FACULTAD 50 AÑOS DE INGENIERÍA UNaM
JIDeTEV 2024 ISSN 2591-4219
XIV Jornadas de Investigación, Desarrollo Tecnológico, Extensión, Vinculación y Muestra de la Producción

# Desarrollo de software de análisis de datos de voltamperometría cíclica en Python con interfaz gráfica

Cristian J. Quiñones [a], Leandro E. Antunez [a], Matías G. Krujoski [a,b], Javier E. Kolodziej [a,b]

[a] *Universidad Nacional de Misiones, Facultad de Ingeniería, Oberá, Misiones, Argentina.*
[b] *Instituto de Materiales de Misiones, UNaM-CONICET, Posadas, Misiones, Argentina.*

e-mails: quinonescristianj13@gmail.com, antunezleandroe@gmail.com, matias.krujoski@fio.unam.edu.ar, javier.kolodziej@fio.unam.edu.ar

***Resumen***

Este informe describe el desarrollo de un software en Python para el análisis de datos a partir de archivos CSV (*Comma Separated Values*). Se detallan la ejecución del programa, sus funcionalidades y los conflictos encontrados durante su desarrollo. Se abordan las librerías utilizadas, el manejo de datos, el desarrollo de la interfaz gráfica y la creación del archivo compilado para su distribución y uso. Además, se especifica el uso correcto del programa, los sistemas operativos compatibles y la interpretación de la información mostrada en pantalla. Finalmente, se analizan las estrategias para abordar problemas y errores en diversos aspectos, como la interfaz, la transferencia de datos de los CSV a los *data frames* y la representación gráfica de mediciones con fallos.

***Palabras Clave*** – *Compilado, CSV, Datos, Errores, Interfaz, Librerías, Parseo, Python*

## 1. Introducción

El planteo del análisis de datos medidos conlleva una gran cantidad de variables que se deben tener en cuenta en cada proyecto. Por un lado, existen factores generales que se pueden implementar en situaciones genéricas donde un problema puede ocurrir independiente del entorno específico para el cuál se crea la solución al manejo de dichos datos. Por otro lado, existen inconvenientes y observaciones a las que se deben prestar atención referidas al contexto específico de cada uno, donde problemas y opciones variarán según el dispositivo de medición, el formato de los datos medidos, o incluso las herramientas (como librerías) que se utilicen para el parseo de los mismos.

En nuestro caso particular, utilizamos el lenguaje de programación Python para el estudio de la información recaudada de cada medición debido a las comodidades que brinda al ser un lenguaje de alto nivel. Además, nos valimos de distintas librerías para su implementación, como Pandas y Numpy para el parseo y la conversión general de los datos; Matplotlib para los gráficos utilizados; Tkinter para la creación y diseño de la interfaz gráfica de usuario. Algunas situaciones destacadas ocurrieron en base a estas herramientas específicas, mientras que otras más generales como la interpretación de los datos, la creación y modificación de archivos básicos de Python pueden surgir en cualquier ámbito.

El objetivo y la función principal del software es mostrar los datos en forma directa y cómoda para su análisis en el contexto del problema al que pertenecen. Partimos de tres formatos distintos de CSVs, donde todos cuentan con valores de corriente y tensión correspondientes a ensayos de voltamperometría cíclica, algunos de los formatos de CSV cuentan con datos extra como los

* *quinonescristianj13@gmail.com*

parámetros de configuración del dispositivo de medición o los intervalos de tiempo entre cada medida. En base a esto el código fue implementado para generalizar estos tres formatos en uno genérico, salvaguardando aquellos trozos de interés para mostrarlos en pantalla en una distribución intuitiva y fácil de leer. Finalmente, se busca realizar algunos cálculos como pueden ser valores mínimos y máximos, medias, varianzas, etcétera. Todo compilado en un único archivo portable que contiene todas las dependencias y funciones necesarias, diseñado para funcionar en distintos sistemas operativos.

## 2. Desarrollo

### *2.1. Selección de formatos*

El formato de los archivos con los que se trabaja en el programa es CSV, es decir, varias columnas separadas por coma (u otro separador diferente) con un determinado encabezado que define la variable medida. En total, se utilizan tres modelos distintos que poseen distintas disposiciones (más o menos columnas, parámetros iniciales, etcétera). El código diferencia cada uno según sus encabezados, como puede apreciarse en la Fig. 1.

FORMATO "1".

```
CheckPoint5 03:01 p. m.,,,,,,,Measurement Parameters Settings:,Cyclic Voltammetry
Time (s), Current (nA), Time (s),Stimulus (mV),,,,Initial Potential,-1000,mV
0,-6764,181640625,0,-999,8046875,,,,Final Potential,1000,mV
0,0120011996477842,-6175,28173828125,0,0120011996477842,-998,730224609375,,,,Step Potential,0,6,
0,0240023992955685,-5969,3408203125,0,0240023992955685,-997,655700683594,,,,Scan Rate,50,mV/s
0,0360035970807076,-5819,1025390625,0,0360035970807076,-996,581237792969,,,,Current Range,± 9 µA
0,0480047985911369,-5694,57421875,0,0480047985911369,-995,506774902344,,,,Calibration Resistor,1
0,0600060001015663,-5585,845703125,0,0600060001015663,-993,89501953125,,,,RLoad,50 Ω,
0,0720071941614151,-5492,650390625,0,0720071941614151,-992,820556640625,
```

