

Trabajo Especial Ejercicios con Iris

(septiembre, 2021)

1st Guerra Silva Erick Iván
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
erickivanguerra0.0@gmail.com

2nd Lázaro Martínez Abraham Josué
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
abrahamlazaro@comunidad.unam.mx

3rd Martínez Gutiérrez Carlos Giovanni
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
cgiovanni@comunidad.unam.mx

Resumen—Este documento presenta los resultados obtenidos para los ejercicios planteados en la tarea para la manipulación básica del conjunto de datos de flores Iris .

Índice de Términos—Iris, Python, pandas, filas, columnas, gráfica, percentil, media, desviación.

I. INTRODUCCIÓN

El conjunto de datos flor Iris o conjunto de datos iris de Fisher es un conjunto de datos multivariante introducido por Ronald Fisher en su artículo de 1936. A veces, se llama Iris conjunto de datos de Anderson porque Edgar Anderson colecciónó los datos para cuantificar la variación morfológica de la flor Iris de tres especies relacionadas.[1]

El conjunto de datos contiene 50 muestras de cada una de tres especies de Iris (Iris setosa, Iris virginica e Iris versicolor). Se midió cuatro rasgos de cada muestra: el largo y ancho del sépalo y pétalo, en centímetros. Basado en la combinación de estos cuatro rasgos, Fisher desarrolló un modelo discriminante lineal para distinguir entre una especie y otra.[2]

Regresión logística simple y múltiple.

La Regresión Logística Simple, desarrollada por David Cox en 1958, es un método de regresión que permite estimar la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa. Una de las principales aplicaciones de la regresión logística es la de clasificación binaria, en el que las observaciones se clasifican en un grupo u otro dependiendo del valor que tome la variable empleada como predictor.

La regresión logística múltiple es una extensión de la regresión logística simple. Se basa en los mismos principios que la regresión logística simple pero ampliando el número de predictores. Los predictores pueden ser tanto continuos como categóricos.[3]

Regresión lineal.

El análisis de la regresión lineal se utiliza para predecir el valor de una variable según el valor de otra. La variable que desea predecir se denomina variable dependiente. La variable que está utilizando para predecir el valor de la otra variable se denomina variable independiente.[4]

Matriz de confusión.

Una matriz de confusión es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real., o sea en términos prácticos nos permite ver qué tipos de aciertos y errores está teniendo nuestro modelo a la hora de pasar por el proceso de aprendizaje con los datos.[5]

A) pandas.

Pandas es una librería de Python especializada en el manejo y análisis de estructuras de datos.

Las principales características de esta librería son:

- Define nuevas estructuras de datos basadas en los arrays de la librería NumPy pero con nuevas funcionalidades.
- Permite leer y escribir fácilmente ficheros en formato CSV, Excel y bases de datos SQL.
- Permite acceder a los datos mediante índices o nombres para filas y columnas.
- Ofrece métodos para reordenar, dividir y combinar conjuntos de datos. [6]

B) matplotlib

Matplotlib es una librería de Python especializada en la creación de gráficos en dos dimensiones.

Permite crear y personalizar los tipos de gráficos más comunes, entre ellos:

RECONOCIMIENTO DE PATRONES.

- Diagramas de barras
- Histograma
- Diagramas de sectores
- Diagramas de caja y bigotes [7]

C) seaborn

Seaborn es una librería para Python que permite generar fácilmente elegantes gráficos. Seaborn está basada en matplotlib y proporciona una interfaz de alto nivel que es realmente sencilla de aprender.[8]

D) sklearn

sklearn es una biblioteca orientada al machine y deep learning en Python. Esta biblioteca o paquete contiene muchas funciones auxiliares, conjuntos de datos de prueba y modelos neuronales que podemos utilizar. Cuenta también con algoritmos de clasificación regresión, clustering y reducción de dimensionalidad. También es compatible con otras bibliotecas como NumPy, SciPy y Matplotlib.

Una de las principales ventajas de esta biblioteca es la variedad de herramientas y algoritmos que nos permiten aprender los conceptos básicos y no tan básicos de la ciencia de datos.[9]

En específico, de Scikit-Learn utilizamos principalmente el módulo de regresión, con un modelo de regresión logística.

II. OBJETIVO

Al término de esta tarea, el alumno conocerá la definición del conjunto de datos iris, así como algunas manipulaciones básicas, para el procesamiento de los mismos.

III. RESULTADOS

A continuación, se presentan los ejercicios desarrollados en esta tarea. Antes de realizar los ejercicios se realizó la descarga y descompresión de los archivos para poder trabajar con ellos. Además, se cargaron las librerías necesarias para la manipulación y visualización de los datos como es el caso de la librería Pandas, NumPy, Scipy, Matplotlib y seaborn.

A) Ejercicios Básicos

1. Cargue los datos iris en un data frame (pandas) e imprima la forma de los datos, tipo y las 10 primeras filas de los datos. Fuente de datos: <https://archive.ics.uci.edu/ml/datasets/Iris>.

<https://aprendeia.com/machine-learning-clasificador-flor-iris-python/>

Para la carga de datos Iris en un Dataframe primero se importó la librería pandas. Posteriormente, se definió la ruta donde se encontraban los datos. Adicionalmente, se definieron los headers para después ser agregados en el dataset.

```
# definimos la ruta para cargar los datos
```

```
rutaDatos = "/content/iris.data"

# definimos los headers, el archivo no los tiene
headers = ["sepal length","sepal width","petal length","petal width","class"]

dictRename = {i:headers[i] for i in range(len(headers))}
```

Una vez definidos los headers procedimos a leer el archivo *iris.data* el cual contenía los datos del conjunto de flores iris. Esto se realizó utilizando la función *read_csv*, algo interesante que notamos es que la lectura se realizó de forma correcta a pesar de que se esperaba un archivo con extensión .csv y se obtuvo un archivo con extensión .data.

```
# cargamos la información en un dataframe
df = pd.read_csv(rutaDatos,header=None)
```

Una vez leídos los datos, se agregaron los headers creados anteriormente con la función *rename* de pandas.

```
# cambiamos los headers
df.rename(columns=dictRename,inplace=True)
```

El siguiente paso era imprimir algunas características de los datos como es el caso del número de datos, el número de columnas, los tipos de datos y una pequeña muestra. Para obtener el número de datos se utilizó la función *size* sobre el dataframe que contenía los datos, para los renglones y columnas se utilizó la función *shape*, para el tipo de datos se realizó un ciclo for para poder obtener el tipo de datos por cada columna utilizando la función *dtype*. Finalmente, para la muestra de datos se utilizó slicing sobre el dataframe, mostrando así los 10 primeros renglones de los datos.

```
# Forma (shape) de los datos
print("Número de datos:",df.size)
row,col = df.shape

for i in df.keys():
    print(i,":",df[i].dtype)

# primeras 10 columnas
print(df[:10])
```

A continuación, se presentan los resultados obtenidos.

RECONOCIMIENTO DE PATRONES.

```

Forma de los datos
Número de datos: 750
Columnas: 5
Filas: 150

Tipo de datos por columna
sepal length : float64
sepal width : float64
petal length : float64
petal width : float64
class : object

Primeras 10 filas
   sepal length  sepal width  petal length  petal width      class
0           5.1         3.5        1.4       0.2  Iris-setosa
1           4.9         3.0        1.4       0.2  Iris-setosa
2           4.7         3.2        1.3       0.2  Iris-setosa
3           4.6         3.1        1.5       0.2  Iris-setosa
4           5.0         3.6        1.4       0.2  Iris-setosa
5           5.4         3.9        1.7       0.4  Iris-setosa
6           4.6         3.4        1.4       0.3  Iris-setosa
7           5.0         3.4        1.5       0.2  Iris-setosa
8           4.4         2.9        1.4       0.2  Iris-setosa
9           4.9         3.1        1.5       0.1  Iris-setosa

```

Figura 1. Información sobre el conjunto de datos

2. Imprima las llaves y el número de filas y de columnas.

Para la impresión de llaves o headers se utilizó la función *keys* y para la impresión de filas y columnas se utilizó la función *shape* como en el ejercicio anterior.

```

# llaves
print(list(df.keys()))

# Número de filas y columnas
row,col = df.shape
print("{} filas, {} columnas".format(row,col))

```

A continuación, se presentan los datos obtenidos.

```

Llaves de la tabla
['sepal length', 'sepal width', 'petal length', 'petal width', 'class']

Número de filas y columnas
150 filas, 5 columnas

```

Figura 2. Llaves, filas y columnas del conjunto de datos

3. Obtenga el número de muestras faltantes o Nan.

Para este ejercicio se operó sobre el dataframe, utilizando la función *isnull* para encontrar los elementos nulos o NaN en los datos y la función *sum* para realizar un conteo.

```

# número de nulos totales
print("Número de NaN:",df.isnull().sum().sum())

```

Primero se identificó a los elementos faltantes con la función *isnull*, con la primera función *sum* se obtuvo el número de nulos por columna y finalmente, el segundo *sum* se utilizó para sumar los nulos de cada columna y así obtener el número total de nulos en el conjunto de datos.

