11/08/25 Primer material para estudiar

Measurement 171 (2021) 108829

Contents lists available at ScienceDirect

Measurement

journal homepage: www.elsevier.com/locate/measurement

Digital image colorimetry on smartphone for chemical analysis: A review

Yajun Fan [a], Jiawei Li [a], Yaping Guo [a], Lianwu Xie [a,*], Gui Zhang [a,b,*]

[a] College of Sciences, Central South University of Forestry and Technology, Changsha 410004, PR China
[b] Key Laboratory for Digital Dongting Lake Basin of Human Province, Central South University of Forestry and Technology, Changsha 410004, PR China

ARTICLE INFO

Keywords:
Digital image colorimetry
Chemical analysis
Color space
Smartphone
Digital application

ABSTRACT

Digital image colorimetry (DIC) on smartphone is regarded as a powerful, fast and low-cost analysis method to measure target analyte with color changes of digital image obtained by the built-in camera. We summarized the basic procedure of DIC, the color spaces (RGB, CMYK, HSB/HSL, CIE XYZ, L*a*b*, and YUV), the principal architectures (tools for capturing image, lighting conditions, and color quantification APPs and DIC APPs), and current status of DIC on smartphone in analysis of metals/heavy metals, herbicides, pesticides, antibiotics, biological and medical indicators, natural compounds, and bacteria/viruses. The advantages and disadvantages of DIC are also revealed. Nowadays, DIC on smartphone must be further refined with controlled geometry and standard lighting sources to become robust and reliable analytical procedures. And it will be improved in the near future with the continuous development of smartphones owing to the rapid development of smartphone's camera technology and the continuous optimization of related software.

Paper 10 paginas
leído completo

Lab on a Chip

Devices and applications at the micro- and nanoscale

rsc.li/loc

Themed issue: Lab on a Chip Reviews Issue 2025

ISSN 1473-0197

ROYAL SOCIETY OF CHEMISTRY

CRITICAL REVIEW
W. Russ Algar et al.
Smartphones as a platform for molecular analysis:

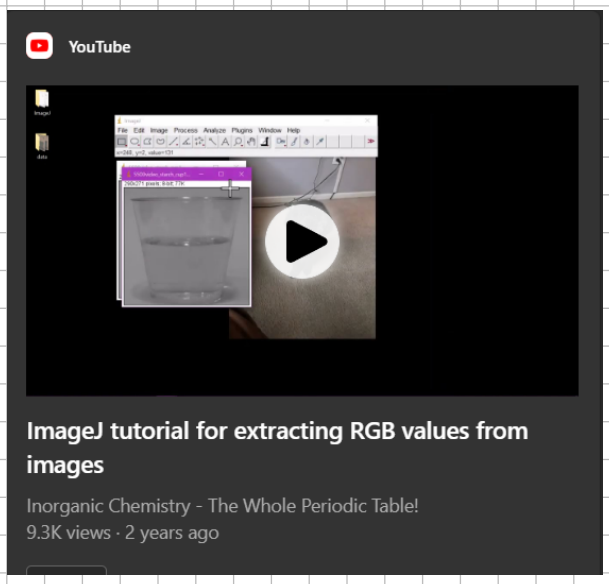
1 / 73

Solo leí introducción y p. 23
capítulo sobre

cial 3.0 Unported

7. Analytes and assays

7.1 Metanalysis



También reviso este vídeo, con el cual aprendo a usar ImageJ para entender el problema, además de usar los valores R, G, B para el primer gráfico

26/08/2025

Usando imageJ se obtienen los RGB y se obtienen valores para cada color que despues serán transformados en rectas

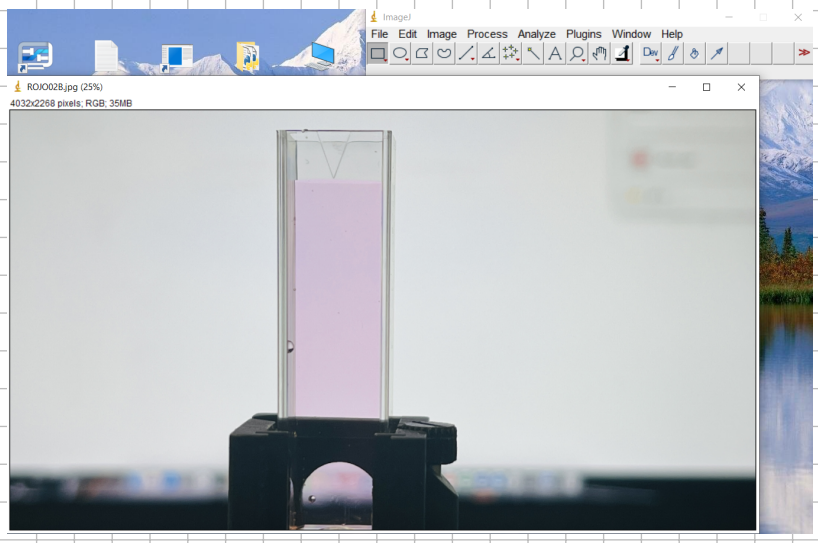


Imagen de referencia

,Area,Mean,Min,Max
1,326144,209.750,171,218
2,326144,213.566,189,224
3,326144,214.694,177,223
4,326144,215.660,169,224
5,326144,214.812,193,226

ResultadosRectangulo (Red)....

ResultadosRectangulo (Red).csv

Primeras pruebas para hacer el macro y poder graficar

,Area,Mean,Min,Max
1,75952,208.362,196,215
2,75952,212.491,198,221
3,75952,213.602,204,219
4,75952,214.580,207,220
5,75952,212.891,203,221

ResultadosRectPequeñoHori...

ResultadosRectPequeñoHorizontal (Red).csv

Esto se debe hacer

Usaste las fotos de SNC (Tiocianato)

FeCN05E: 2E-5 mol/L
FeCN05D: 4E-5 mol/L
FeCN05C: 6E-5 mol/L
FeCN05B: 8E-5 mol/L
FeCN05A: 10E-5 mol/L

Debes graficar x versus y: La concentración (x) vesrsus El mean (y) de tu tabla. Aprovecha y hace uno grafico para cada R, G y B

27/08/25

Rojo $y = 61090\,x + 210{,}03$

Concentraciones deberían ser $(2, 4, 6, 8, 10)\cdot 10^{-5}$

Valores de y

```
Mean Min Ma
4 209.750 1
4 213.566 1
4 214.694 1
4 215.660 1
4 214.812 1
```

El peor

$y = 209{,}750 \Rightarrow x_0 = -4{,}5\cdot 10^{-6}$

$y = 213{,}566 \Rightarrow x_1 = 5{,}78\cdot 10^{-5}$

$y = 214{,}694 \Rightarrow x_2 = 7{,}63\cdot 10^{-5}$

$y = 215{,}660 \Rightarrow x_3 = 9{,}21\cdot 10^{-5}$

$y = 214{,}812 \Rightarrow x_4 = 7{,}82\cdot 10^{-5}$

x esperados

Datos demasiado dispersos

02/09/25

Verde $y = -722200\,x + 219{,}3$

```
Valores de x calculados para cada dato verde:
Para y = 199.248, x = 0.00002777
Para y = 195.477, x = 0.00003299
Para y = 179.013, x = 0.00005578
Para y = 162.671, x = 0.00007841
Para y = 143.431, x = 0.00010505
```

Azul $y = -1217635\,x + 225{,}724$

```
Valores de x calculados para cada dato azul:
Para y = 204.119, x = 0.00001774
Para y = 176.107, x = 0.00004075
Para y = 149.918, x = 0.00006226
Para y = 125.562, x = 0.00008226
Para y = 107.628, x = 0.00009699
```