FORMATO "2".

```
CheckPoint2 09:15 a. m.,,,,,Measurement Parameters Settings:,Cyclic Voltammetry
Stimulus (mV),Current (µA),,,,Initial Potential,-1000,mV
-999,8046875,-245,566024780273,,,,Final Potential,1000,mV
-994,969543457031,-80,5068435668945,,,,Step Potential,2,5,mV
-990,134338378906,-40,6737327575684,,,,Scan Rate,40,mV/s
-985,299194335938,-24,6494216918945,,,,Cycle Count,5,
-979,9267578125,-17,3202686309814,,,,Current Range,± 150 µA,
-975,091613769531,-13,4761838912964,,,,Calibration Resistor,10000,Ω
-970,256469726563,-11,3045835494995,,,,RLoad,100 Ω,
-964,884033203125,-9,92106342315674,,,,PGA Gain Select,GNPGA_1,
-960,048889160156,-9,05417442321777,
```

FORMATO "3".

```
Index   X/V Y/µA
3   -0.994964599609375   -50.8524894714355
```

**Fig. 1. Detalle de la estructura de los tres modelos de formatos CSV.**

Debido al alto grado de especificidad del programa, la elección del formato se realiza de manera simple, detectando los encabezados de la segunda fila (Fig. 1) podemos diferenciar claramente los tres tipos existentes. El formato "1" contiene encabezados de tiempo, corriente y estímulo de voltaje además de los parámetros iniciales a la derecha. El formato "2" es similar, pero no posee valores de tiempo. Finalmente, el formato "3" es el más distintivo, en este, los valores de la tensión y la corriente están expresadas de otra forma, además, cuenta con una columna de índice (que más adelante se verá por qué es irrelevante) y su carácter de separación es una tabulación en lugar de una coma. Con esta información simplemente comparamos subcadenas que contengan la información deseada. Por ejemplo, si deseamos definir si un archivo corresponde al formato "2" buscamos si el parámetro "*Current*" se encuentra en la segunda fila del archivo mientras que este no posea la cadena "*Time*".

### *2.2. Características específicas de las mediciones del dispositivo*

Antes de comenzar a parsear mediante el uso de Pandas [1], es importante notar varias situaciones que se pueden presentar cuando los CSVs se crean sin mucha precaución. Si intentásemos crear un *data frame* con estos valores en crudo, surgirían varios problemas.

Primero, tanto el formato "1" como el "2" tienen un desfase de encabezados, como se puede observar en la Fig. 1, la configuración de los parámetros de medición se encuentran en la fila 1 mientras que el resto de encabezados está en la fila 2.

Segundo, los mismos formatos poseen una cantidad de comas innecesarias que buscan diferenciar los parámetros de las mediciones tomadas. Si intentásemos crear el *data frame*, crearíamos varias columnas vacías sin datos ya que se detectaría cada coma como una columna individual.

Tercero, el dispositivo que registró las mediciones (en los mismos formatos anteriores), separa los decimales con "," en lugar ".". Esto genera otro problema, ya que al crear el data frame, se crearán dos columnas distintas para los valores enteros y sus partes decimales.

Cuarto, el formato número "1" cuenta con un duplicado de la columna de tiempo.

Para solucionar todos estos inconvenientes surgidos de la diferencia de formatos, antes de utilizar las herramientas de Pandas, se modifican los archivos CSVs como si fuesen simples archivos de texto. Por un lado, se eliminan las comas en exceso para evitar que se creen columnas innecesarias. Por otro lado, se utiliza un bucle para recorrer las primeras diez filas del archivo, así podemos copiar mediante *slicing* las subcadenas que poseen los datos de los parámetros y guardarlas en un archivo temporal aparte (se observa un ejemplo en la Fig. 2). Este archivo se creará automáticamente durante la ejecución del programa y luego se utilizará para mostrar los parámetros en pantalla.

* *quinonescristianj13@gmail.com*

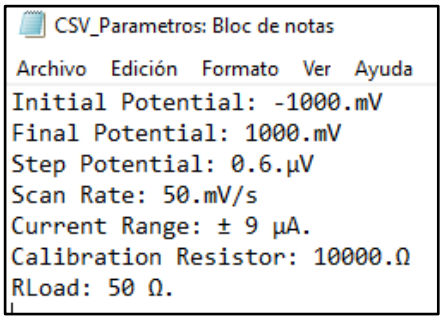


**Fig. 2. Archivo de parámetros de medición.**

### *2.3. Manejo de datos mediante el uso de Pandas*

Una vez que tenemos un archivo limpio, sin parámetros y con los encabezados correctamente ubicados, podemos utilizar Pandas para crear el *data frame*. En el estado actual, si procediésemos sin más modificaciones, se crearían las columnas de los decimales separadas de sus enteros y las columnas duplicadas del tiempo. Para evitar lo primero, utilizamos *slicing* para enviar las filas al *data frame*. Tras utilizar split(‘,’) para separar las filas en elementos dentro de una lista, podemos usar un bucle que aplique el método append() a una lista que contendrá la fila modificada mediante la concatenación de los elementos de la lista inicial con saltos de dos. Así, se unirán las columnas separadas en pares, teniendo en cuenta que cada valor debería tener su decimal, deberíamos tener seguridad sobre esto.

```
def parse_line(line):
    if format_type == 1 or format_type == 2 or format_type == 3:
        parts = line.split(',')
    else:
        raise ValueError ("Formato desconocido.")
    cont_exceptios=0
    parsed_line = []
    if format_type == 1 or format_type == 2:
        for i in range(0, len(parts)-1, 2):
            try:
                # Concatenar partes y convertir a float
                value = float(f"{parts[i]}.{parts[i+1]}")
                parsed_line.append(value)
```

