.chdir(dir)

    with open('plot/data_s_plot.txt', 'w') as f:
        df_string = df.to_string(header=False, index=False)
        f.write(df_string)

#==================================================================================

def receiver_f(x,y):


    #Take into input Shots
    shot = int(x)

    #We form a list to index the shots. This way we would be able to track the data
    s=[]
```

```python
    for i in range(1,shot+1):
        num = str(2000+i)
        s.append(num + "1G1")


    #Lattitude
    #We take the lattitude from a file and store it into a list. Ordered values are preferred

    lat = [0.0 for i in range(shot)]


    #Longitude
    #We take the longitude from a file and store it into a list. Ordered values are preferred

    long = [0.0 for i in range(shot)]




    #Elevation
    #We take the elevation from a file and store it into a list. Ordered values are preferred

    elev = [0.0 for i in range(shot)]


    #We also obtain the shot value from a file
    #example file

    #shot offset due to shot spacing

    re_v = float(y)


    #We convert it into float as this is the last entry in the row and we want to move to the next row.
    re = []
    for x in range(shot):
        n = 0 + x*re_v
        re.append(n)

    #thing for formatting
    S1 = ["R1" for i in range(shot)]
    space1 = [" "*17 for i in range(shot)]
    #index
    space2 = [" "*1 for i in range(shot)]
    zero1 = ["0" for i in range(shot)]
    #lattitude
    space3 = [" "*1 for i in range(shot)]
    zero2 = ["0 0" for i in range(shot)]
    #longitude
    space4 = [" "*2 for i in range(shot)]
    #offset
    space5 = [" "*5 for i in range(shot)]
    #elevation
    space6 = [" " for i in range(shot)]
    #some zeroes
    number = [" 1235959" for i in range(shot)]



    df = pd.DataFrame({"S1":S1, "Space1":space1,"Shot_no": s, "space2":space2, "zero1":zero1, "Lattitude": lat, "space3":space3, "zero2":zero2, "Longitude": long, "space4":space4, "Receiver_offset": re, "space5":space5, "Elevation": elev, "space6":space6, "zeroes":elev, "number":number})

```

```

    with open("header/reciever.txt") as f:
        os.chdir("{}".format(l_n_e.get()))
        with open("{}.rps".format(l_n_e.get()), "w") as f1:
            for line in f:
                f1.write(line)
            f1.write("\n")
            df_string = df.to_string(header=False, index=False)
            f1.write(df_string)
        os.chdir(dir)


    with open('plot/data_r_plot.txt', 'w') as f:
        df_string = df.to_string(header=False, index=False)
        f.write(df_string)

#==========================================================================================

def relation(x,y):



    #Importing the shot data from a random column
    df_s = pd.read_fwf("plot/data_s_plot.txt", usecols=[1], names=['index'])
    #print(df_s)

    #Importing the reciever data from a rqandom column
    df_r = pd.read_fwf("plot/data_r_plot.txt", usecols=[1], names=['index'])
    #print(df_r)

    #Selecting the starting reciever on the basis of data obtained from shot position and reciever position

    p = [x for x in df_r["index"]]
    q = [x for x in df_s["index"]]

    #print(p,q)

    #using loops to find where the shot data matches the reciever

    #the first active receiver

    print("Press enter if all recievers are working")
    active = int(x)
    working = int(y)
    if 0<active<len(p):
        if active<working<len(p):
            p = p[active-1:working]
        else:
            label = Label(root, text="Use number between {} and {}".format(active, len(p)))
            label.grid(row=3, column=0, columnspan=2)
    else:
        label = Label(root, text="Too large number")
        label.grid(row=3, column=0, columnspan=2)
    print("wow")



    shot = len(q)
    #defining indices for measurement
    s=[]
    for i in range(shot):
        num = str(int(index.get())*1000+i+int(f_s_e.get()))
        s.append(num + "1")
    print(s)

    #starting reciever
```

```
    r_s = [active for i in s]
    #print(r_s)