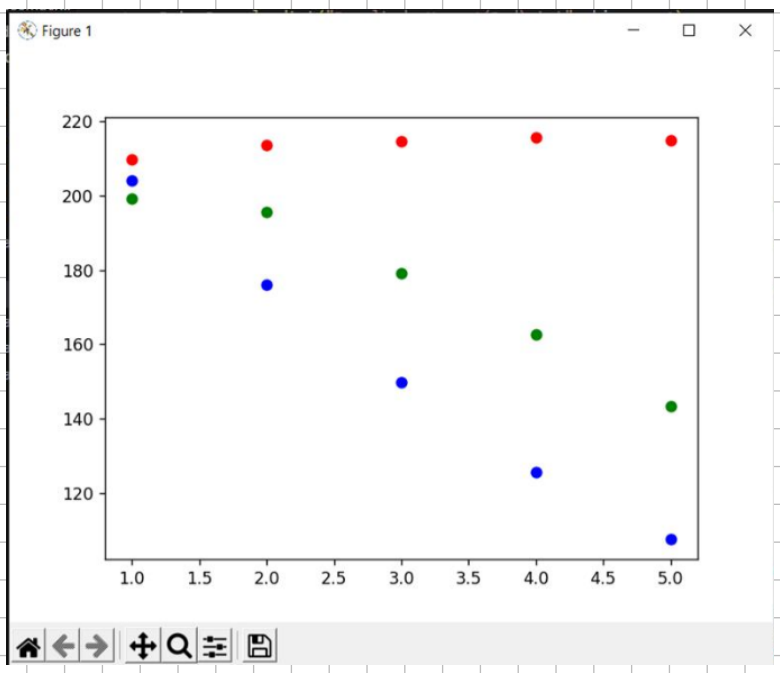
```
 El coeficiente de correlacion para rojo es R^2 =  0.68835005139033
La recta para rojo es y = 61090.000000000095x + 210.03099999999998
 El coeficiente de correlacion para verde es R^2 =  0.962623173711992
La recta para verde es y = -722199.9999999998x + 219.3
 El coeficiente de correlacion para azul es R^2 =  0.9938078708273564
La recta para azul es y = -1217634.9999999998x + 225.72490000000002
PS C:\Users\Diego\OneDrive - Universidad de Concepción\Documentos\Unive
```

28/08/25

Primer codigo hecho con valores de imageJ, antes el profesor me habia enviado una rutina para hacer lo mismo que hice con ImageJ pero usando python, luego de este codigo, implemento eso

Hola profe, este código es el que hice de primeras solo para ver los valores de r,g,b que obtuve con el ImageJ, tengo algunas dudas con las mediciones todavía, igual comprobé el "macro" varias veces pero los resultados usando otros rectángulos para medir eran parecidos. los valores dados los tengo en csv y txt. al principio grafiqué r,y pero después me di cuenta que se visualizaba mejor de la forma y,r (se parecía mas al grafico del paper y creo que el profe cesar dijo que era así). Le envió esto para saber si voy por buen camino y así hacer las otras imágenes.

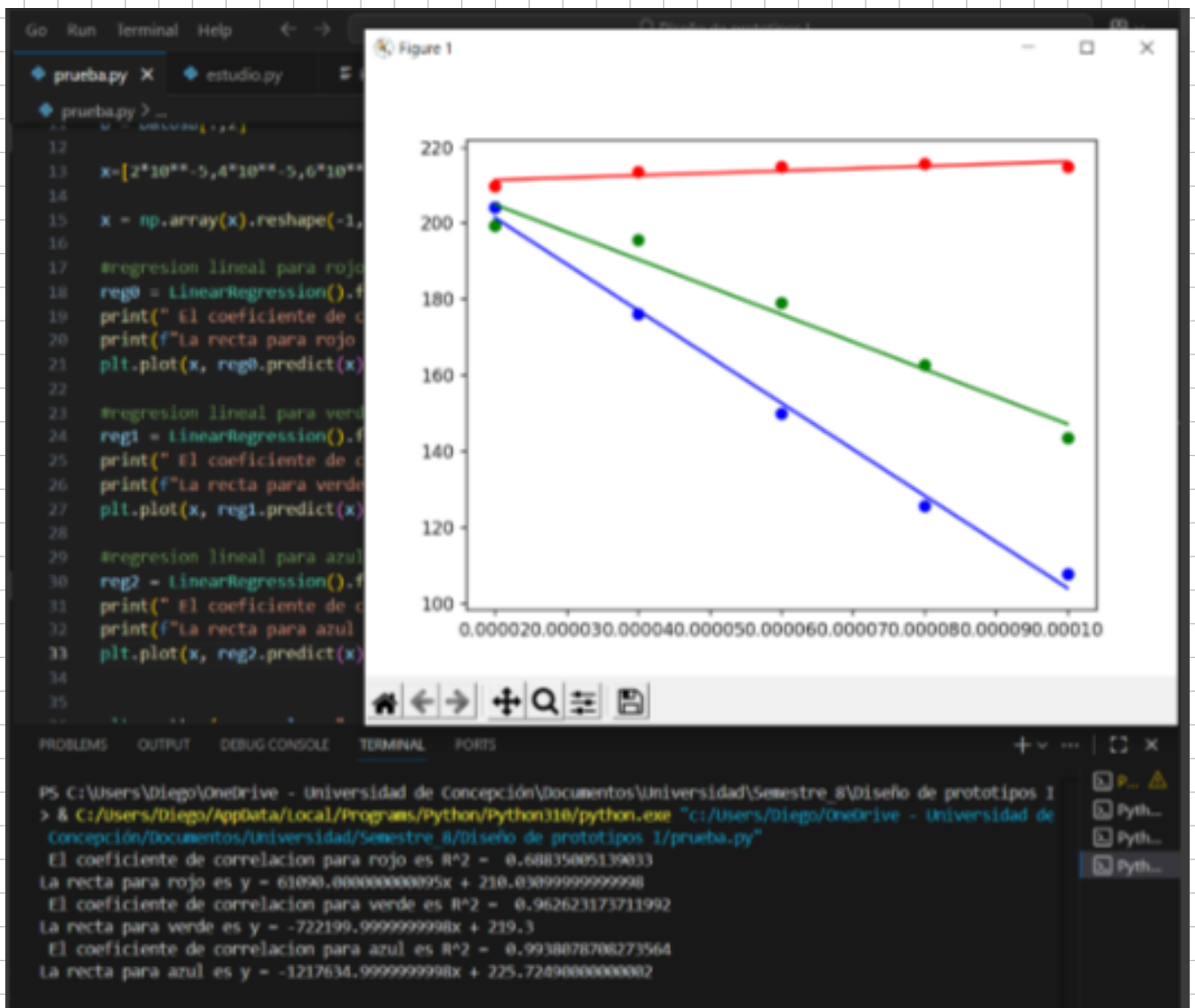
```
import numpy as np
import matplotlib.pyplot as plt

DatosR=np.loadtxt("ResultadosMacro1(Red).txt",skiprows=1)
DatosG=np.loadtxt("ResultadosMacro1(Green).txt",skiprows=1)
DatosB=np.loadtxt("ResultadosMacro1(Blue).txt",skiprows=1)

r = DatosR[:,2]
g = DatosG[:,2]
b = DatosB[:,2]

y=[1,2,3,4,5] #arreglo cualquiera solo para revisar valores de r, g y b

plt.scatter(y,r, color= "red") #plot es la idea cuando se use la absorbancia (?)
plt.scatter(y,g, color="green")
plt.scatter(y,b,color="blue") #recuerda hacer esto con las otras fotos
plt.show()

print("hola", r, g ,b )
```



03/09/25

Descubrí el ANOVA, que es importante despues

Primer implemento de regresion lineal y rutina para calcular valores promedios de RGB



Anotar el error a cada dato asociado

Analisis de varianza ANOVA

EL calculo estadístico del ajuste te debe dar el error de cada dato.

falta info ahí . Eso se llama ANALISIDS DE VARIANZA (ANOVA)

04/09/25

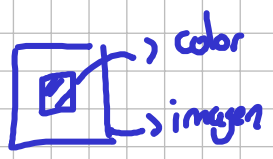
Que el canal rojo en el RGB no sea una buena curva puede ser literalmente debido al color de la muestra

-Hacer tarea usando curvas azul y verde ✓

-Revisar como hacer anova en excel

-Aprender libreria en python para analisis de color

* Recuerda que para interfaz de usuario siempre se puede



No desesperar

15/09/25 No trabajo hace harto... hay razón.