Podemos observar que no se encontró ningún elemento NaN en el conjunto de datos.

Número de NaN: 0

Figura 3. Número de elementos NaN en el conjunto de datos

4. Cree un arreglo 2-D de tamaño 5x5 con unos en la diagonal y ceros en el resto. Convierta el arreglo NumPy a una matriz dispersa de ScyPy en formato CRS. Nota: una matriz se considera dispersa cuando el porcentaje de ceros es mayor a 0.5.

Para este ejercicio lo primero que se realizó fue importar la librería Numpy para poder manipular matrices y SciPy para poder utilizar matrices dispersas.

Lo primero que se hizo en este ejercicio fue crear una matriz identidad de dimensión 5x5, es decir, una matriz con ceros excepto en la diagonal principal donde se tienen unos. Esto se realizó utilizando la función *identity* de NumPy.

```
mat = np.identity(5)
```

Una vez creada esta matriz, se calculó el porcentaje de ceros para verificar que se cumplía la condición para ser una matriz dispersa, es decir, el porcentaje de ceros era mayor a 0.5, lo cual se cumplió obteniendo un porcentaje de ceros de 0.8.

```
print("\nPorcentajes de ceros:",
(mat == 0).sum()/mat.size)
```

Finalmente, a partir de la matriz identidad creada anteriormente, se creó una matriz dispersa utilizando la función *csr_matrix* de SciPy. [10]

```
s = csr_matrix(mat)
```

Algo interesante que pudimos observar es que este tipo de matrices no almacena los valores uno detrás de otro, más bien, puede utilizar diferentes estructuras como:

- Diccionarios
- Lista de listas
- Lista de coordenadas

Después de ejecutar la celda pudimos observar que SciPy maneja las matrices dispersas como lista de coordenadas, en donde se tiene la posición y el valor en esa posición. A continuación, se presentan los resultados obtenidos.

RECONOCIMIENTO DE PATRONES.

```
Matriz de 5x5 con 1's en la diagonal (matriz identidad 5)
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

Porcentajes de ceros: 0.8

```
Matriz dispersa
 (0, 0)      1.0
 (1, 1)      1.0
 (2, 2)      1.0
 (3, 3)      1.0
 (4, 4)      1.0
```

Figura 4. Matriz y matriz dispersa obtenida

5. Muestre estadísticas básicas como percentil, media, mínimo, máximo y desviación estándar de los datos. Use `describe` para ello. Imprima sólo la media y la desviación estándar.

Para este ejercicio se utilizó la función `describe` asociada al dataframe, esta función regresa un conjunto de medidas estadísticas relevantes por cada columna del dataframe como es el caso de un conteo total de renglones, la media, la desviación estándar, el valor mínimo, el valor máximo y los percentiles 25, 50 y 75.

```
stats = df.describe()
```

Sin embargo, en este caso únicamente se pide imprimir la media y la desviación estándar, para esto se utiliza la función `iloc` en el conjunto de medidas estadísticas obtenidas. La función `iloc` nos va a permitir realizar un filtrado en los datos.

```
print(stats.iloc[[1,2],[0,1,2,3]])
```

En la línea de código anterior se puede observar que la primera lista con los índices 1 y 2 corresponden a la media y la desviación estándar, mientras que la segunda lista hace referencia a las columnas que desean ser impresas, en este caso serían las 4 columnas de largo y ancho de los pétalos y sépalos. A continuación, se presentan los resultados obtenidos.

Estadísticas básicas de los datos				
	LargoSepalo	AnchoSepalo	LargoPetalo	AnchoPetalo
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Media y desviación				
	LargoSepalo	AnchoSepalo	LargoPetalo	AnchoPetalo
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161

Figura 5. Medidas estadísticas del conjunto de datos

6. Obtenga el número de muestras para cada clase.

Para este ejercicio primero se agrupó el dataframe según los tipos de clase (Iris-setosa, Iris-Virginica, Iris-Versicolor), esto se realizó utilizando la función `groupby` y especificando el criterio, en este caso fue la clase.

```
# se agrupan los datos de acuerdo a la última
# columna
grupos = df.groupby(df["class"])
```

Posteriormente, se realizó un ciclo para realizar un conteo por cada clase e imprimir el total de muestras de cada una.

```
tipos = set(df["class"])
for tipo in tipos:
    print(tipo, "=", (df["class"]==tipo).sum())
```

Al ejecutar este código obtuvimos que cada clase consta de 50 muestras.

```
Número de muestras por clase
Iris-setosa = 50
Iris-virginica = 50
Iris-versicolor = 50
```

Figura 6. Número de muestras por cada clase de iris

7. Añada un encabezado a los datos usando los nombres en `iris.names` y repita el ejercicio anterior.

Este ejercicio consistió en cambiar los headers de los datos y realizar nuevamente el ejercicio anterior. Lo primero que se realizó fue definir los nuevos headers que tendrían los datos, para cambiar los headers, únicamente al atributo `columns` del dataframe se asignó la lista de nuevos headers.

```
headers
=['LargoSepalo', 'AnchoSepalo', 'LargoPetalo',
 'AnchoPetalo', 'Clase']
df.columns=headers
```

A continuación, se presentan los resultados obtenidos con los nuevos headers.

Cambiando headers					
	LargoSepalo	AnchoSepalo	LargoPetalo	AnchoPetalo	Clase
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
Número de muestras por Clase
Iris-setosa = 50
Iris-versicolor = 50
Iris-virginica = 50
```

Figura 7. Información sobre el conjunto de datos con los nuevos headers

8. Imprima las diez primeras filas y las dos primeras

RECONOCIMIENTO DE PATRONES.

columnas del data frame usando los índices de las columnas.

Para este ejercicio se utilizó la función *iloc* para especificar las filas y columnas a imprimir.

```
print(df.iloc[0:10, 0:2])
```

En este caso, se indica que se impriman las 10 primeras filas y las 2 primeras columnas, obteniendo la siguiente salida.

```
Primeras 10 filas y primeras dos columnas de los datos
   SepalLength  SepalWidth
0         5.1        3.5
1         4.9        3.0
2         4.7        3.2
3         4.6        3.1
4         5.0        3.6
5         5.4        3.9
6         4.6        3.4
7         5.0        3.4
8         4.4        2.9
9         4.9        3.1
```

Figura 8. Primeras diez filas y dos primeras columnas del conjunto de datos

B) Ejercicios de visualización

Utilizando matplotlib y/o seaborn

1. Cree una gráfica de barras que muestre la media, mínimo y máximo de todos los datos.

Para este ejercicio de visualización se utilizó la librería Matplotlib. Lo primero que se realizó fue nuevamente obtener las estadísticas del conjunto de datos con la función *describe* y con la función *iloc* se filtró la media, el mínimo y el máximo de todos los datos.

```
# obteniendo estadísticas de nuevo
stats = df.describe()

# seleccionando media, min y max
info = stats.iloc[[1,3,7],[0,1,2,3]]
```

Posteriormente, se definieron listas de las estadísticas por cada característica de las clases (LargoPetalo, AnchoPetalo, LargoSepalo, anchoSepalo) y así poder graficar de manera independiente cada conjunto de datos.

```
# etiquetas
etq = ["media", "min", "max"]

# series de datos
sl = list(info["LargoSepalo"])
sw = list(info["AnchoSepalo"])
pl = list(info["LargoPetalo"])
pw = list(info["AnchoPetalo"])
```

Una vez con los conjuntos definidos, se configuró una gráfica de matplotlib para poder agregar las 3 medidas para cada

característica. Para generar la gráfica de barras se utilizó la función *bar* de matplotlib. A continuación, se presenta la gráfica obtenida.