    #last reciever
    r_e1 = [working for i in s]
    r_e = []
    for i in r_e1:
        n = " {x}11".format(x = i)
        r_e.append(n)

    #print(r_e)


    #exact starting reciever
    r_s1 = []
    for i in range(active, active+shot):
        n = 2000 + i
        r_s1.append(n)
    #print(r_s1)

    #exact ending reciever
    r_s2 = []
    for i in range(working, working+shot):
        n = 20000 + i*10 + 1
        r_s2.append(n)
    #print(r_s2)



    #things for formatting the text file
    R1 = ["X0" for i in range(shot)]
    space1 = [" "*7+"0111"+" "*17 for i in range(shot)]
    #index
    space2 = [" "*1 for i in range(shot)]
    #starting_receiver
    #last receiver
    space3 = [" "*17 for i in range(shot)]
    #reciver_start
    space4 = [" "*2 for i in range(shot)]
    #reciver_end




    #forming the dataframe by pandas
    dataset = {"X0":R1, "space1":space1, "Index":s, "space2":space2, "Start": r_s,  "
End": r_e, "space3":space3, "start_reciever_ID":r_s1, "space4":space4, "
end_reciever_ID":r_s2}
    dataframe = pd.DataFrame(dataset)


    with open("header/relation.txt") as f:
        os.chdir("{}".format(l_n_e.get()))
        with open("{}.xps".format(l_n_e.get()), "w") as f1:
            for line in f:
                f1.write(line)
            f1.write("\n")
            df_string = dataframe.to_string(header=False, index=False)
            f1.write(df_string)
        os.chdir(dir)


    with open("plot/data_relation.txt", "w") as f1:
            df_string = dataframe.to_string(header=False, index=False)
            f1.write(df_string)

#==================================================================================

```

```python

button1 = Button(frame1, text="Make the shot file", command=lambda: shot_f(n_s_e.get
    (),d_s_e.get()), bg="black", fg="white")
button1.grid(row=3, column=0, columnspan=2, pady=12, padx=10, sticky='ew')



button2 = Button(frame2, text="Make the receiver file", command=lambda: receiver_f(
    n_r_e.get(),d_r_e.get()), bg="black", fg="white")
button2.grid(row=2, column=0,  columnspan=2, pady=12, padx=10, sticky='ew')



relation1 = Label(frame3, text="First Active receiver:")
relation1.grid(row=0, column=0, pady=12, padx=10, sticky='ew')

relation1_e = Entry(frame3)
relation1_e.grid(row=0, column=1, pady=12, padx=10, sticky='ew')


relation2 = Label(frame3, text="Last Active receiver:")
relation2.grid(row=1, column=0, pady=12, padx=10, sticky='ew')

relation2_e = Entry(frame3)
relation2_e.grid(row=1, column=1, pady=12, padx=10, sticky='ew')



button3 = Button(frame3, text="Make the relation file", command=lambda: relation(
    relation1_e.get(), relation2_e.get()), bg="black", fg="white")
button3.grid(row=2, column=0, columnspan=2, pady=12, padx=10, sticky='ew')

#==============================================================================

label1 = Label(frame4, text="Acquisition length (milisec):")
label1.grid(row=0, column=0, pady=12, padx=10, sticky='ew')

label1_e = Entry(frame4)
label1_e.grid(row=0, column=1, pady=12, padx=10, sticky='ew')


label2 = Label(frame4, text="RECORD LENGTH (milisec):")
label2.grid(row=1, column=0, pady=12, padx=10, sticky='ew')

label2_e = Entry(frame4)
label2_e.grid(row=1, column=1, pady=12, padx=10, sticky='ew')

label3 = Label(frame4, text="Time Shift (sec):")
label3.grid(row=2, column=0, pady=12, padx=10, sticky='ew')

label3_e = Entry(frame4)
label3_e.grid(row=2, column=1, pady=12, padx=10, sticky='ew')


timefile = Label(frame4, text="Write the name of the excel file correctly:")
timefile.grid(row=3, column=0, pady=12, padx=10, sticky='ew')