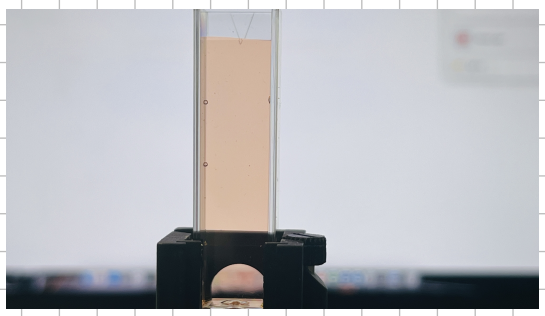
*Haciendo mismo procedimiento para rojo metilo ✓

Image J => código python ✓

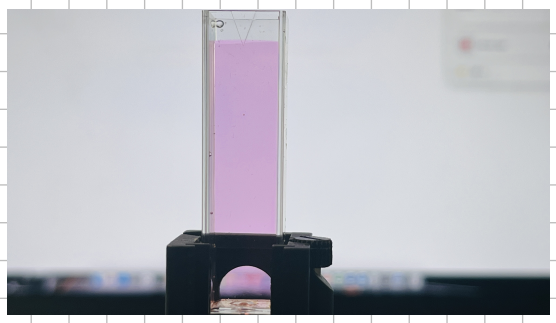
* Se usará el mismo macro ✓

	A	B	C	D	E	F	G	H
3	Water01	Water01	0					
4		Rojo metilo			SNC			
5		ROJO05B	4.0	ppm	FeCN01A:	2,00E-05	mol/L	
6		ROJO04B	2.0	ppm	FeCN02A:	4,00E-05	mol/L	
7		ROJO03B	1.0	ppm	FeCN03A:	6,00E-05	mol/L	
8		ROJO02B	0.5	ppm	FeCN04A:	8,00E-05	mol/L	
9		ROJO01B	0.25	ppm	FeCN05A:	1,00E-04	mol/L	
10								
11								
12								

Tarea de calcular concentraciones enviadas por el profesor.



FeCN ¿Cual es la concetración?



Rojo metilo ¿Cual es la concentración?

```
 El coeficiente de correlacion para rojo es R^2 =  0.68835005139033
La recta para rojo es y = 61090.000000000095x + 210.03099999999998
 El coeficiente de correlacion para verde es R^2 =  0.962623173711992
La recta para verde es y = -722199.9999999998x + 219.3
 El coeficiente de correlacion para azul es R^2 =  0.9938078708273564
La recta para azul es y = -1217634.9999999998x + 225.72490000000002
```

```
 El coeficiente de correlacion para rojo es R^2 =  0.8360238653240935
La recta para rojo es y = 2.343661290322579x + 208.57512499999999
 El coeficiente de correlacion para verde es R^2 =  0.9929728163860311
La recta para verde es y = -43.4001129032258x + 210.761375
 El coeficiente de correlacion para azul es R^2 =  0.9945054438979913
La recta para azul es y = -12.390919354838708x + 219.370125
```

Se debe usar ImageJ con Macro 1 para obtener los mean de R,G,B y luego reemplazar esos valores en la recta obtenida por la regresión lineal, esto nos dará valores aproximados de la concetración para esa imagen.

Rojo metilo

Se comprueba usando los mismos valores mean para obtener la concentración de 0.25 ppm, se obtienen tres valores para r,g,b y luego se calcula el promedio obteniendo el siguiente resultado:

```
Ingresa un valor de y (rojo): 209.324
Para y = 209.324, x = 0.3195320941179825
Ingresa un valor de y (verde): 200.349
Para y = 200.349, x = 0.23991585052365327
Ingresa un valor de y (azul): 216.664
Para y = 216.664, x = 0.2183958205605834
```

$$\frac{0.3195 + 0.2399 + 0.2184}{3} = 0.2592 \text{ ppm}$$

Habría que calcular el error

Ahora calculamos la concentración para la imagen incógnita dada por el profesor

Results

File Edit Font Results

	Area	Mean	Min	Max	
1	326144	207.635	126	221	Blue
2	326144	169.057	91	182	green
3	326144	201.008	131	217	red

```
Ingresa un valor de y (rojo): 201.008
Para y = 201.008, x = -3.2288
Ingresa un valor de y (verde): 169.057
Para y = 169.057, x = 0.9609
Ingresa un valor de y (azul): 207.635
Para y = 207.635, x = 0.9471
```

Desconfiando del valor negativo se hará el promedio de b,g

$$\frac{0.9609 + 0.9471}{2} = 0.954 \text{ ppm}$$

0.954 ppm sería la concentración para Rojo Metilo aproximadamente.

Fe CN

Mismo procedimiento que para Rojo metilo, primero comprobar para un valor dado

```
Ingresa un valor de y (rojo): 209.750
Para y = 209.75, x = -4.5998e-06
Ingresa un valor de y (verde): 199.248
Para y = 199.248, x = 2.7765e-05
Ingresa un valor de y (azul): 204.119
Para y = 204.119, x = 1.7744e-05
```

Valor negativo no tiene sentido →

$$\frac{(2.7765 + 1.7744) \cdot 10^{-5}}{2} = 2.2754 \cdot 10^{-5} \frac{mol}{L}$$

Ahora calculamos la concentración para la imagen incógnita dada por el profesor

Results

File Edit Font Results

	Area	Mean	Min	Max	
1	326144	164.078	113	177	Blue
2	326144	184.714	131	198	Green
3	326144	212.551	167	228	Red

```
Ingresa un valor de y (rojo): 212.551
Para y = 212.551, x = 4.1251e-05
Ingresa un valor de y (verde): 184.714
Para y = 184.714, x = 4.7890e-05
Ingresa un valor de y (azul): 164.078
Para y = 164.078, x = 5.0628e-05
```

Como el valor de rojo tiene un R^2 lejos de 1, se tomará solo g,b

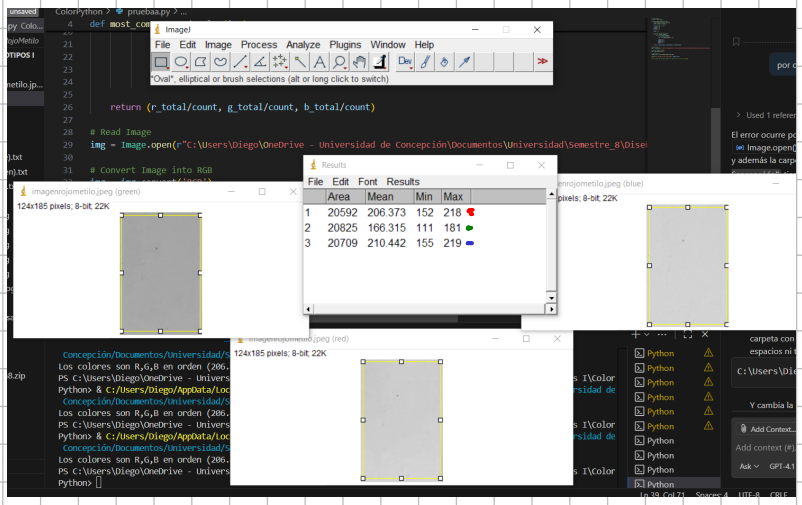
$$\frac{(4.7890 + 5.0628) \cdot 10^{-5}}{2} = 4.9259 \cdot 10^{-5} \frac{mol}{L}$$

4.9259e-5 mol/L sería la concentración para FeCN aproximadamente

* Graficar el análisis espectral de las sustancias !!

25/09/25

luego la idea es añadir libreria de python q tu le des una imagen, que extraiga los valores mean de r, g, b y con esos valores de mean calcule la concentracion usando las funciones de arriba



El códgio funciona bastante bien para obtener los mean de los valores r, g, b para una sola imagen.-
Ahora debo buscar como hacerlo para muchas imagenes a la vez en un mismo código, para asi poder tener muchos datos y hacer lo ya realizado para el ajuste de curvas.

```
# Import Module
from PIL import Image

def most_common_used_color(img):
    # Get width and height of Image
    width, height = img.size

    # Initialize Variable
    r_total = 0
    g_total = 0
    b_total = 0

    count = 0

    # Iterate through each pixel
    for x in range(0, width):
        for y in range(0, height):
            # r,g,b value of pixel
            r, g, b = img.getpixel((x, y))

            r_total += r
            g_total += g
            b_total += b
            count += 1

    return (r_total/count, g_total/count, b_total/count)

# Read Image
img = Image.open(r"C:\Users\Diego\OneDrive - Universidad de Concepción\Documentos\Universidad\Semestre_8\Diseño de prototipos I\ColorPython\imagenrojometilo.jpeg")

# Convert Image into RGB
img = img.convert('RGB')

# call function
common_color = most_common_used_color(img)

print("Los colores son R,G,B en orden", common_color)
```