**Fig. 3. Utilización de listas y *slicing* para la reescritura de las filas.**

Como se puede observar en la Fig. 3, “*line*” contiene la cadena original, tras detectar el formato previamente guardado, se recorre un bucle en saltos de dos desde 0 a la longitud de las partes (valores separados por coma guardados en una lista mediante el método split(“,”)) menos una unidad y se concatenan como flotante en el variable “*value*” que luego se guardará en la lista que contendrá los valores finales a parsear (“*parsed_line*”).

Este fue el método más eficiente que encontramos para evitar bucles innecesarios y posteriores modificaciones complejas al *data frame*. Son las listas "*parsed_line*" las que se utilizarán en Pandas, aun así, es importante notar que tras crearlo, debemos eliminar la columna duplicada, aunque es sumamente sencillo utilizando las funcionalidades específicas de Pandas (método drop() en este caso). El resultado es el siguiente:

**Modelo a:**

| | Tiempo | Corriente | Tensión |
|---|---|---|---|
| 0 | 0.0625 | 0.249789 | -994.969543 |
| 1 | 0.1250 | 0.248858 | -990.134338 |
| 2 | 0.1875 | 0.250254 | -985.299194 |
| 3 | 0.2500 | 0.250719 | -979.926758 |
| 4 | 0.3125 | 0.249789 | -975.091614 |
| .. | ... | ... | ... |
| 794 | 49.6875 | 0.248858 | -975.091614 |
| 795 | 49.7500 | 0.248393 | -979.926758 |
| 796 | 49.8125 | 0.247928 | -985.299194 |
| 797 | 49.8750 | 0.248858 | -990.134338 |
| 798 | 49.9375 | 0.248858 | -994.969543 |

**Modelo b:**

| | Tensión | Corriente |
|---|---|---|
| 0 | -994.969543 | -80.506844 |
| 1 | -990.134338 | -40.673733 |
| 2 | -985.299194 | -24.649422 |
| 3 | -979.926758 | -17.320269 |
| 4 | -975.091614 | -13.476184 |
| ... | ... | ... |
| 3994 | -975.091614 | -4.649677 |
| 3995 | -979.926758 | -4.597138 |
| 3996 | -985.299194 | -4.535844 |
| 3997 | -990.134338 | -4.509574 |
| 3998 | -994.969543 | -4.474549 |

**Fig. 4. *Data frame* finalizado (formato 1 [a] y formatos 2 y 3 [b]).**

El formato "3" se maneja de manera similar pero resulta más sencillo ya que este posee diferenciación entre los separadores de decimales y las mediciones. Tras reemplazar los encabezados por los nombres iguales a los otros formatos (con intención de generalizar uno base), se crea el *data frame* y se elimina la columna "*index*" ya que, como se puede ver en la Fig. 4, el propio *data frame* distingue entre filas con valores enteros. Además, se puede ver que el formato "1" posee una columna única con el contenido de los tiempos entre mediciones.

Como se puede notar, estos inconvenientes son específicos de nuestros datos, pero son situaciones que se podrían presentar siempre que el guardado de datos no esté correctamente programado en los instrumentos de mediciones. Finalmente, solo resta un último tipo de problema que puede surgir durante la carga de los datos desde los archivos CSV.

### *2.4. Datos dañados o perdidos y mediciones erróneas*

Existen situaciones en las que distintas anomalías pueden corromper una medición, o incluso la mala configuración del instrumento puede generar problemas. En nuestro caso, se nos presentó una situación de mala configuración.

Cuando se medía un valor entero, este se guardaba como tal. Al no guardarse como flotante (y teniendo en cuenta que los decimales se separan con comas), se perdía una coma que debía separar una columna en el *data frame*.

* *quinonescristianj13@gmail.com*

```
18  0,9375,0,250253796577454,0,9375,-925,128234863281,
19  1,0,250253796577454,1,-920,293090820313,
20  1,0625,0,250718951225281,1,0625,-914,920654296875,
```

**Fig. 5. Ejemplo de datos inconsistentes.**

En la Fig. 5 se presenta un ejemplo tomado del caso más desfavorable que nos encontramos, en uno de los CSVs; el intervalo de tiempo de muestreo es tal que genera valores enteros en cada unidad. Es decir, en cada valor entero de segundo, el archivo contiene una coma menos que en las otras líneas. Estas inconsistencias se podrían afrontar de distintas formas; podríamos haber eliminado esos datos erróneos o podríamos haber interpolado un punto utilizando el resto de datos que tenemos. Nosotros optamos por la última solución, principalmente con intención de reducir el perjuicio a los gráficos y evitar que se vean afectadas las funciones que requieren trabajo matemático como máximos, mínimos, varianza, entre otras.