Información				
	LargoSepalo	AnchoSepalo	LargoPetalo	AnchoPetalo
mean	5.843333	3.054	3.758667	1.198667
min	4.300000	2.000	1.000000	0.100000
max	7.900000	4.400	6.900000	2.500000

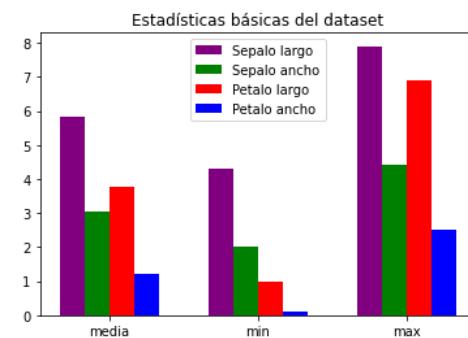


Figura 9. Gráfica de barras de la media, mínimo y máximo de las características de los pétalos y sépalos

En esta gráfica podemos observar que el ancho y largo de los sépalos es más grande que el ancho y largo de los pétalos.

2. Muestre la frecuencia de las tres especies como una gráfica de pastel.

En este ejercicio nuevamente utilizamos la librería matplotlib para realizar el gráfico de pastel. Lo que hicimos en este ejercicio fue obtener las etiquetas de las 3 clases de Iris y guardarlas en una lista para después calcular las frecuencias de cada clase en el dataframe realizando un conteo con la función *sum*.

```
# sacamos las etiquetas
etiquetas = list(set(df["Clase"]))

# sacamos las frecuencias de cada etiqueta
frecuencias = [(df["Clase"]==tipo).sum() for tipo in etiquetas]
```

Una vez obtenidas las frecuencias, procedimos a graficar utilizando la función *pie* a la que pasamos como parámetros las frecuencias de cada clase y las etiquetas como labels, además de que se configuraron los ejes y el título del gráfico

```
plt.pie(frecuencias, labels=etiquetas)
plt.axis("equal")
plt.title("Frecuencia de clases")
plt.show()
```

A continuación, se agrega el gráfico obtenido donde podemos observar que hay la misma cantidad de datos para cada clase.

RECONOCIMIENTO DE PATRONES.

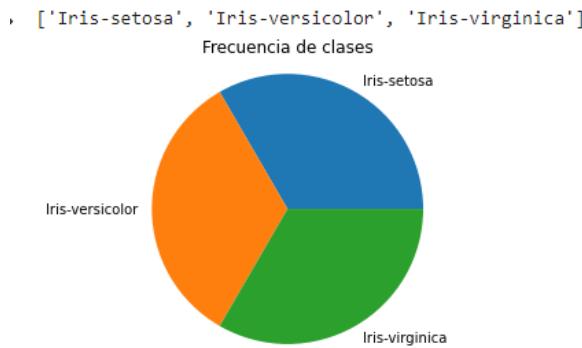


Figura 10. Gráfica de pastel de la frecuencia de las clases

3. Cree una gráfica que muestre la relación entre la longitud y ancho del sépalo de las tres especies conjuntamente.

Para este ejercicio nuevamente se utilizó la librería matplotlib. Lo primero que se realizó fue crear dos listas a partir del dataframe, una con los datos de la longitud del sépalo y otra con el ancho del sépalo.

```
sepalLength = list(df["LargoSepalo"])
sepalWidth = list(df["AnchoSepalo"])
```

Una vez creadas las listas se graficó utilizando la función *plot*, se graficó la relación entre el ancho y el largo del sépalo para cada especie de Iris, además de que se realizaron las configuraciones para los ejes y el título del gráfico.

```
plt.plot(sepalLength[:50],sepalWidth[:50],label="Iris-versicolor",color="blue")

plt.plot(sepalLength[50:100],sepalWidth[50:100],label="Iris-setosa",color="red")

plt.plot(sepalLength[100:150],sepalWidth[100:150],label="Iris-virginica",color="black")

plt.legend()
plt.title("Relación longitud-ancho de sepalo")
plt.xlabel("Longitud Sepalo [cm]")
plt.ylabel("Ancho Sepalo [cm]")
plt.show()
```

A continuación, se presenta el gráfico obtenido.

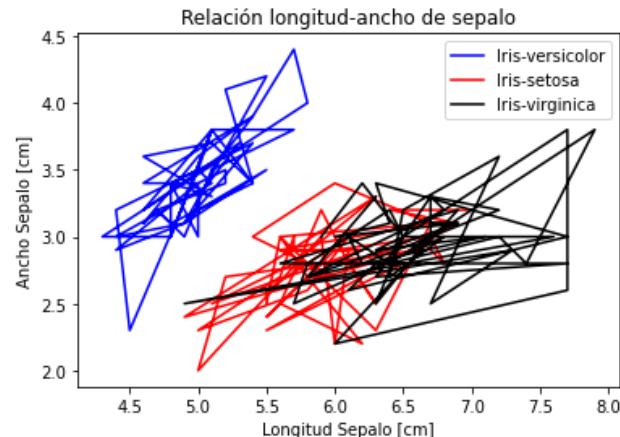


Figura 11. Gráfica de la relación entre la longitud y el ancho del sépalo

4. Obtenga los histogramas de las variables SepalLength, SepalWidth, PetalLength y PetalWidth.

Para generar los histogramas se utilizó la función *hist* de matplotlib. A esta función se pasó como parámetros los datos del dataframe para cada característica.

```
plt.hist(df["LargoSepalo"],50,color="blue")
plt.title("Longitud Sepalo")
plt.show()
```

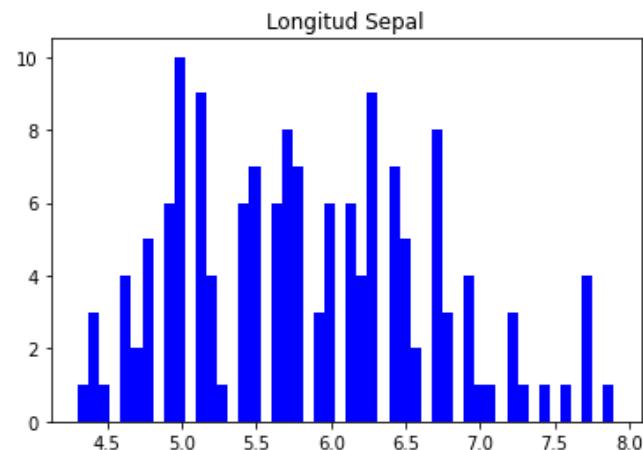


Figura 12. Histograma de la longitud del sépalo

RECONOCIMIENTO DE PATRONES.