27/09/25

Agregué el ciclo for al codigo para obtener los promedios de r,g,b

```
#Código que me mandó el profe para obtener el mean de r,g,b de una imagen
from PIL import Image
import os

def most_common_used_color(img):
    # Get width and height of Image
    width, height = img.size

    # Initialize Variable
    r_total = 0
    g_total = 0
    b_total = 0

    count = 0

    # Iterate through each pixel
    for x in range(0, width):
        for y in range(0, height):
            # r,g,b value of pixel
            r, g, b = img.getpixel((x, y))

            r_total += r
            g_total += g
            b_total += b
            count += 1

    return (r_total/count, g_total/count, b_total/count)

# Read Image
#img = Image.open(r"C:\Users\Diego\OneDrive - Universidad de Concepción\Documentos\Universidad\Semestre_8\Diseño de prototipos I\ColorPython\imagenrojometilo.jpeg")

# Convert Image into RGB
#img = img.convert('RGB') #RUTINA ORIGINAL DEL PROFESOR PARA UNA SOLA IMAGEN

# call function
#common_color = most_common_used_color(img)

#print("Los colores son R,G,B en orden", common_color)

#voy a escribir yo el codigo que me sugeriste para entenderlo mejor

imagenes = [r"C:\Users\Diego\OneDrive - Universidad de Concepción\Documentos\Universidad\Semestre_8\Diseño de prototipos I\ColorPython\imagenrojometilo.jpeg",
            r"C:\Users\Diego\OneDrive - Universidad de Concepción\Documentos\Universidad\Semestre_8\Diseño de prototipos I\ColorPython\rojo4.jpeg"]

for i in imagenes:
    img = Image.open(i).convert('RGB') #abrir y convertir la imagen a RGB
    common_color = most_common_used_color(img) #le aplicas la funcion a la imagen
    nombre = os.path.basename(i) #para mostrar solo el nombre de la imagen y no toda la ruta
    print(f"Los colores de la imagen {nombre} son R,G,B en orden {common_color}")

#Un f-string te permite incrustar variables o expresiones directamente dentro de la cadena usando llaves { }.

#Este codigo funciona para obtener el mean de los r,g,b de una imagen
```

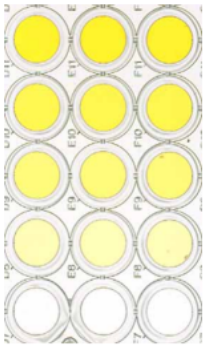
El siguiente problema es revisar si python tiene algo para cortar las imagenes o en realidad mandarle directamente cortadas las imagenes para que haga la regresión.

-Queremos tomar una imagen como la del paper

Cost-Effective Teacher
edited by
Harold H. Harris
University of Missouri—St. Louis
St. Louis, MO 63121

Demonstration of Absorbance Using Digital Color Image Analysis and Colored Solutions

Shane K. Kohl, James D. Landmark, and Douglas F. Stickle*
Department of Pathology & Microbiology, University of Nebraska Medical Center, Omaha, NE 68198-3135;
*dstickle@unmc.edu



-Falta avanzar en la tabla (Conc predicha, nominal, error, sigma)
-Anova

- Me parece complicado el ver como poner las imagenes para que sea correcto...

Probar usando la misma imagen siempre de los mismos pixel x pixel

Otro mini problema es el como de los valores que calcula python r,g,b poner, por ejemplo, todos los r como una sola lista...

01/09/2025

Consulto con ChatGPT para poder decidir si es mejor colocar imagen manualmente o si es mejor hacer un codigo que las recorte

🔹 **Opción 1: Cortar/ajustar las imágenes manualmente antes de meterlas al código**

- Tú te aseguras que todas las imágenes que vas a usar ya estén en 50×50.
- Ventaja: tu código queda más limpio porque no necesita redimensionar nada.
- Desventaja: si te llega una imagen nueva más grande o más chica, **tienes que volver a editarla manualmente**.

🔹 **Opción 2: Hacer que el código recorte/redimensione automáticamente cada imagen**

- Usas **PIL** para que todas las imágenes se adapten a 50×50 antes de calcular RGB.
- Ventaja: nunca tienes que preocuparte del tamaño original, siempre se ajustan automáticamente.
- Desventaja: si las imágenes tienen regiones importantes distintas, el reescalado puede distorsionar un poco los colores (aunque en 50×50 eso no es tan grave).

Con tu caso, yo recomiendo **Opción 2** porque es más flexible: así no dependes de editar manualmente cada imagen.

```python
    return (r_total/count, g_total/count, b_total/count)


imagenes = [r"C:\Users\Diego\OneDrive - Universidad de Concepción\Documentos\Universidad\Semestre_8\Diseño de
            r"C:\Users\Diego\OneDrive - Universidad de Concepción\Documentos\Universidad\Semestre_8\Diseño de

for i in imagenes:
    img = Image.open(i).convert('RGB') #abrir y convertir la imagen a RGB
    common_color = most_common_used_color(img) #le aplicas la funcion a la imagen
    nombre = os.path.basename(i) #para mostrar solo el nombre de la imagen y no toda la ruta
    print(f"Los colores de la imagen {nombre} son R,G,B en orden {common_color}")

#Un f-string te permite incrustar variables o expresiones directamente dentro de la cadena usando llaves { }.

#Este codigo funciona para obtener el mean de los r,g,b de una imagen
```

Se puede modificar el ciclo for para poder forzar la imagen a 50x50 usando la misma libreria

Duda con append y listas

3. Las listas vacías `r_vals, g_vals, b_vals`

python — Copiar código

```python
r_vals, g_vals, b_vals = [], [], []
```

Aquí estás creando **tres listas vacías** que servirán para **guardar los resultados** de varias imágenes.

- `r_vals` → lista donde irán todos los promedios rojos de las imágenes.
- `g_vals` → lista de promedios verdes.
- `b_vals` → lista de promedios azules.

4. Uso de `.append()`

Dentro del loop:

python — Copiar código

```python
r_vals.append(common_color[0])
g_vals.append(common_color[1])
b_vals.append(common_color[2])
```

- `.append(valor)` **agrega un valor nuevo al final de una lista.**
  Ejemplo:

python — Copiar código

```python
nums = []
nums.append(10)   # nums = [10]
nums.append(20)   # nums = [10, 20]
```

En tu código, cada vez que analizas una imagen:

- Guardas el valor promedio de rojo en `r_vals`.
- Guardas el promedio de verde en `g_vals`.
- Guardas el promedio de azul en `b_vals`.

De esta forma, cuando proceses varias imágenes, terminas con algo como:

duda con convert("RGB")

en esta perte "for i in imagenes: # Abrir imagen img = Image.open(i).convert('RGB')" el convert to RGB es una funcion que separa los 3 canales de una sola imagen rgb cierto ?

¡Muy buena observación! Te lo explico con precisión 👇

¿Qué hace `Image.open(i).convert('RGB')`**?**

1. `Image.open(i)`
   - Abre la imagen que está en la ruta `i`.
   - Esa imagen puede estar en diferentes modos de color:
     - `"RGB"` (ya en rojo, verde, azul).
     - `"L"` (escala de grises).
     - `"CMYK"` (cian, magenta, amarillo, negro, usado en impresión).
     - `"P"` (paleta de colores).
     - etc.

2. `.convert('RGB')`
   - Lo que hace es **forzar que la imagen esté en modo RGB** (es decir, con 3 canales: rojo, verde y azul).
   - Si la imagen ya estaba en RGB, no pasa nada.
   - Si estaba en otro formato (por ejemplo, escala de grises o CMYK), la transforma a RGB.

Entonces, ¿separa los canales?

👉 No exactamente.

- **No separa los tres canales en arrays distintos.**
- Lo que hace es **asegurarse de que la imagen tenga información en 3 canales (R, G y B).**

Después, cuando usas:

python — Copiar código

```python
r, g, b = img.getpixel((x, y))
```

recién ahí **extraes** el valor de cada canal **para un píxel específico.**

¿Por que Array en vez de seguir con lista?

🔹 1. Listas de Python (`[]`)

- Son **estructuras nativas** de Python.
- Pueden contener **cualquier tipo de dato**: números, strings, objetos, incluso mezclados.

```python
lista = [10, "hola", 3.14, True]
```

- **Flexibles**: puedes agregar o quitar elementos fácilmente con `.append()`, `.remove()`, etc.
- Pero… **no están optimizadas para cálculos numéricos**.
  Ejemplo: si quieres multiplicar todos los elementos por 2:

```python
lista = [1,2,3]
lista * 2   # => [1,2,3,1,2,3]  (repite la lista, no multiplica los valores)
```

🔹 2. Arrays de NumPy (`np.array`)

- Son estructuras específicas de la librería **NumPy**.
- Están pensados para **almacenar solo números** (enteros o decimales) en forma de vectores o matrices.
- Mucho más **rápidos y eficientes** para operaciones matemáticas.
- Ejemplo:

```python
import numpy as np
arr = np.array([1,2,3])
arr * 2   # => array([2,4,6]) ✅ multiplica cada elemento
```

- Soportan álgebra lineal, estadísticas, transformaciones de matrices, etc.