timefile_e = Entry(frame4)
timefile_e.grid(row=3, column=1, pady=12, padx=10, sticky='ew')


def time_f():

    df = pd.read_csv(l_n_e.get() + '/csv/' + timefile_e.get(), sep=",")

    a = ["1/6/1980" for i in range(len(df.index))]
    print(a)

```

```
    df["fix"] = a

    df["fix"] = pd.to_datetime(df["fix"])
    df["Time"] = pd.to_datetime(df["Time"])

    print(df['fix'])


    df["diff"] = ((df["Time"] - df["fix"]).dt.total_seconds() + int(label3_e.get()))*10**6
    print(df["diff"])



    description = ["Impulsive" for i in range(len(df.index))]
    src_l = [1 for i in range(len(df.index))]
    src_s = []
    for i in range(len(df.index)):
        num = str(int(index.get())*1000+i+int(f_s_e.get()))
        src_s.append(num + "1")

    ffid = []
    for i in range(len(df.index)):
        num = str(int(index.get())*1000+i+int(f_s_e.get()))
        ffid.append(num + "1")

    acq = [label1_e.get() for i in range(len(df.index))]
    rec = [label2_e.get() for i in range(len(df.index))]

    sweep = [0 for i in range(len(df.index))]
    blast = [6 for i in range(len(df.index))]
    exit_s = [1024 for i in range(len(df.index))]
    exit_t = [0,0.3,0.2,0.9]+ [0.2 for i in range(len(df.index)-4)]




    df1 = pd.DataFrame({"DESCRIPTION":description, "GPS TIMESTAMP":df["diff"], "SRC LINE":src_l, "SRC STATION":src_s, "FFID":ffid, "ACQ LENGTH":acq, "PROCESS TYPE":src_l, "STACKING FOLD":src_l, "ACQ NUMBER":src_l, "RECORD LENGTH":rec, "SWEEP LENGTH":sweep, "AUTOCORREL PEAK":sweep, "CORREL TRACE NUMBER":sweep, "TYPE OF DUMP":sweep, "SOURCE TYPE":src_l, "UPHOLE TIME":sweep, "BLASTER ID":blast, "BLASTER STATUS":sweep, "EXT HEADER SIZE":exit_s, "EXT HEADER TEXT":exit_t})

    os.chdir("{}".format(l_n_e.get()))
    df1.to_csv("{}.txt".format(l_n_e.get()), sep=",")
    os.chdir(dir)

Button4 = Button(frame4, text="Compile time Data", command=time_f, bg="black", fg="white")
Button4.grid(row=4, column=0, columnspan=2, pady=12, padx=10, sticky='ew')

def plot_g():

    df_s = pd.read_csv("plot/data_s_plot.txt", sep = "\s+", header=None)

    df_r = pd.read_csv("plot/data_r_plot.txt", sep = "\s+", header=None)

    #df_rel = pd.read_csv("data_relation.txt", sep = "\s+", header=None)


    a = int(relation2_e.get())
    b = int(relation1_e.get())

    source = np.array(df_s[7])
    receiver = np.array(df_r[7])

    print(source)
```

```
    x = np.arange(len(source))
    ys = [i+x+(i*x)**2 for i in range(len(source))]

    colors = cm.rainbow(np.linspace(0, 1, len(ys)))



    ax = plt.gca()
    i = 0
    for j, k in zip(range(len(source)), colors):
        plt.scatter(receiver[b-1+j+1:b-1+a+j-1], [-source[j] for i in range(b+1, a)], marker="^", color=k)
        plt.scatter(source[j], -source[j], marker="*", color='black')
        plt.text(source[j], -source[j], shot_n[i][:len(shot_n[i])-2])
        i+=1

    plt.ylabel("Varying Source")
    ax.axes.yaxis.set_ticklabels([])
    plt.title("Acquisition Geometry")
    plt.xlabel("Offset")
    plt.show()

button5 = Button(root, text="Click to see the location graph", command=plot_g, bg="black", fg="white")
button5.grid(row=4, column=0, columnspan=2)


root.mainloop()
```

THANK YOU