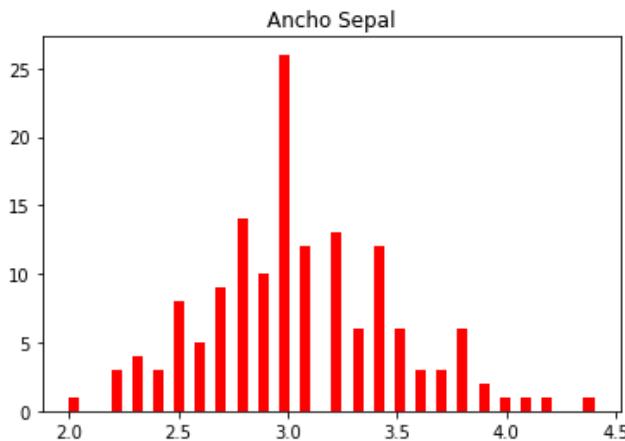


Figura 13. Histograma del ancho del sépalo

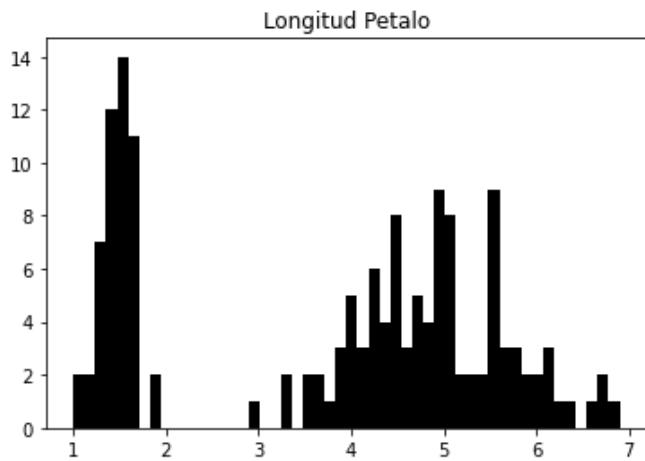


Figura 14. Histograma de la longitud del pétalo

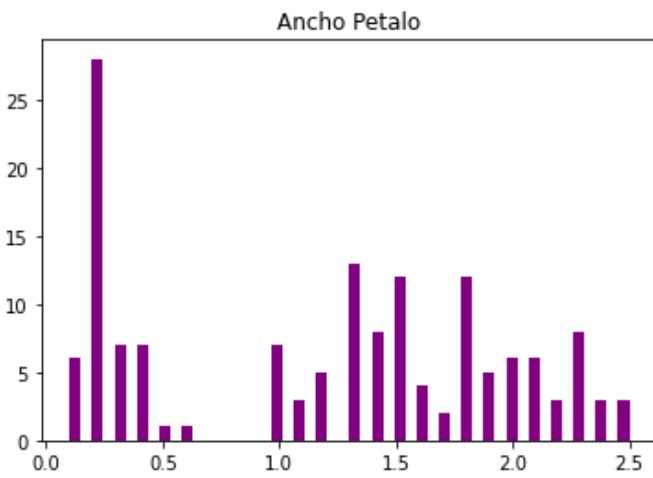


Figura 15. Histograma del ancho del pétalo

5. Cree gráficas de dispersión usando pairplot de seaborn y muestre con distintos colores las tres especies en las gráficas de dispersión.

Para este ejercicio se utilizó la librería seaborn. Para crear las

gráficas de dispersión de las 3 especies de iris se utilizó la función *pairplot* a la que se pasó como parámetro el dataframe y se distinguieran los datos de las tres clases de iris.

```
sns.pairplot(df, hue="Clase", markers=["o", "s", "D"])
```

Con esta gráfica podemos observar la relación entre las características del sepalo y pétalo para cada clase de iris

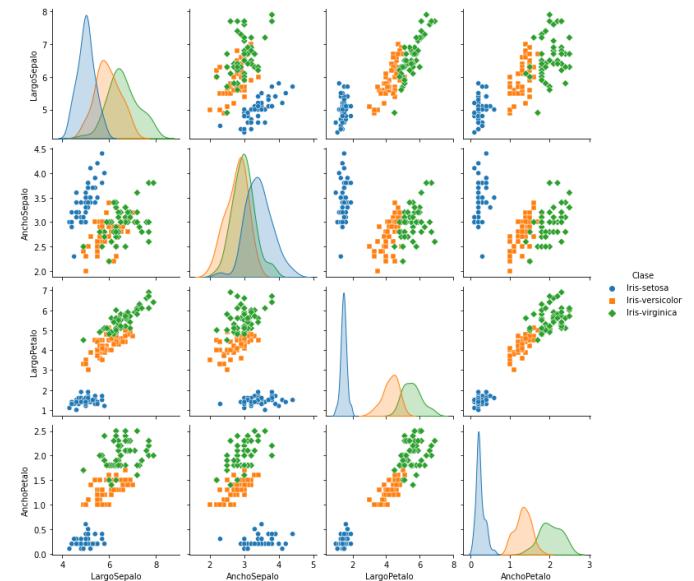


Figura 16. Gráficas de dispersión de las características de las 3 clases de iris

6. Cree una gráfica usando jointplot de seaborn para mostrar la dispersión entre la longitud y ancho del sépalo y las distribuciones de estas dos variables.

Para este ejercicio nuevamente se utilizó la librería seaborn. En este caso se utilizó la función *jointplot* para poder mostrar la dispersión de los datos entre la longitud y ancho del sépalo al igual que las distribuciones de ambas variables.

```
sns.jointplot(data=df, x="LargoSepalo", y="AnchoSepalo")
```

En la gráfica resultante podemos observar que en la parte superior se encuentra la distribución del largo del sepalo y del lado derecho se encuentra la distribución del ancho del sepalo, mientras que en la parte central del gráfico se observa la dispersión entre ambas variables.

RECONOCIMIENTO DE PATRONES.

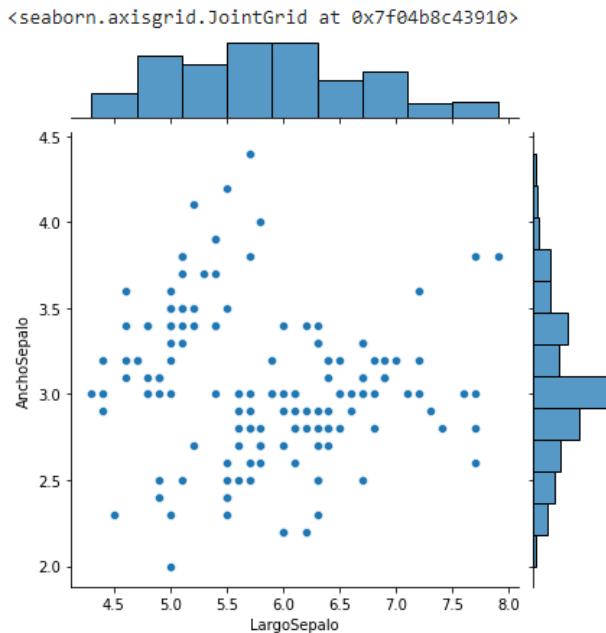


Figura 17. Gráfica de dispersión entre la longitud y el ancho del sépalo y las distribuciones de ambas variables

7. Repita el ejercicio anterior, pero esta vez usando joinplot con kind="hexbin".

La única diferencia entre este ejercicio y el anterior es que se agregó el parámetro `kind = "hexbin"` en la función `joinplot`, este parámetro representa el tipo de gráfico de dispersión que se utilizaría, en este caso sería un gráfico de tipo hexbin con el que observamos las zonas con mayor concentración de los datos de un color más oscuro mientras donde no hay mucha concentración se encuentra con un color más claro e incluso blanco cuando no se tiene ningún dato en ese punto.

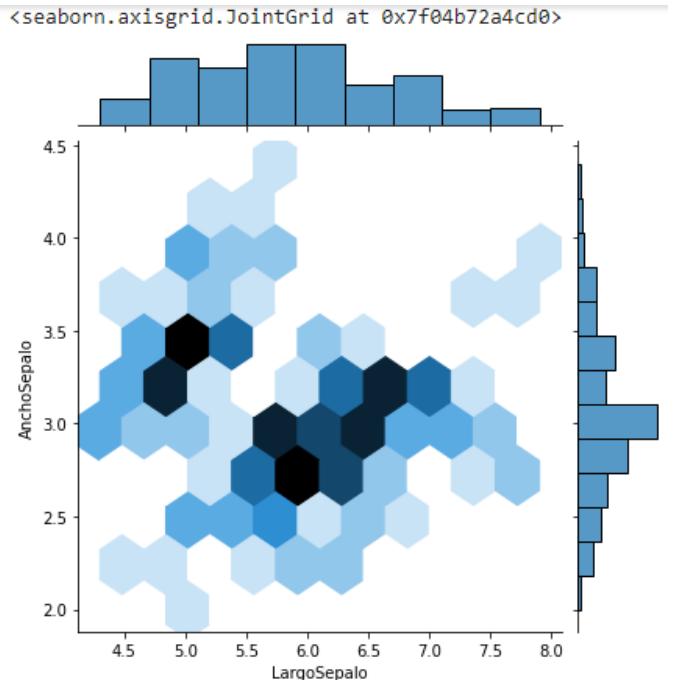


Figura 18. Gráfica de dispersión hexbin entre la longitud y el ancho del sépalo y las distribuciones de ambas variables

C) Ejercicios de regresión logística

1. Muestre los percentiles, media y desviación estándar de cada especie ('Iris-setosa', 'Iris-versicolor' e 'Iris-virginica').