¿Por qué redimensionar (50×50, 100×100, etc.)?

Redimensionar una imagen significa cambiar cuántos píxeles la representan. Las razones por las que se hace en tu código son principalmente estas:

- **Velocidad y memoria**: menos píxeles → menos operaciones → todo corre más rápido y requiere menos RAM (importante si tienes muchas imágenes).
- **Uniformidad**: muchas tareas (regresión, ML, gráficos) requieren que todas las imágenes tengan el mismo tamaño.
- **Estimación robusta del color promedio**: reducir la imagen usando un buen filtro (p. ej. LANCZOS) actúa como un *suavizado* → elimina ruido y pequeñas variaciones, y deja una buena aproximación del color global.
- **Control de resolución espacial**: si te interesa sólo el color global (no detalles finos), no necesitas miles de píxeles; un 50×50 suele ser suficiente.
- **Requisitos de modelos**: redes neuronales preentrenadas esperan tamaños concretos (p. ej. 224×224); para esos casos redimensionas a lo que pida el modelo.

¿Qué pasa exactamente cuando reduces?

- El proceso de reducción (downsampling) normalmente **promedia** bloques de píxeles. Eso tiene dos consecuencias:
  - **Preserva el color medio global** (si la reducción usa interpolación que promedia), por lo que el `r_total/count` aproximará bien el color general.
  - **Pierde detalles pequeños** (manchas pequeñas, textos, bordes finos), porque esos detalles se mezclan con su vecindario.
- Es básicamente un **filtro pasabajas**: elimina altas frecuencias (bordes, ruido) y deja bajas frecuencias (tonalidad general).

¿Cómo afecta al error/precisión del promedio?

Si modelamos ruido en los canales como variaciones aleatorias con desviación típica σ, el **error estándar** del promedio de N píxeles es $\sigma/\sqrt{N}$.
Ejemplo numérico (ilustrativo):

- 50×50 → $N = 2500$. Si $\sigma = 30$: SE≈$30/\sqrt{2500} = 30/50 = 0.6$.
- 100×100 → $N = 10000$. SE≈$30/100 = 0.3$.
  Conclusión: más píxeles → menor incertidumbre estadística del promedio. Pero recuerda: la reducción mediante interpolación ya está *promediando bloques*, por lo que el efecto práctico suele ser que incluso 50×50 ya da un promedio muy razonable.

Yo quiero recortar y redimensionar imagen

🔹 Cómo recortar con Pillow (PIL)

Con `Image.crop()` puedes seleccionar un **rectángulo** de la imagen indicando sus coordenadas:

```python
from PIL import Image

# Abrir la imagen
img = Image.open("ROJO05B.jpg").convert("RGB")

# Definir la caja de recorte (left, top, right, bottom)
# OJO: coordenadas en píxeles
crop_box = (x_min, y_min, x_max, y_max)

# Recortar
img_cropped = img.crop(crop_box)

# Mostrar el recorte
img_cropped.show()
```

🔹 Ejemplo práctico

Si la imagen mide, digamos, 1200×800 píxeles y quieres recortar un cuadrado que encierre solo el color rosado en el centro:

```python
crop_box = (500, 200, 800, 600)  # izquierda, arriba, derecha, abajo
img_cropped = img.crop(crop_box)
```

Eso te dará solo esa parte.

🔹 Después del recorte

Una vez que tienes `img_cropped`, puedes aplicar tu función de promedio de color:

```python
common_color = most_common_used_color(img_cropped)
print("RGB promedio del recorte:", common_color)
```

De esta forma, el análisis se hará **solo en la región recortada** (donde está el líquido) y no en el fondo blanco ni en la parte negra del soporte.

Para pruebas en pc, usar seleccion con mouse, luego en el celular usar cuadrícula.

Método PIL de Python | Image.crop()

Última actualización: 17 de junio de 2021

PIL es la biblioteca de imágenes de Python que proporciona al intérprete de Python funciones de edición de imágenes. El método PIL.Image.crop() se utiliza para recortar una porción rectangular de cualquier imagen.

Sintaxis: *PIL.Image.crop(box = None).*
Parámetros:
box: *una tupla de 4 elementos que define las coordenadas de píxel izquierda, superior, derecha e inferior.*
Tipo de retorno: *Imagen (Devuelve una región rectangular como una tupla (izquierda, superior, derecha, inferior)).*
Retorno: *Un objeto* ***Imagen*** *.*

Las imagenes que me mandó el profe son de 4032x2268, buscar la región de color y definir cuadrado.

intenté enviar imagenes cortadas con Macro de imageJ pero es inconsistente

se me ocurrió una buena idea creo yo de como hacer en el computador para poder recortar la imagen con python, que aparezca un cuadrado predefinido y asi posicionar en la parte que quiero para recortar la imagen, para que solo se vea lo que necesito.

encontre en matplotlib un codigo para hacer todo eso y funciona genial archivo "pruebacuadrado.py" 19:40 pm 01/10/25.

02/10/25

Separar el codigo en partes para revisión mas detallada.

Colocar concetracion en la salida de rgb como archivo txt o csv, generar solo ese archivo.

Archivo de constantes de la regresion para cada r,g,b tambien txt o csv.

todavía falta analisis estadístico

Generar matriz ANOVA (calcula las estadisticas de si el ensayo es significativo o no)

Para guardar datos, generar matriz y guardar datos en archivos. Falta lo que tengo escrito en whatsapp.

RojoMetilo > Macro1(red).txt

```
a Area Mean Min Max
1 326144 209.324 199 219
2 326144 209.897 155 223
3 326144 212.317 194 221
4 326144 210.597 136 224
5 326144 218.904 161 236
```

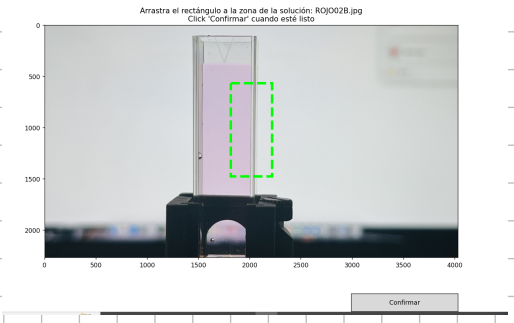
Agregar el valor predicho por la recta y ver la diferencia (residuo).

Histograma en imageJ, entre mas se asemeje a delta de dirac, mas me dice que es SOLO UN COLOR.

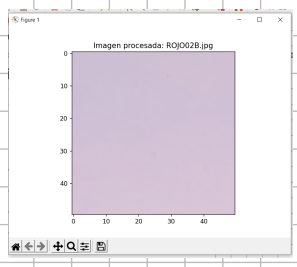
03/10/25

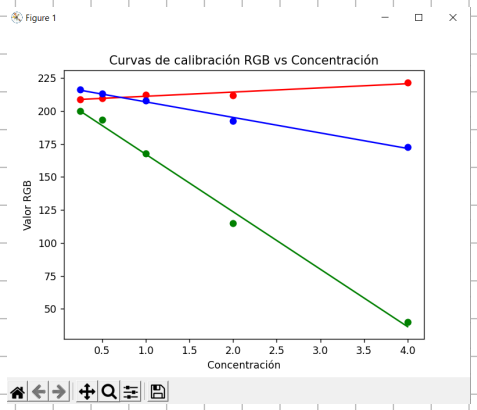
Olvidé poner que valores obtuve con el codigo...

Se hace esto para cada imagen



Esta es la imagen de la que obtiene cada r,g,b





-----Hola profe, estos son los valores de rgb de cada imagen para el Rojo metilo, están en orden, la imagen 1 es la que tiene menor intensidad y la 5 es la que tiene mayor intensidad

Procesando: ROJO01B.jpg
Los colores de la imagen ROJO01B.jpg son R,G,B: (208.9228, 199.9192, 216.2816)

Procesando: ROJO02B.jpg
Los colores de la imagen ROJO02B.jpg son R,G,B: (209.9088, 193.5736, 213.0768)

Procesando: ROJO03B.jpg
Los colores de la imagen ROJO03B.jpg son R,G,B: (212.3892, 167.7464, 207.9304)

Procesando: ROJO04B.jpg
Los colores de la imagen ROJO04B.jpg son R,G,B: (212.0512, 115.1532, 192.5616)

Procesando: ROJO05B.jpg
Los colores de la imagen ROJO05B.jpg son R,G,B: (221.6216, 40.048, 172.6828)

Valores del rojo metilo NOMINAL

x = [0.25, 0.5, 1, 2, 4]

$$\frac{y-d}{m} = x$$

------También si necesita las rectas para cada color, aquí están las tres con su coeficiente de correlación correspondiente.