Partiendo de la base que los puntos se grafican en un plano (estando los valores de tensión en el eje de abscisas y los de corriente en el eje de ordenadas), utilizamos interpolación lineal. Una recta une los puntos anterior y posterior al dato erróneo, y un punto estimado que se coloca entre ambos valores. Estos valores serán la tensión y corriente (x, y) .

```
15  0.9375,0.250253796577454,-925.128234863281
16  1.0,0.2504863739013675,-920.024444580078
17  1.0625,0.250718951225281,-914.920654296875
```

**Fig. 6. Conjunto de datos interpolados.**

Como se observa en la Fig. 6, con intención de no imponer sobrecarga modificando los datos con más bucles para mantener las mediciones que se tomaron correctamente, la interpolación se realiza para toda la fila sin importar qué columna es la fallida. De esta manera nos aseguramos que siempre y cuando se tengan los valores anterior y posterior tomados de manera correcta, cualquier columna podrá ser “corregida”. Es importante notar que se optó por una simple interpolación lineal debido a la naturaleza de las mediciones, al ser intervalos de tiempo tan pequeños, los valores creados se aproximan lo suficientemente bien a lo esperado.

Durante la lectura del archivo, todas estas filas se guardarán como excepciones en un nuevo archivo temporal que, al igual que los parámetros, se genera automáticamente. Como se puede apreciar en la Fig. 7, en este se guarda la línea donde se encontró la anomalía, los datos originales y un NaN (*not a number*) en la posición del fallo. Este archivo también se mostrará en pantalla en la ejecución de la interfaz gráfica.

```
1  Línea 19:    1.0,250253796577454.1,nan
2  Línea 35:    2.0,250253796577454.2,nan
3  Línea 51:    3.0,251649260520935.3,nan
```

**Fig. 7. Ejemplo de archivo con excepciones.**

## *2.5. Gráficos*

Existen distintas configuraciones que podríamos seleccionar para la creación de los gráficos. Para el formato "1" podríamos graficar valores de corriente o tensión en función del tiempo, para el resto, simplemente las alternativas son tensión en función de la corriente o viceversa. Debido a la naturaleza de las mediciones, el programa grafica valores de corriente en función del estímulo de voltaje aplicado.

Como se aclaró con anterioridad, utilizamos Matplotlib [2], una librería polivalente a la hora de realizar gráficos, que ofrece una gran cantidad de posibilidades, desde gráficos de barra, funciones, diagramas de caja y bigote, gráficos de pastel, curvas de nivel, etcétera. En nuestro caso, simplemente utilizamos curvas generadas mediante las propias mediciones como se muestra en las Fig. 8a y Fig. 8b.

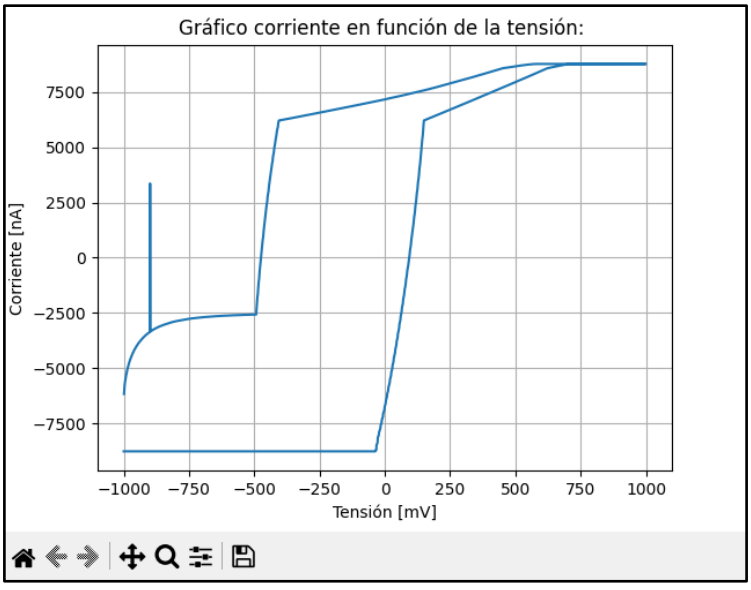


**Fig. 8a. Ejemplo "a" de gráfico de datos.**

* *quinonescristianj13@gmail.com*

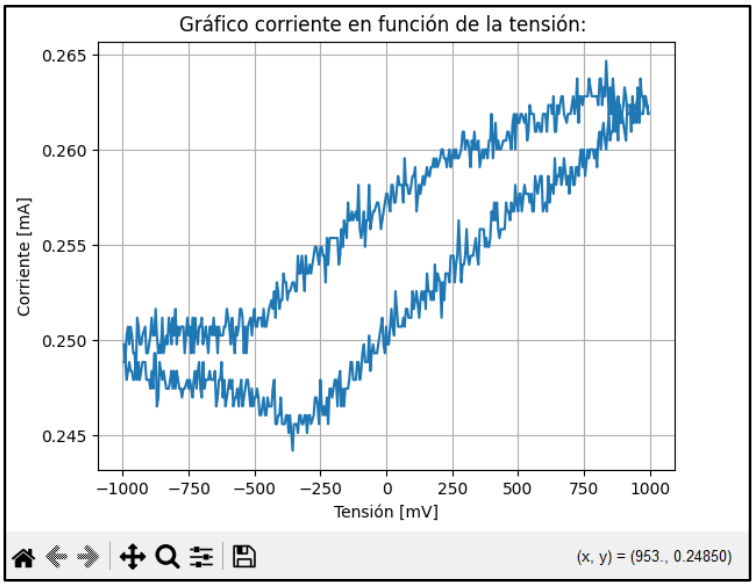


**Fig. 8b. Ejemplo "b" de gráfico de datos.**

A primera vista se puede observar en los ejemplos presentados que la propia ventana incorpora varias funcionalidades por defecto, como ser aumentar y reducir el zoom dentro del gráfico, cambiar las configuraciones de las escalas de los ejes y guardar el gráfico como una imagen en formato PNG. Estas funcionalidades pueden quitarse o implementarse dentro del programa mediante el uso de funciones, por ejemplo, mediante el uso de una botonera simple dentro de una interfaz gráfica, podríamos guardar en el sistema el gráfico en un formato distinto.