Para este ejercicio nuevamente se utilizó la función `describe` para obtener los percentiles, la media y la desviación estándar para cada especie de iris. Lo primero que se realizó fue crear una copia del dataframe, con esta copia y con la función `loc`, se obtuvieron los datos para cada clase de iris y posteriormente realizar el cálculo de las medidas para cada clase.

```
copia = df.copy()
copia.set_index("Clase", inplace = True)
ISe = copia.loc["Iris-setosa"]
IVe = copia.loc["Iris-versicolor"]
IVi = copia.loc["Iris-virginica"]
```

Una vez con los datos de cada clase, se utilizó un ciclo para realizar el cálculo de las medidas para todas las clases e imprimirlas. Se hizo uso de la función `iloc` para encontrar las características deseadas .

```
for especie in ["Iris-setosa", "Iris-versicolor",
                "Iris-virginica"]:
    print("Especie:", especie)
    aux = copia.loc[especie]
    print(aux.describe().iloc[[4, 5, 6, 1, 2]], "\n")
```

RECONOCIMIENTO DE PATRONES.

```

Especie: Iris-setosa
    LargoSepalo  AnchoSepalo  LargoPetalo  AnchoPetalo
25%      4.80000     3.125000     1.400000     0.20000
50%      5.00000     3.400000     1.500000     0.20000
75%      5.20000     3.675000     1.575000     0.30000
mean     5.00600     3.418000     1.464000     0.24400
std      0.35249     0.381024     0.173511     0.10721

Especie: Iris-versicolor
    LargoSepalo  AnchoSepalo  LargoPetalo  AnchoPetalo
25%      5.60000     2.525000     4.000000     1.20000
50%      5.90000     2.800000     4.350000     1.30000
75%      6.30000     3.000000     4.600000     1.50000
mean     5.93600     2.770000     4.260000     1.326000
std      0.516171    0.313798    0.469911    0.197753

Especie: Iris-virginica
    LargoSepalo  AnchoSepalo  LargoPetalo  AnchoPetalo
25%      6.22500     2.800000     5.100000     1.80000
50%      6.50000     3.000000     5.550000     2.00000
75%      6.90000     3.175000     5.875000     2.30000
mean     6.58800     2.974000     5.552000     2.02600
std      0.63588     0.322497    0.551895    0.27465

```

Figura 19. Tabla sobre algunos datos de las diferentes clases

```

media de Iris-setosa
0   5.006
1   3.418
2   1.464
3   0.244
Name: mean, dtype: float64
desviación Iris-setosa
0   0.352490
1   0.381024
2   0.173511
3   0.107210
Name: std, dtype: float64

```

Figura 20. Media y desviación de la clase Iris-setosa

```

media de Iris-versicolor
0   5.936
1   2.770
2   4.260
3   1.326
Name: mean, dtype: float64
desviación Iris-versicolor
0   0.516171
1   0.313798
2   0.469911
3   0.197753
Name: std, dtype: float64

```

Figura 21. Media y desviación de la clase Iris-versicolor

```

media de Iris-virginica
0   6.588
1   2.974
2   5.552
3   2.026
Name: mean, dtype: float64
desviación Iris-virginica
0   0.635880
1   0.322497
2   0.551895
3   0.274650
Name: std, dtype: float64

```

Figura 22. Media y desviación de la clase Iris-virginica

2. Cree una gráfica de dispersión de la longitud del sépalo y ancho del pétalo mostrando en la gráfica las tres especies con distintos colores.

Para este ejercicio se obtuvo la gráfica de dispersión entre la longitud del sépalo y el ancho del pétalo para las 3 especies de flores iris. Lo primero que se realizó en este ejercicio fue obtener las características longitud del sépalo y ancho de pétalo para todo el conjunto de datos.

Una vez creados los conjuntos de datos para ambas características se realizaron las gráficas de dispersión para cada clase de iris utilizando la función *scatter* de Matplotlib. Al analizar la gráfica obtenida podemos observar que existe una relación más o menos lineal entre ambas variables, para todas las clases de iris, al igual que podemos observar que la longitud del sépalo y el ancho de pétalo de la clase Iris-Virginica son las más grandes de todas las clases mientras que las de la Iris-Setosa son las más pequeñas.

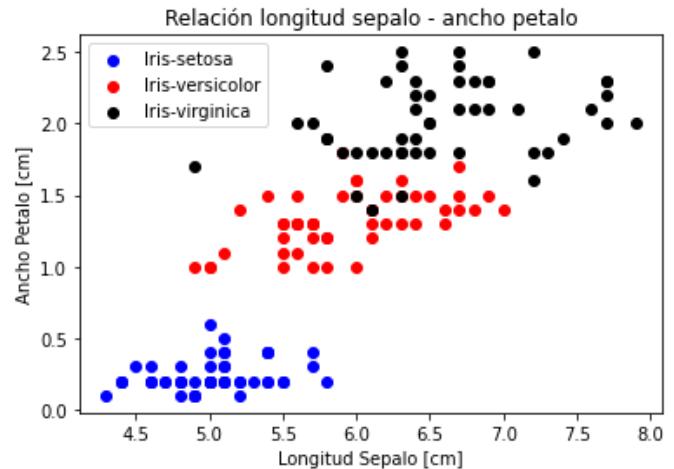


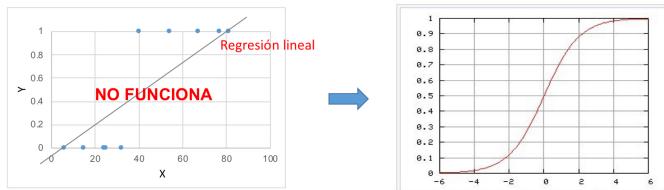
Figura 23. Gráfica de la relación entre la longitud del sépalo y el ancho del pétalo

3. En el modelado estadístico, el análisis de regresión es un proceso para estimar la relación entre variables. Investigue y describa la regresión logística.

RECONOCIMIENTO DE PATRONES.

La regresión Logística es un tipo de algoritmo de aprendizaje supervisado cuyo objetivo es predecir valores binarios (0 o 1). Este algoritmo consiste en una transformación a la regresión lineal.

La transformación se debe a que una regresión lineal no funciona para predecir una variable binaria. Se utiliza la misma estructura que la regresión lineal, pero se transforma la variable respuesta (0 o 1) en una probabilidad. Para esta transformación se utiliza la función logística conocida también como sigmoide.



Función Logística

Esta función convierte la entrada en un valor dentro de un rango de 0 a 1.

$$\text{Función logística} = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-(a+bX)}}$$

Funcionamiento

Paso 1

Se transforma Y en el logaritmo de la probabilidad de Y, esto es:

$$\ln\left(\frac{p}{1-p}\right)$$

$$Y = a + b_i X_i + u$$

A esta transformación también se conoce como razón de probabilidad de ser verdadero (Odds Ratio).

$$\text{Probabilidad} = \ln\left(\frac{p}{1-p}\right) = a + b_i X_i$$

Paso 2

Se calcula la regresión lineal para predecir el logaritmo:

$$\ln\left(\frac{P}{1-P}\right) = a + b_i X_i$$

Paso 3

Se transforma el resultado de la regresión lineal en la probabilidad final.

$$\frac{1}{1+e^{-x}} = \frac{1}{1+e^{-(a+b_i X_i)}}$$

- Si la probabilidad es superior a 0.5 se asigna 1.
- Si es menor a 0.5 se asigna 0.