El coeficiente de correlación para rojo es R^2 = 0.9242380638397744
La recta para rojo es y = 3.1804322580645152x + 208.04905

El coeficiente de correlación para verde es R^2 = 0.9940480309843314
La recta para verde es y = -43.59905591397849x + 210.86661666666663

El coeficiente de correlación para azul es R^2 = 0.9929097181310527
La recta para azul es y = -11.777726881720431x + 218.76211666666666

-------Por ultimo, estos son los colores en rgb de las imagenes de agua que me envió:

Procesando: Water01.jpg
Los colores de la imagen Water01.jpg son R,G,B: (186.96, 188.194, 203.4916)

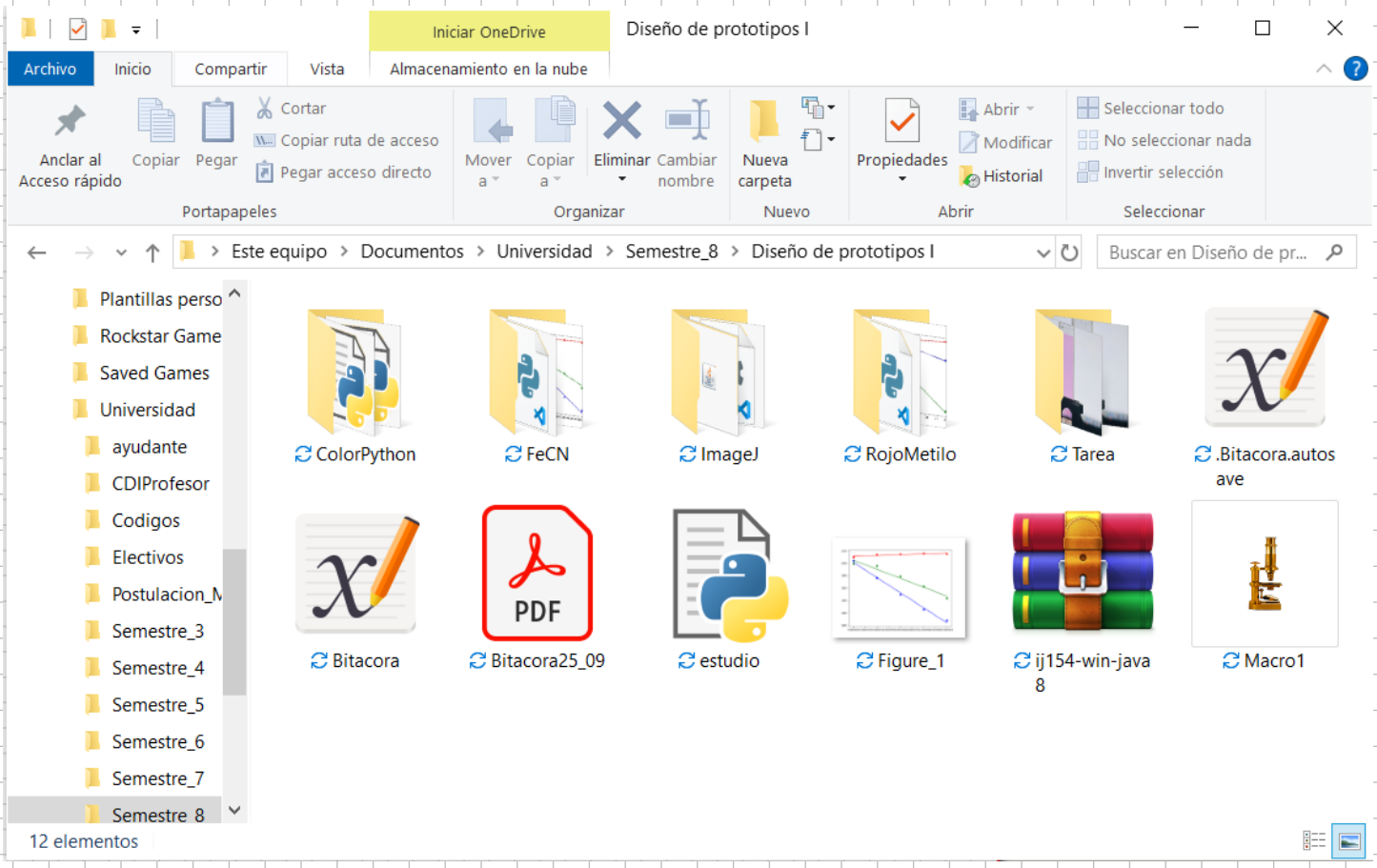
Procesando: Water02.jpg
Los colores de la imagen Water02.jpg son R,G,B: (187.126, 188.2916, 203.7488)

Procesando: Water03.jpg
Los colores de la imagen Water03.jpg son R,G,B: (187.7656, 188.8476, 204.1752)

Procesando: Water04.jpg
Los colores de la imagen Water04.jpg son R,G,B: (186.746, 187.8408, 203.5708)

Las imágenes estaban un poco sucias porque había manchas en los vasos, así que deben estar afectados los datos.

Carpetas creadas hasta el momento, carpeta color python tiene las ultimas actualizaciones



16/10/25

Haciendo todo el analisis estadistico

RESPUESTA CORTA:

Debes calcular la desviación estándar CON los datos originales (incluyendo negativos), NO con el valor absoluto.

```python
# ✅ CORRECTO
desv_std = np.std(residuos, ddof=1)  # Con negativos

# ❌ INCORRECTO
desv_std = np.std(np.abs(residuos), ddof=1)  # Con valores absolutos
```

Error individual = concetracion nominal - concentracion predicha (residuo)
Error promedio
Desviacion estandar
analisis de varianza (ANOVA)
R^2 (ya lo tengo)

Cosas a calcular...

```python
# Convertir listas a arrays
#si bien para cada recta tenemos una concentracion predicha distinta, para tener un unico valor de concentracion predicha por cad
conc_predicha = (np.array(cr)+np.array(cv)+np.array(ca))/3

print("Concentracion nominal:", c_nominal)

print("Concentracion predicha promedio:", conc_predicha)

print("error absoluto cn - cp en cada dato: +-", err_abs := c_nominal - conc_predicha) #error absoluto (Residuos)

print("error medio absoluto +-:", np.mean(np.abs(err_abs)))

print("desviacion estandar del error absoluto std (cn - cp):", np.std(err_abs, ddof=1)) #revisar bitacora explicacion ddof=1

print("error porcentual medio (relativo):", np.mean(np.abs(err_abs/c_nominal))*100, "%") #error relativo
```

Comparación de resultados:

Con `ddof=0` (tu código original - INCORRECTO):

```python
np.std(err_abs, ddof=0)
# Resultado: 0.0576... (divide por 5)
```

Con `ddof=1` (CORRECTO para muestra):

```python
np.std(err_abs, ddof=1)
# Resultado: 0.0648... (divide por 4)
```

¿Por qué `ddof=1`?

Tienes **5 muestras** (no toda la población de posibles concentraciones). Cuando trabajas con una **muestra**, necesitas la corrección de Bessel (`ddof=1`) para obtener un estimador **insesgado** de la varianza poblacional.

Fórmula:

Desviación estándar poblacional (ddof=0):

$$\sigma = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n}}$$

Desviación estándar muestral (ddof=1):

$$s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$$

Buscando como realizar ANOVA, haciendo codigo para calcular las otras partes estadisticas.

¡Sí! Hay varias formas de hacer ANOVA en Python. Te muestro las opciones:

Opción 1: Usando `scipy.stats` (más simple) ⭐

```python
from scipy import stats

# ANOVA de un factor
f_stat, p_value = stats.f_oneway(grupo1, grupo2, grupo3, ...)
```

Opción 2: Usando `statsmodels` (más completo, con tabla ANOVA)

```python
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

Concepto Excel	En el código
Suma de cuadrados (Regresión)	`ss_regresion`
Suma de cuadrados (Residuos)	`ss_residual`
Suma de cuadrados (Total)	`ss_total`
Promedio de los cuadrados	`cm_regresion` y `cm_residual`
F	`f_estadistico`
Valor crítico de F	`p_valor`
R^2	`r_cuadrado`

Hecho con claude.ai, revisar y comparar por mi cuenta...