La implementación en cuanto a código se refiere, es sumamente sencilla. Matplot puede tomar un *data frame* creado en Pandas directamente para valerse de los datos, debido a que estos valores están divididos en columnas, simplemente se selecciona el encabezado de cada columna y se asigna a uno de los dos ejes (x, y). En casos más generales, también se pueden utilizar otros tipos de estructuras de datos como listas, tuplas, diccionarios o incluso arreglos de Numpy [3]. Esta última librería nombrada no fue muy utilizada en el desarrollo de este *software*. Su principal aplicación fue en la resolución de un problema encontrado al intentar compilar en sistemas operativos que utilizan el núcleo de Linux, donde los gráficos no se generaban correctamente al utilizar directamente el *data frame*. La solución fue simplemente modificar el *frame* mediante la función array() que genera un arreglo propio de Numpy con los valores que recibe como argumento. Aun así, es importante resaltar que esta librería tiene muchas más utilidades, por ejemplo, los valores de varianza, mediana y demás que se mostrarán luego en pantalla, también podrían calcularse con funciones propias de Numpy; en nuestro caso, optamos por trabajar directamente con las funciones de Pandas.

## *2.6. Historial*

El historial es una función simple que guardará en un archivo de texto permanente el nombre junto al día y hora específicos en los que se ejecutó el programa (en la Fig. 9 se muestra el formato

detallado). Más específicamente, a la hora de seleccionar un archivo CSV desde la interfaz gráfica, se escribe una cadena al final del texto. Esta cadena se crea de manera simple, se concatena el nombre del archivo con el día y la hora exactas del momento obtenidos mediante la librería Datetime [4].

Datetime es una librería muy útil para situaciones en las que se desea trabajar con horarios o fechas, los formatos se pueden modificar y generalizar a gusto, de esta manera resulta la herramienta perfecta para crear historiales o estampas de tiempo.

El uso de este historial se podría ampliar a la ejecución de archivos CSV no solo previamente ejecutados, sino que también modificados directamente desde la GUI.

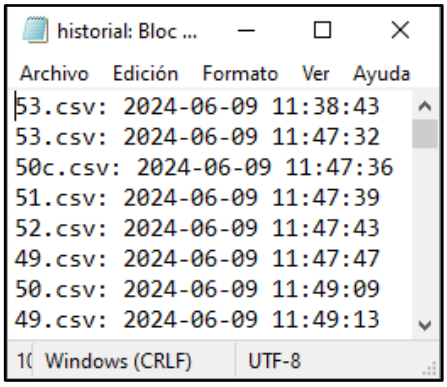


**Fig. 9. Historial.**

## *2.7. Desarrollo de la interfaz*

La interfaz fue desarrollada en base la librería la Tkinter, una librería integrada a Python. Para la codificación se partió del propósito fundamental del proyecto, que es analizar los datos extraídos del archivo CSV. En base a eso se definió el prototipo de la interfaz, como se puede observar en la Fig 10, consta de una ventana dividida en cuatro porciones, una de ellas (la primera a la izquierda del usuario) se puede visualizar los botones que permiten la interacción con el programa. Las tres restantes permiten representar los datos cargados, en las columnas: Características, Datos y Excepciones.


**Fig. 10. Ventana desplegada al abrir el programa.**

* *quinonescristianj13@gmail.com*

Todas las características se tomaron siguiendo la documentación de la librería de Tkinter [6]. En primer lugar, se definió la ventana principal de la interfaz del programa en la cual se implementan los respectivos *widgets* o componentes gráficos (Fig. 11).

```
# Crear la ventana principal
root = tk.Tk()
root.title("PGD Proyecto en desarrollo")
```

**Fig. 11. Creación de la ventana principal.**

En la interfaz se utilizó los *widgets*: **tk.frame** que permite crear el contenedor donde se puede agrupar botones, texto, imágenes, añadir *widgets*, etc. permitiendo al usuario interactuar con la interfaz. Por ejemplo, para el *frame* características se codificó como se observa en la Fig. 12.

```
# Crear el frame1 para mostrar datos
frame1 = tk.Frame(root, bg="lightskyblue", bd=2, relief="solid")
```

**Fig. 12. Creación del frame para las Características.**

El código que aparece dentro del paréntesis, excepto root, son parámetros, **bg** permite elegir el color del *frame*, **bd** y **relief** definen el ancho en pixeles y estilo de relieve, respectivamente, del borde del *frame*. El primer argumento **root** se refiere al *widget* donde se colocará el *frame*.