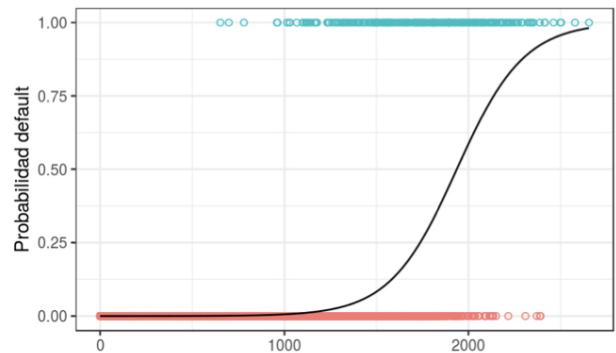


Figura 24. Gráfica de regresión lineal, con probabilidad final

4. Clasifique los datos mediante regresión logística y mida el desempeño de su modelo. Describa la medida usada para medir el desempeño de su modelo.

Como se mencionó en la introducción, utilizamos un modelo de regresión logística multinomial o multiclas, ya que contamos con tres clases distintas.

La clase LogisticRegression de sklearn puede ser configurada como regresión multiclas. En el caso de multiclas, el algoritmo de entrenamiento usa un algoritmo uno-contra-todos si se escoge la opción 'ovr' en el parámetro `multi_class`. En caso de que se escoga '`multi_class`', se utilizará un método de cross-entropy loss.[11]

Por defecto, el parámetro tiene el valor '`auto`', que selecciona '`ovr`' si la información es binaria, y '`multinomial`' si la información no es binaria.

También se implementa una regresión logística regularizada con la biblioteca liblinear. Esta biblioteca permite que se utilicen diferentes algoritmos de optimización para problemas varios. Esto se especifica en el parámetro `solver`, donde podemos seleccionar el algoritmo. La documentación muestra recomendaciones de en qué casos es mejor utilizar cierto algoritmo. Por default, se utiliza '`lbfgs`' para el caso multinomial. [12]

Se utiliza una primera forma de evaluación con el método `score`, el cual regresa el promedio de la exactitud del modelo bajo un conjunto de datos de prueba.

RECONOCIMIENTO DE PATRONES.

Como segundo método de evaluación, realizamos una matriz de confusión para poder visualizar los aciertos y errores.

Una matriz de clasificación, conocida también como matriz de confusión, se utiliza para evaluar una clasificación binaria.

En la variable clase, el conjunto de entrenamiento toma dos valores posibles: 0 o 1; positivo o negativo; falso o verdadero.

- Los valores positivos y negativos que se predicen correctamente se conocen como verdaderos positivos (VP) y verdaderos negativos (VN), respectivamente.
- Los valores clasificados incorrectamente se denominan falsos positivos (FP) y falsos negativos (FN).

		Predicción	
		Positivos	Negativos
Observación	Positivos	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 25. Matriz de clasificación

IV. CONCLUSIONES

Es un poco complicado entrar de lleno a algo tan complejo como las redes neuronales. En este caso, se implementa un modelo casi estadístico sin una real implementación (o desconocida por nuestra parte) de una red neuronal.

Anteriormente, alguno de nosotros había tomado un curso introductorio al deep learning en Coursera, en el cual se implementó una clasificación logística binaria con una red sencilla de solo 1 capa oculta. En este caso, no encontramos más información de cómo se implementan las redes en la clase LogisticRegression de sklearn.

Aparte de aprender acerca de la regresión logística y sus variantes (binaria y multiclase). También nos dimos cuenta de ciertas cosas.

A) Selección de características

Como vimos durante la clase, la selección de características para realizar nuestros modelos es muy importante, puede que realizando una selección de características bueno, reduzcamos drásticamente el procesamiento y la carga computacional de entrenar y usar nuestro modelo. A su vez, una buena selección también nos permite obtener mejores resultados de predicciones.

Durante la práctica observamos esto con la prueba de 3 modelos distintos, utilizando diferentes características del dataset en cada uno.

Primero utilizamos las 4 características del dataset. Si bien el modelo tiene un desempeño bueno (93%), durante el entrenamiento observamos que el modelo no converge con el número estándar de iteraciones, por lo cual necesitamos modificar el parámetro *max_iter* del modelo para que converja. Aquí podemos observar como el uso de muchas características nos genera un modelo pesado.

Para el segundo, utilizamos las características del largo de pétalo y ancho de pétalo. No es sorpresa que tuviéramos un buen desempeño, ya que incluso podemos ver en las gráficas anteriores que esta selección de features nos otorga una buena separación de clases, en especial entre Iris-versicolor e Iris-virginica.

Para el tercero, utilizamos el Largo del sépalo y el ancho del pétalo. Tomamos esta decisión al observar esta combinación de características en la gráfica obtenida por *pairplot*. En esta gráfica podemos observar lo difícil que es escoger un conjunto de características que separe de manera correcta las clases Iris-versicolor e Iris-virginica. Este modelo tuvo un desempeño más bajo.

En la tabla 1 podemos observar las características seleccionadas para cada modelo (largo de pétalo (LP), ancho de pétalo (AP), largo de sépalo (AP) y ancho de sépalo (AS)) y su desempeño (accuracy/precisión (A%)) en porcentaje.

Modelo	LP	AP	LS	AS	A%
1	1	1	1	1	93
2	1	1	0	0	93
3	0	1	1	0	83

Tabla 1. Comparación de modelos

B) Selección de datasets

Algo que también observamos es que cambiar el conjunto de entrenamiento y de prueba hace mucha diferencia. Durante la práctica utilizamos una función que lo divide de manera aleatoria, por lo cual, siempre que corremos el programa, obtenemos conjuntos distintos.

Un problema que encontramos es que, la mayoría de las veces, estos conjuntos no son representativos, ya que no contienen el mismo número de ejemplos para cada clase. Esto hace que el entrenamiento y la evaluación arrojen valores distintos cada vez que ejecutamos el programa.

RECONOCIMIENTO DE PATRONES.

A su vez, el que el dataset principal contenga pocos datos, aumenta este problema, ya que como sabemos, los algoritmos de machine learning ocupan muchos datos, y no podrán converger a una forma satisfactoria si no cuentan con los suficientes datos para su entrenamiento.

REFERENCIAS

- [1] colaboradores de Wikipedia. (2021, 14 julio). “Conjunto de datos flor iris.” Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Conjunto_de_datos_flor_iris (accessed Sep. 24, 2021).
- [2] “El conjunto de datos de Iris: un poco de historia y biología.” (2020, 27 noviembre). ICHI.PRO. <https://ichi.pro/es/el-conjunto-de-datos-de-iris-un-poco-de-historia-y-biologia-276286974306229> (accessed Sep. 24, 2021).
- [3] Rodrigo, J. A. (s. f.). “Regresión logística simple y múltiple. Ciencia de Datos.” https://www.cienciadedatos.net/documentos/27_Regresion_logistica_simple_y_multiple#Regresi%C3%B3n_log%C3%ADstica_m%C3%A9tricas (accessed Sep. 24, 2021).
- [4] “Acerca de la regresión lineal.” (s. f.). México | IBM. <https://www.ibm.com/mx-es/analytics/learn/linear-regression> (accessed Sep. 24, 2021).
- [5] Arce, J. I. B. (2021, 15 agosto). “La matriz de confusión y sus métricas.” Juan Barrios. <https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/> (accessed Sep. 24, 2021).
- [6] Alberca, A. S. (2021, 14 mayo). “La librería Pandas.” Aprende con Alf. <https://aprendeconalf.es/docencia/python/manual/pandas/> (accessed Sep. 24, 2021).
- [7] Alberca, A. S. (2020, 4 octubre). “La librería Matplotlib.” Aprende con Alf. <https://aprendeconalf.es/docencia/python/manual/matplotlib/> (accessed Sep. 24, 2021).
- [8] Rodríguez, D. (2018, 1 diciembre). “Visualización de datos en Python con Seaborn.” Analytics Lane. <https://www.analyticslane.com/2018/07/20/visualizacion-de-datos-con-seaborn/> (accessed Sep. 24, 2021).
- [9] “Scikit-Learn, herramienta básica para el Data Science en Python.” (2019, 8 abril). Máster en Data Science. <https://www.master-data-scientist.com/scikit-learn-data-science/> (accessed Sep. 24, 2021).
- [10] Rodríguez, D. (2020, 18 enero). “Matrices dispersas (“Sparse Matrix”).” Analytics Lane. <https://www.analyticslane.com/2019/10/21/matrices-dispersas-sparse-matrix/> (accessed Sep. 24, 2021).
- [11] Brownlee, J. (2020, 1 septiembre). “Multinomial Logistic Regression With Python.” Machine Learning Mastery. <https://machinelearningmastery.com/multinomial-logistic-regression-with-python/> (accessed Sep. 24, 2021).
- [12] “sklearn.linear_model.LogisticRegression.” (s. f.). Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (accessed Sep. 24, 2021).

```
# obtenemos el archivo de datos desde internet
!wget -q https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
```

```
# bibliotecas necesarias
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

▼ A) Ejercicios Básicos

1. Cargue los datos iris en un data frame (pandas) e imprima la descripción de los datos (columnas y renglones), tipo y las 10 primeras filas de los datos. Fuente de datos: <https://archive.ics.uci.edu/ml/datasets/Iris>.