```
============================================================
ANÁLISIS DE VARIANZA (ANOVA)
============================================================

--- PASO 1: SUMAS DE CUADRADOS ---
SS Total (SST):        9.3000000000
  → Variabilidad total de los datos

SS Regresión (SSM):  9.3210314126
  → Variabilidad EXPLICADA por el modelo

SS Residual (SSR):   0.0210314126
  → Variabilidad NO explicada (error)

Verificación: SST = SSM + SSR
  9.3000000000 = 9.3210314126 + 0.0210314126
  Diferencia: 0.0420628252687 08

--- PASO 2: GRADOS DE LIBERTAD ---
gl Total:      4
gl Regresión:  1
gl Residual:   3

--- PASO 3: CUADRADOS MEDIOS ---
MS Regresión (MSM): 9.3210314126
MS Residual (MSE):  0.0070104709

--- PASO 4: ESTADÍSTICO F ---
F = 1329.5870669299

--- PASO 5: VALOR P Y CONCLUSIÓN ---
p-valor = 0.0000453650

✓ CONCLUSIÓN: p < 0.05 → El modelo ES SIGNIFICATIVO
```

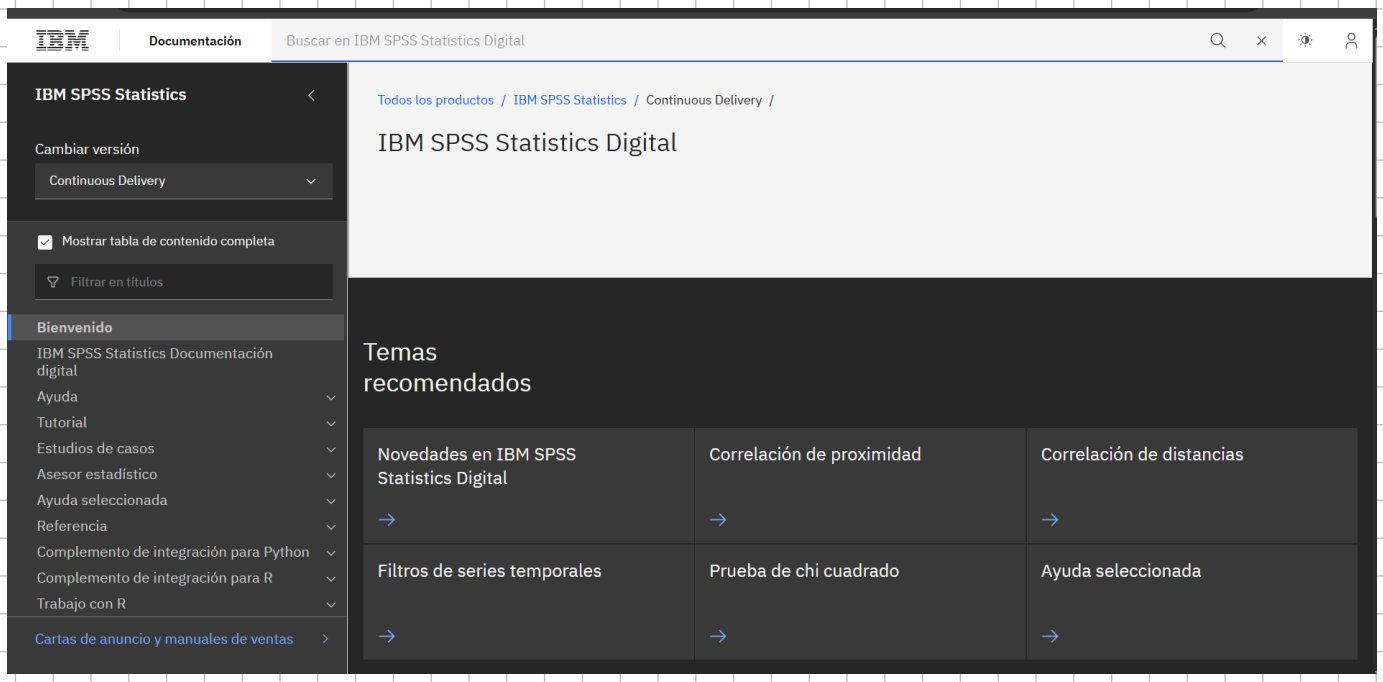
```
==============================================================================
TABLA ANOVA
==============================================================================
Fuente          SS                gl      MS                F               p-valor
------------------------------------------------------------------------------
Regresión       9.3210314126      1       9.3210314126      1329.587067     0.0000453650
Residual        0.0210314126      3       0.0070104709
Total           9.3000000000      4
==============================================================================
PS C:\Users\Diego\OneDrive - Universidad de Concepción\Documentos\Universidad\Semestre_8\Diseño de prototipos I>
```

Codigo para el analisis de datos teniendo las listas de valores de r,g,b

Listas necesarias para el análisis estadístico

```
import numpy as np
import matplotlib.pyplot as plt

#CONCETRACION NOMINAL (real)
c_nominal = np.array([0.25, 0.5, 1, 2, 4]) # total r,g,b

r = np.array([208.9228,209.9088,212.3892,212.0512,221.6216]) #valores mean para rojo
g = np.array([199.9192,193.5736,167.7464,115.1532,40.048]) #valores mean para verde
b = np.array([216.2816,213.0768,207.9304,192.5616,172.6828]) #valores mean para azul

cr = [] #concentracion rojo, verde, azul
cv = []
ca = []

for i in r: #calculo de concentracion a partir de mean para rojo
    x = (i - 208.04905) / 3.1804322580645152
    cr.append(x) #guarda valor en la lista creada
    #print("concentracion predicha recta roja:", x)

for i in g: #calculo de concentracion a partir de mean para verde
    x = (i - 210.86661666666663) / -43.59905591397849
    cv.append(x) #guarda valor en la lista creada
    #print("concentracion predicha recta verde:", x)

for i in b: #calculo de concentracion a partir de mean para azul
    x = (i - 218.76211666666666) / -11.777726881720431
    ca.append(x) #guarda valor en la lista creada
    #print("concentracion predicha recta azul:", x)

# Convertir listas a arrays #basicamente me deja los datos como los txt que tenia para poder hacer el ajuste lineal y hacer el promedio para comparar con cn
conc_predicha = np.array([np.array(cr[0]+cv[0]+ca[0])/3, np.array(cr[1]+cv[1]+ca[1])/3, np.array(cr[2]+cv[2]+ca[2])/3, np.array(cr[3]+cv[3]+ca[3])/3, np.array(cr[4]+cv[4]+ca[4])/3])

#conc_predicha = np.array(cr[0]+cv[0]+ca[0])/3, np.array(cr[1]+cv[1]+ca[1])/3, np.array(cr[2]+cv[2]+ca[2])/3, np.array(cr[3]+cv[3]+ca[3])/3, np.array(cr[4]+cv[4]+ca[4])/3
#es mejor tenerlo todo en ([...]) para que quede como array de 5 elementos

print("Concentracion nominal:", c_nominal)

print("Concentracion predicha promedio:", conc_predicha)

print("error absoluto cn - cp en cada dato: +-", err_abs := c_nominal - conc_predicha) #error absoluto (Residuos)

print("error medio absoluto +-:", np.mean(np.abs(err_abs)))

print("desviacion estandar del error absoluto std (cn - cp):", np.std(err_abs, ddof=1)) #revisar bitacora explicacion ddof=1

print("error porcentual medio (relativo):", np.mean(np.abs(err_abs/c_nominal))*100, "%") #error relativo
```

Pagina revisar como se hace ANOVA + ver los videos de youtube



18/10/2025

Revisando ANOVA.

```
  → El residuo pequeño indica buena predicción
  → ANOVA mide si TODOS los residuos son pequeños
```

El código que te di está CORRECTO ✅

Hace exactamente lo que debe hacer:

1. **Toma tus datos nominales** (reales): `c_nominal`
2. **Toma tus predicciones**: `conc_predicha`
3. **Calcula las diferencias** (residuos): `err_abs = c_nominal - conc_predic
4. **Descompone la variabilidad**:
   - Total (de c_nominal)
   - Explicada (por conc_predicha)
   - No explicada (residuos)
5. **Prueba estadística F**: ¿Es significativa la predicción?

tabla de mi codigo

Explicación breve:

- {valor:<15} → alineado a la izquierda dentro de un campo de ancho 15 caracteres.
- {valor:>15} → alineado a la derecha dentro de 15 caracteres.
- {valor:<18.10f} → para floats: ancho 18, alineado a la izquierda, con 10 decimales (f es formato de punto fijo).
- {valor:18.10f} o {valor:>18.10f} → ancho 18, 10 decimales, alineado a la derecha (recomendado para números).

24/10/2025

La idea es lograr esto:

Explicacion de como funciona el codigo

Cambios realizados:

1. Clase `InteractiveCrop` modificada:

```python
# ANTES: 1 rectángulo
self.rect = Rectangle(...)