Además, se utilizó el *widget* **tk.Label** que nos permite mostrar texto o imágenes en una ventana (Fig. 13).

```
#Agregar un título al frame1
titulo_frame1 = tk.Label(frame1, text="Características\n\n\n\n", bg="lightskyblue", font=("Arial", 14, "bold "))
```

**Fig. 13. Añadir texto al frame Características.**

Los parámetros: **text** posibilita la escritura del texto, **font** define el formato y tamaño del texto. El argumento **frame1** referencia al frame donde se colocará el texto.

Los botones interactivos de la interfaz se implementan con **tk.Button**, como se observa en la Fig. 14.

```
# Añadir botones al menú
btn_nuevo = tk.Button(menu_frame, text="Nuevo Archivo", command=nuevo_archivo)
```

**Fig. 14. Creación del botón Nuevo Archivo.**

Para este *widget* el parámetro **text** define el nombre del botón, **command** se utiliza para asignar una función o método que se ejecutará cuando el botón sea presionado. Al igual que los *widgets* anteriores, el argumento **menu_frame** referencia el *frame* donde colocará el botón.

Los códigos presentados anteriormente representan los desarrollos de *widgets* más importantes de la interfaz. Cualquier otra parte del código se puede extrapolar a partir de ellos, siguiendo la

documentación de la biblioteca, la cual proporciona toda la información necesaria para su correcta implementación.

## *2.8. Guía para el uso de la interfaz*

Cuando se ingresa al programa, se abrirá una ventana exactamente igual a la de la Fig. 10, en la que la columna de la izquierda muestra los botones disponibles para el usuario. El botón **Abrir Archivo** abrirá una ventana que permitirá seleccionar un archivo CSV para analizarlo (Fig. 15).

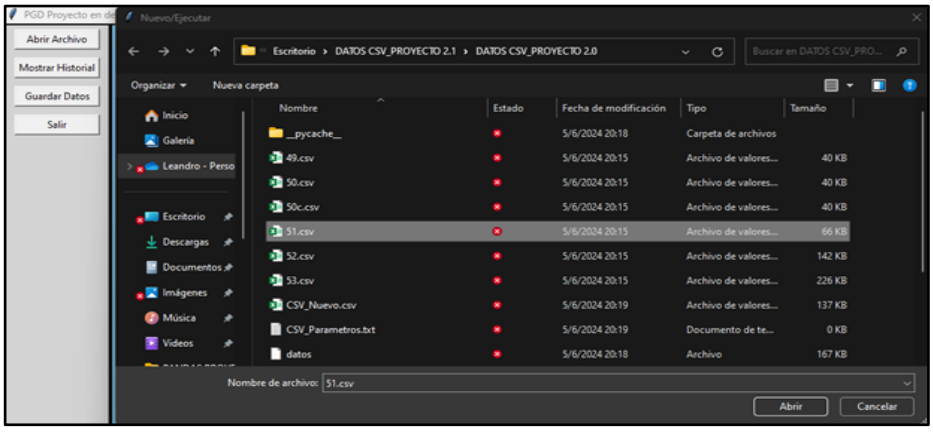


**Fig. 15. Venta desplegada al presionar el botón Abrir Archivo.**

Una vez seleccionado, el programa ejecutará el análisis del CSV, donde se podrán observar y analizar sus características (Fig. 16).

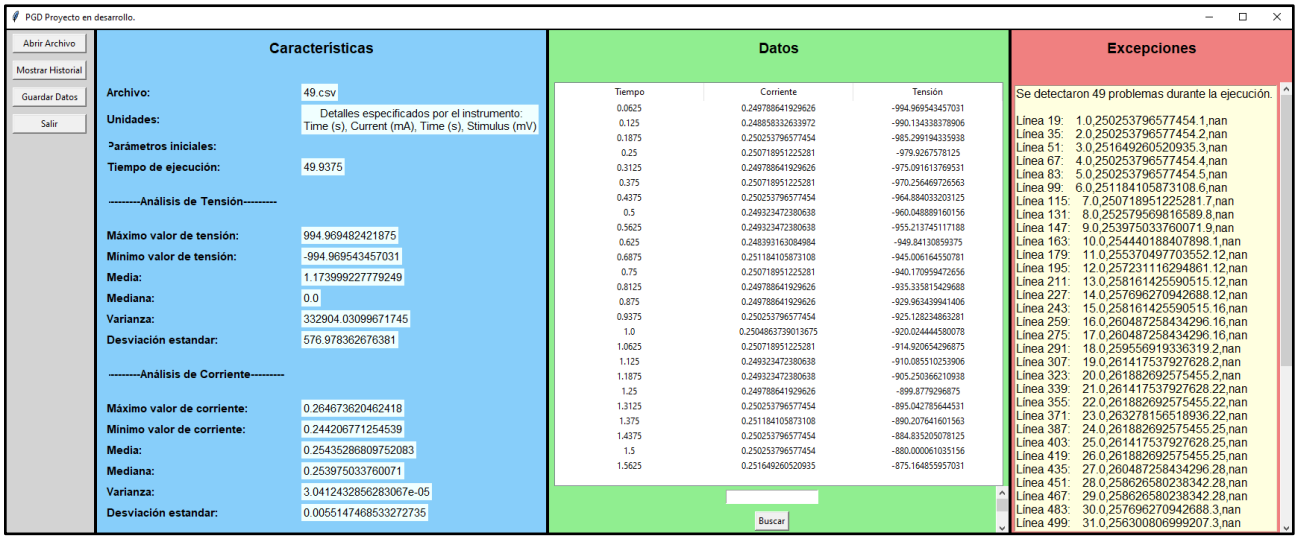


**Fig. 16. Interfaz luego de ejecutar un archivo.**

En el apartado "Características" aparecen medidas importantes que describen el CSV. Estas medidas son: máximos, mínimos, mediana, media, varianza, desviación estándar, tiempo de