```
# definimos la ruta para cargar los datos
rutaDatos = "/content/iris.data"
# definimos los headers, el archivo no los tiene
headers = ["sepal length","sepal width","petal length","petal width","class"]
dictRename = {i:headers[i] for i in range(len(headers))}

# cargamos la información en un dataframe
df = pd.read_csv(rutaDatos,header=None)

# cambiamos los headers
df.rename(columns=dictRename,inplace=True)

# Forma (shape) de los datos
print("Forma de los datos")
print("Número de datos:",df.size)
row,col = df.shape
print("Columnas:",col)
print("Filas:",row)

# tipo de datos por columna
print("\nTipo de datos por columna")
for i in df.keys():
    print(i,":",df[i].dtype)

# primeras 10 columnas
print("\nPrimeras 10 filas")
print(df[:10])
```

```
Forma de los datos
Número de datos: 750
Columnas: 5
Filas: 150

Tipo de datos por columna
sepal length : float64
sepal width : float64
petal length : float64
petal width : float64
class : object

Primeras 10 filas
   sepal length  sepal width  petal length  petal width      class
0          5.1         3.5        1.4       0.2  Iris-setosa
1          4.9         3.0        1.4       0.2  Iris-setosa
2          4.7         3.2        1.3       0.2  Iris-setosa
3          4.6         3.1        1.5       0.2  Iris-setosa
4          5.0         3.6        1.4       0.2  Iris-setosa
5          5.4         3.9        1.7       0.4  Iris-setosa
6          4.6         3.4        1.4       0.3  Iris-setosa
7          5.0         3.4        1.5       0.2  Iris-setosa
8          4.4         2.9        1.4       0.2  Iris-setosa
9          4.9         3.1        1.5       0.1  Iris-setosa
```

2. Imprima las llaves y el número de filas y de columnas.

```
# llaves
print("Llaves de la tabla")
print(list(df.keys()))
# Número de filas y columnas
print("\nNúmero de filas y columnas")
row,col = df.shape
print("{} filas, {} columnas".format(row,col))

Llaves de la tabla
['sepal length', 'sepal width', 'petal length', 'petal width', 'class']

Número de filas y columnas
150 filas, 5 columnas
```

3. Obtenga el número de muestras faltantes o Nan.

```
# número si alguno es nulo por columna
#print(df.isnull().any())
# número de nulos por columnas
```

```
#print(df.isnull().sum())
# número de nulos totales
print("Número de NaN:",df.isnull().sum().sum())

# indices de los elementos nulos
#print(df.columns[df.isnull().any()])
# http://exponentis.es/como-encontrar-valores-nan-en-un-dataframe-python-pandas-y-modificarlos
```

Número de NaN: 0

4. Cree un arreglo 2-D de tamaño 5x5 con unos en la diagonal y ceros en el resto. Convierta el arreglo NumPy a una matriz dispersa de ScyPy en formato CRS. Nota: una matriz se considera dispersa cuando el porcentaje de ceros es mayor a 0.5.

```
# 4
# calcular porcentaje de ceros
mat = np.identity(5)
print("Matriz de 5x5 con 1's en la diagonal (matriz identidad 5)")
print(mat)

print("\nPorcentajes de ceros:", (mat == 0).sum()/mat.size)

s = csr_matrix(mat)
print("\nMatriz dispersa")
print(s)
# https://www.analyticslane.com/2019/10/21/matrices-dispersas-sparse-matrix/
```

Convertir un matriz dispersa en densa
#D = s.todense()

Matriz de 5x5 con 1's en la diagonal (matriz identidad 5)

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

Porcentajes de ceros: 0.8

Matriz dispersa

(0, 0)	1.0
(1, 1)	1.0
(2, 2)	1.0
(3, 3)	1.0
(4, 4)	1.0

5. Muestre estadísticas básicas como percentil, media, mínimo, máximo y desviación estándar de los datos. Use describe para ello. Imprima sólo la media y la desviación estándar.

```
# 5
print("Estadísticas básicas de los datos")
stats = df.describe()
print(stats)
# media y desviación
print("\nMedia y desviación")
# hay que especificar explicitamente los indices
print(stats.iloc[[1,2],[0,1,2,3]])
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html
```

Estadísticas básicas de los datos

	sepal length	sepal width	petal length	petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Media y desviación

	sepal length	sepal width	petal length	petal width
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161

6. Obtenga el número de muestras para cada clase.

```
#https://www.delftstack.com/es/howto/python-pandas/split-pandas-dataframe/
print("Número de muestras por clase")
```

se agrupan los datos de acuerdo a la última columna

```
grupos = df.groupby(df["class"])
tipos = set(df["class"])
for tipo in tipos:
    print(tipo,"=", (df["class"]==tipo).sum())
```

Número de muestras por clase
Iris-setosa = 50
Iris-virginica = 50
Iris-versicolor = 50

7. Añada un encabezado a los datos usando los nombres en iris.names y repita el ejercicio anterior.

```

print("Cambiando headers")
headers = ['LargoSepalo','AnchoSepalo','LargoPetalo','AnchoPetalo','Clase']
df.columns=headers
print(df.head())

print("\nNúmero de muestras por Clase")
# se agrupan los datos de acuerdo a la última columna
grupos = df.groupby(df["Clase"])
tipos = set(df["Clase"])
for tipo in tipos:
    print(tipo,"=", (df["Clase"]==tipo).sum())

```

```

Cambiando headers
   LargoSepalo  AnchoSepalo  LargoPetalo  AnchoPetalo      Clase
0           5.1          3.5          1.4          0.2  Iris-setosa
1           4.9          3.0          1.4          0.2  Iris-setosa
2           4.7          3.2          1.3          0.2  Iris-setosa
3           4.6          3.1          1.5          0.2  Iris-setosa
4           5.0          3.6          1.4          0.2  Iris-setosa

```

```

Número de muestras por Clase
Iris-versicolor = 50
Iris-setosa = 50
Iris-virginica = 50

```

8. Imprima las diez primeras filas y las dos primeras columnas del data frame usando los índices de las columnas.

```

#https://datacarpentry.org/python-ecology-lesson-es/03-index-slice-subset/
print("Primeras 10 filas y primeras dos columnas de los datos")
print(df.iloc[0:10, 0:2])

```

```

Primeras 10 filas y primeras dos columnas de los datos
   SepalLength  SepalWidth
0           5.1          3.5
1           4.9          3.0
2           4.7          3.2
3           4.6          3.1
4           5.0          3.6
5           5.4          3.9
6           4.6          3.4
7           5.0          3.4
8           4.4          2.9
9           4.9          3.1

```

▼ B) Ejercicios de visualización

Utilizando matplotlib y/o seaborn 1. Cree una gráfica de barras que muestre la media, mínimo y máximo de todos los datos.