# AHORA: 3 rectángulos
self.rectangles = []  # Lista de 3 rectángulos
colors = ['red', 'lime', 'yellow']  # Colores diferentes para distinguirlos
```

2. Posiciones iniciales de los 3 cuadrados:

```python
positions = [
    (x, height // 3),      # Cuadrado superior (rojo)
    (x, height // 2),      # Cuadrado central (verde lima)
    (x, 2*height // 3)     # Cuadrado inferior (amarillo)
]
```

3. Detección de cuál cuadrado se está moviendo:

```python
def on_press(self, event):
    # Detecta cuál de los 3 rectángulos fue clickeado
    for rect in self.rectangles:
        contains, _ = rect.contains(event)
        if contains:
            self.active_rect = rect  # Guarda cuál está activo
            break
```

4. Procesamiento de las 3 mediciones:

```python
# Para cada imagen:
r_temps = []  # Valores R de los 3 cuadrados
g_temps = []  # Valores G de los 3 cuadrados
b_temps = []  # Valores B de los 3 cuadrados

# Procesar cada cuadrado
for idx, coords in enumerate(coords_list):
    # ... procesar ...
    r_temps.append(valor_r)
    g_temps.append(valor_g)
    b_temps.append(valor_b)

# Calcular promedio
r_promedio = np.mean(r_temps)  # (c1 + c2 + c3) / 3
g_promedio = np.mean(g_temps)
b_promedio = np.mean(b_temps)
```

5. Visualización de las 3 regiones:

Ahora muestra las 3 regiones recortadas lado a lado para cada imagen, para que puedas verificar que seleccionaste bien.

Cómo usar:

1. **Aparecerán 3 cuadrados** de colores (rojo, verde lima, amarillo)
2. **Arrastra cada uno** a diferentes zonas de la solución (superior, centro, inferior)
3. **Click en "Confirmar"** cuando los 3 estén bien posicionados
4. El código calcula automáticamente el **promedio RGB** de los 3 cuadrados
5. Ese promedio es el que se usa para la regresión lineal

Ventaja:

Esto te da una medición más robusta porque promedias 3 zonas diferentes de la misma cubeta, reduciendo el efecto de:

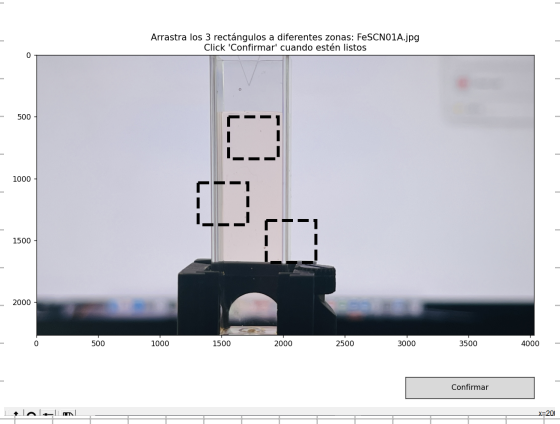
- Gradientes de color
- Burbujas
- Reflexiones locales
- Imperfecciones

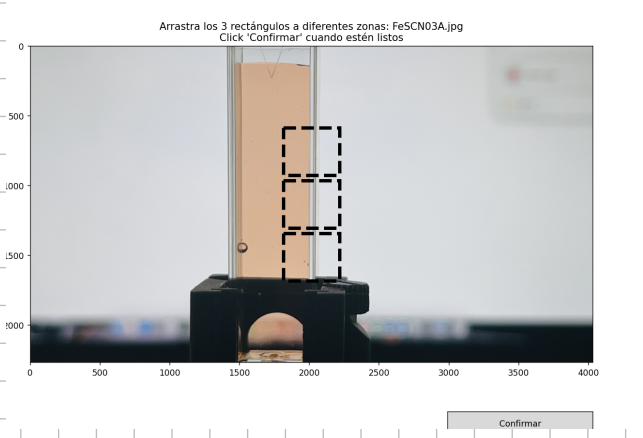
- plt.scatter(x, r, color="red", s=100, alpha=0.7, edgecolors='black', linewidth=1.5)
  - x, r: arrays con coordenadas X e Y. Si x tiene shape (n,1) usar x.flatten() o x.ravel() para pasar 1D.
  - color="red": color de relleno de los marcadores. Puede ser nombre, hex, o un array (para mapa de colores con c=... y cmap=...).
  - s=100: tamaño de los puntos en unidades de área (puntos^2). Puede ser array para tamaños variables.
  - alpha=0.7: transparencia (0 totalmente transparente, 1 opaco).
  - edgecolors='black': color del borde del marcador. Atención: para marcadores no rellenos algunos backends ignoran edgecolors.
  - linewidth=1.5: ancho del borde (también puede usarse linewidths=...).
- plt.xlabel(...), plt.ylabel(...), plt.title(...)
  - fontsize: tamaño de la fuente.
  - fontweight: 'normal','bold', etc.
  - puedes añadir color, fontfamily, loc (para title: 'left','right','center').
- plt.legend([...], loc='best')
  - Mejor práctica: pasar label="..." en cada plot/scatter y luego llamar plt.legend().
  - loc: posición ('best','upper right','lower left', o tuplas y bbox_to_anchor para posición exacta).
  - argumentos útiles: ncol (columnas), frameon (mostrar marco), framealpha (transparencia del fondo de la leyenda), fontsize, title.
- plt.grid(True, alpha=0.3)
  - activa la rejilla. Parámetros: linestyle='--', color='gray', which='both' (mayores y menores), linewidth.
- plt.tight_layout()
  - ajusta márgenes para evitar que etiquetas/leyenda se corten.
- plt.show()
  - muestra la figura en pantalla. Antes de guardar usar plt.savefig(..., bbox_inches='tight', dpi=300).

Documentacion de matplotlib.pyplot

31/10/2025

Avanzando en presentación en google y en modificar el codigo para que sean 3 cortes...





Estadistica para FeSCN hecha en python, luego esos datos output, colocados en IA para que simplemente dibuje las tablas de forma correcta.

Métrica	Valor (Tiocianato de Hierro)
Concentraciones nominales (cn)	[2.0×10^{-5}, 4.0×10^{-5}, 6.0×10^{-5}, 8.0×10^{-5} , 1.0×10^{-5}]
Concentraciones predichas (cp)	[1.5923×10^{-5}, 4.2473×10^{-5}, 6.2227×10^{-5}, 8.4427×10^{-5} , 9.4946×10^{-5}]
Error absoluto (cn − cp)	[4.0764×10^{-6}, -2.4739×10^{-6}, -2.2270×10^{-6}, -4.4275×10^{-6}, 5.0532×10^{-6}]
Error medio absoluto (MAE)	3.6516680×10^{-6}
Desviación estándar del error	4.2672274×10^{-6}
Error porcentual medio (MAPE)	8.173362 %

FeCN	SS (Suma de Cuadrados)	gl (grados de libertad)	MS (Media de Cuadrados)	F	p-valor
Regresión	0.0000000041	1	0.0000000041	167.751610	0.0009936382
Residual	0.0000000001	3	0.00000000001	—	—
Total	0.0000000040	4	—	—	—

RECUERDA

SS (Sum of Squares): mide la variación total de los datos.

SS Regresión: variación explicada por el modelo lineal (concentración → valor RGB).

SS Residual: variación que el modelo no logra explicar (error).

gl (grados de libertad): cantidad de valores independientes usados en cada cálculo.

MS (Mean Square): SS dividido por gl.

F: relación entre MS Regresión / MS Residual → indica qué tan bueno es el modelo respecto al error. muy superior a 1 (modelo explica mucho más que el error).

p-valor: probabilidad de que la relación sea producto del azar. Si $p < 0.05$ → el modelo es estadísticamente significativo.

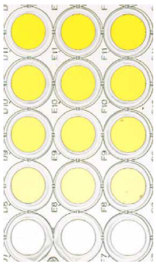


Figure 6. Contrast-enhanced grayscale transparency scanner image of the yellow solutions in a flat-bottom transparent plastic microtiter plate (triplicate aliquots, 200 µL) made using a transparency scanner (Hewlett-Packard Scanjet 7400c). Top to bottom rows: 100%, 75%, 50%, 25%, 0% relative concentration. (See page 515 as well as the online PDF for a color version.)

Durante 04, 05 y 06 de Noviembre se dan ultimas actualizaciones a la presentación.
Siguiente paso es tomar la foto de esta forma

Paper Kohl.