* *quinonescristianj13@gmail.com*

ejecución, parámetros iniciales medidos por el instrumento que relevó los datos y unidades. En la columna "Datos" aparecen los datos procesados, además de un buscador en caso de querer buscar un valor específico. En "Excepciones" se encuentran las líneas del CSV que poseen datos que no se procesaron, esto se debe principalmente a errores de formato en el momento en que se crearon los CSV. Además, se abrirá una ventana que permite interactuar con el gráfico en función de los datos procesados, como se mostró en la Fig. 8**.**

El botón **Mostrar Historial** que permite al usuario tener un seguimiento de los archivos ejecutados en caso de que lo necesite. Al presionarlo, aparecerá una ventana (Fig. 17) con los nombres, fechas y horas de los archivos analizados.

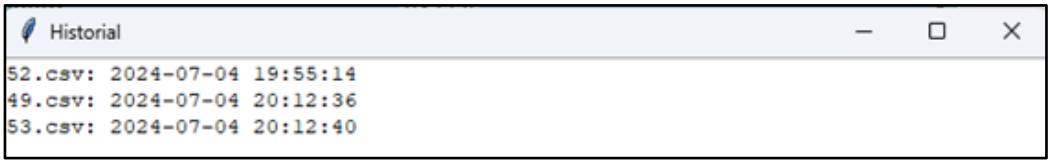


**Figura 17. Ventana abierta al presionar Mostrar Historial.**

Por medio del botón **Guardar Datos**, el programa posibilita guardar los datos procesados en caso de que el usuario los necesite, permitiendo al usuario elegir el directorio y nombre (Fig. 18).

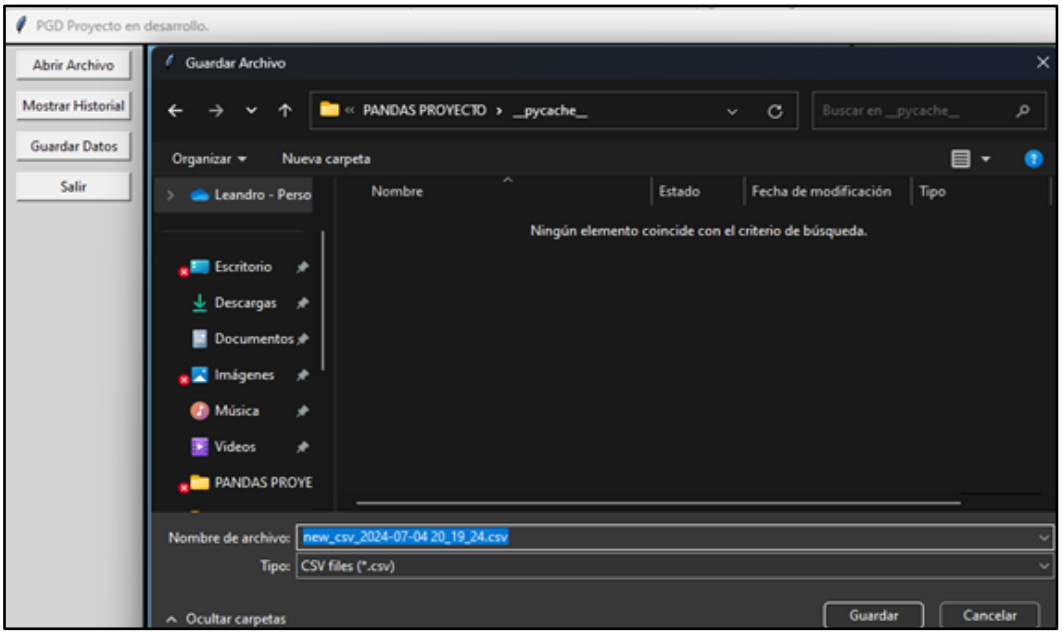


**Fig. 18. Ventana emergente al presionar Guardar Datos.**

Por último, el botón "Salir" abre una ventana de advertencia que interactúa con el usuario, para asegurarse de que no se cierre el programa accidentalmente como se muestra en la Fig. 19.

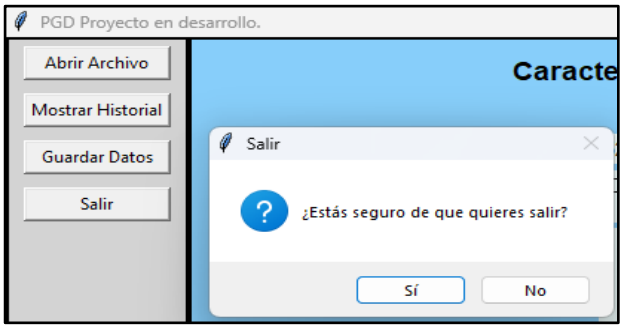


**Fig. 19. Ventana de advertencia, al presionar Salir.**

### *2.9. Compilado y archivo ejecutable*

Una vez el código estaba completo y no se encontraron más errores durante las pruebas realizadas al *software*, el último paso consistió en la creación del archivo ejecutable. Para ello, usamos la utilidad PyInstaller [5], que nos permite empaquetar *scripts* de manera simple.

PyInstaller nos permite crear un archivo ejecutable con o sin las dependencias incluidas. En caso de que se elija la primera opción, el tamaño del archivo será superior ya que se incluirán todas las librerías y demás agregados que necesita el *software* para su correcta ejecución. Esto nos facilita la distribución del programa, ya que, siempre y cuando el sistema operativo sea compatible con la versión que se tiene, no necesitaremos instalar librerías desde la consola de comandos y entre otros. Por otro lado, la segunda opción nos ofrece la posibilidad de que el archivo portable sea mucho más ligero, pero para su funcionamiento se necesitará primero instalar todas las dependencias necesarias.

Es importante resaltar que PyInstaller detecta automáticamente el sistema operativo desde el cuál se está ejecutando y empaquetando, por lo tanto, en caso de que se realice desde Windows, el .exe será funcional en dicho sistema operativo, y en caso de que sea en alguna distribución de GNU/ Linux como Ubuntu, lo será solo para ese sistema.

El proceso es simple, se abre una terminal en la carpeta donde se encuentran los archivos .py del programa (en caso de que este sea modular, PyInstaller detectará todos los archivos que comparten funciones y las incluirá) y se ejecuta la línea de comandos:

***pyinstaller --onefile --icon=icono.ico script.py***

Esta línea posee los argumentos principales, **pyinstaller** es la llamada a la librería, **--onefile** nos empaquetará todas las utilidades en el mismo archivo, **--icon=icono.ico** nos permite elegir un ícono personalizado para el programa y, finalmente, **script.py** es el archivo principal del programa (en nuestro caso, será aquel que contiene el código de la interfaz gráfica).

## 3. Conclusiones

Durante el desarrollo del software en Python, se tuvo que considerar la necesidad del usuario final del programa, en este caso, un ingeniero que tiene conocimiento de la forma de trabajar y el uso que le dará. Por este motivo, nos centramos desde un principio en crear un programa con una interfaz

* *quinonescristianj13@gmail.com*

sencilla que tenga la finalidad de mostrar de forma rápida en pantalla los datos, gráficos y análisis correspondientes a los archivos CSV.

Para lograr el resultado obtenido, fue crucial el uso de la documentación de las librerías de Python disponibles en línea, ya que nos permitió encontrar los métodos que más se adaptan al propósito del programa. La programación nos permitió entender la importancia de tener desde el principio los parámetros a considerar y utilizar. Por ejemplo, en la interfaz gráfica, las dimensiones absolutas y relativas de los *frames* generaron conflictos con las diferentes resoluciones de los monitores donde se probó el programa. Cabe aclarar que el programa funciona y se puede utilizar, si se readaptan manualmente las dimensiones del monitor. Estos son los aspectos que, de alguna manera, forman parte del proceso de programación y fortalecen nuestra curva de aprendizaje, teniendo en cuenta que todo lo relacionado con la interfaz se realizó con la documentación en línea y sin experiencia previa.

En cuanto al manejo de datos, se emplearon librerías de Python como NumPy, Matplotlib y Pandas. Estas herramientas fueron fundamentales para la manipulación y visualización de datos. NumPy permitió realizar operaciones matemáticas y trabajar con arreglos de forma eficiente. Pandas facilitó la lectura, manipulación y análisis de los datos almacenados en archivos CSV. Matplotlib se utilizó para crear gráficos que representan visualmente la información, lo cual es esencial para el análisis y la toma de decisiones.

En el futuro, se pretende seguir trabajando en el programa para mejorar algunos aspectos, como las dimensiones de la interfaz, y si es posible, adaptarlo para virtualizarlo y eliminar la necesidad de tenerlo instalado para ejecutarlo. Con estas mejoras, esperamos ofrecer una herramienta aún más útil y accesible para los potenciales usuarios.

**Agradecimientos**

Se agradece al ingeniero electrónico Skrauba, Axel Alfredo por brindarnos ayuda relacionada al desarrollo eficiente para el código del manejo de archivos.

**Referencias**

[1] Documentación de Pandas - 2.2.2 [En linea]. Available: https://pandas.pydata.org/docs/ [Último acceso: 5 julio 2024].

[2] Documentación de Matplotlib- 3.9.1 [En línea]. Available: https://matplotlib.org/stable/index.html [Último acceso: 5 julio 2024].

[3] Documentación de Numpy- 2.0 [En línea]. Available: https://numpy.org/doc/ [Último acceso: 5 julio 2024].

[4] Documentación de Python - 3.12.4, Datetime [En línea]. Available: https://docs.python.org/3/library/datetime.html [Último acceso: 5 julio 2024].

[5] Documentación de PyInstaller- 6.8.0 [En línea]. Available: https://pyinstaller.org/en/stable/index.html [Último acceso: 5 julio 2024].

[6] Documentación de Python - 3.12.4, Interfaces gráficas de usuario con Tk [En línea]. Available: https://docs.python.org/es/3/index.html [Último acceso: 5 julio 2024].

* *quinonescristianj13@gmail.com*