```

# obteniendo estadísticas de nuevo
stats = df.describe()
# seleccionando media, min y max
info = stats.iloc[[1,3,7],[0,1,2,3]]
print("Información")
print(info,"\n")

# definiendo la gráfica
# etiquetas
etq = ["media","min","max"]
# series de datos
sl = list(info["LargoSepalo"])
sw = list(info["AnchoSepalo"])
pl = list(info["LargoPetalo"])
pw = list(info["AnchoPetalo"])

# grosor de la gráfica
grosor = 1
# posiciones de x para ponerlos
x = np.arange(len(etq))*6
separacion = grosor/2
slg = plt.bar(x-3*separacion,sl,width=grosor,label="Sepalo largo",color="purple")
swg = plt.bar(x-separacion,sw,width=grosor,label="Sepalo ancho",color="green")
plg = plt.bar(x+separacion,pl,width=grosor,label="Petalo largo",color="red")
pwg = plt.bar(x+3*separacion,pw,width=grosor,label="Petalo ancho",color="blue")
plt.legend()
plt.title("Estadísticas básicas del dataset")

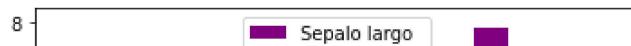
plt.xticks(x,labels=etq) #xticks para agregar nuevos ejes
plt.show()

```

Información

	LargoSepalo	AnchoSepalo	LargoPetaloo	AnchoPetaloo
mean	5.843333	3.054	3.758667	1.198667
min	4.300000	2.000	1.000000	0.100000
max	7.900000	4.400	6.900000	2.500000

Estadísticas básicas del dataset

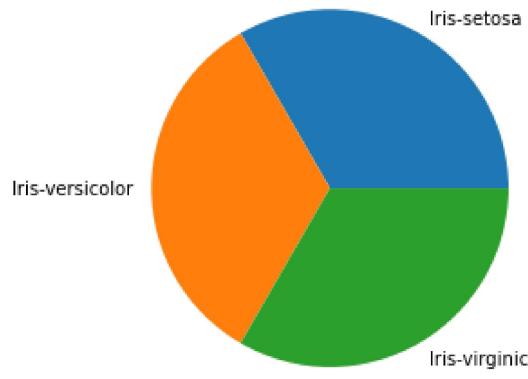


2. Muestre la frecuencia de las tres especies como una gráfica de pastel.

```
#grupos = df.groupby(df["class"])
# sacamos las etiquetas
etiquetas = list(set(df["Clase"]))
# sacamos las frecuencias de cada etiqueta
frecuencias = [(df["Clase"]==tipo).sum() for tipo in etiquetas]
print(etiquetas)
#print(frecuencias)
plt.pie(frecuencias,labels=etiquetas)
plt.axis("equal")
plt.title("Frecuencia de clases")
plt.show()
```

['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

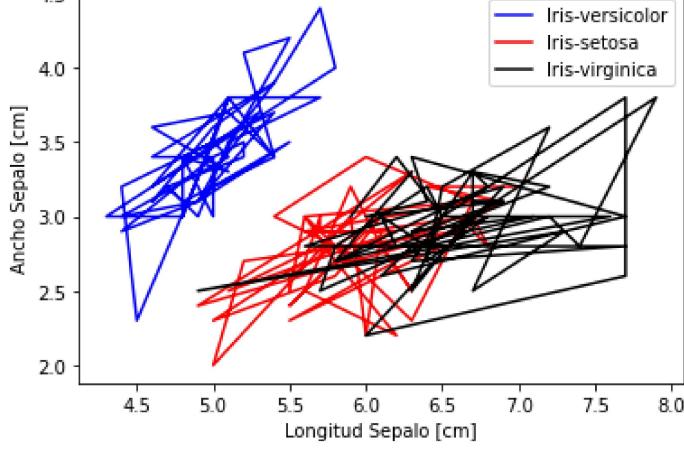
Frecuencia de clases



3. Cree una gráfica que muestre la relación entre la longitud y ancho del sépalo de las tres especies conjuntamente.

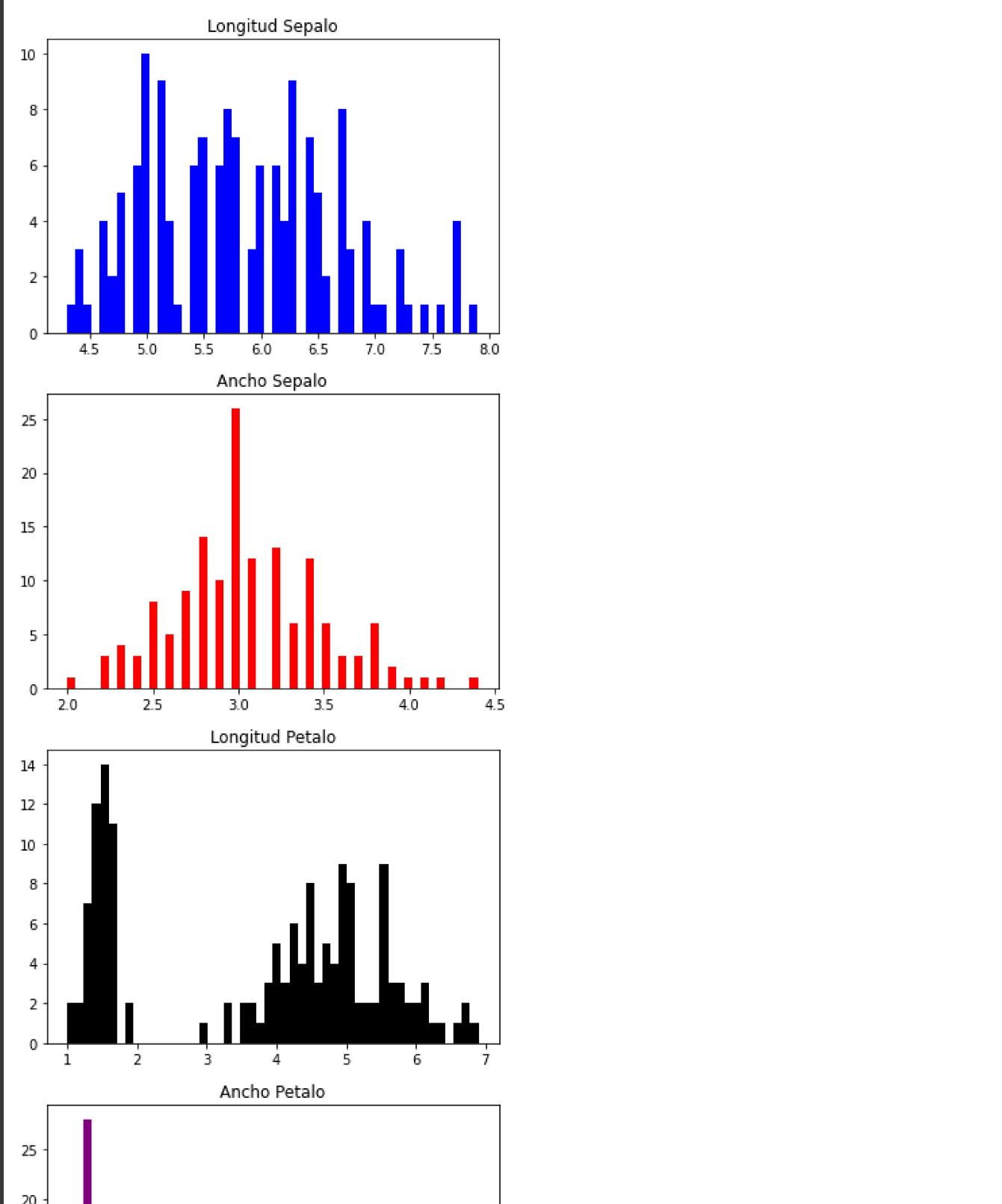
```
sepallength = list(df["LargoSepalo"])
sepalwidth = list(df["AnchoSepalo"])
plt.plot(sepallength[:50],sepalwidth[:50],label="Iris-versicolor",color="blue")
plt.plot(sepallength[50:100],sepalwidth[50:100],label="Iris-setosa",color="red")
plt.plot(sepallength[100:150],sepalwidth[100:150],label="Iris-virginica",color="black")
plt.legend()
plt.title("Relación longitud-ancho de sepalo")
plt.xlabel("Longitud Sepalo [cm]")
plt.ylabel("Ancho Sepalo [cm]")
plt.show()
```

Relación longitud-ancho de sepalo



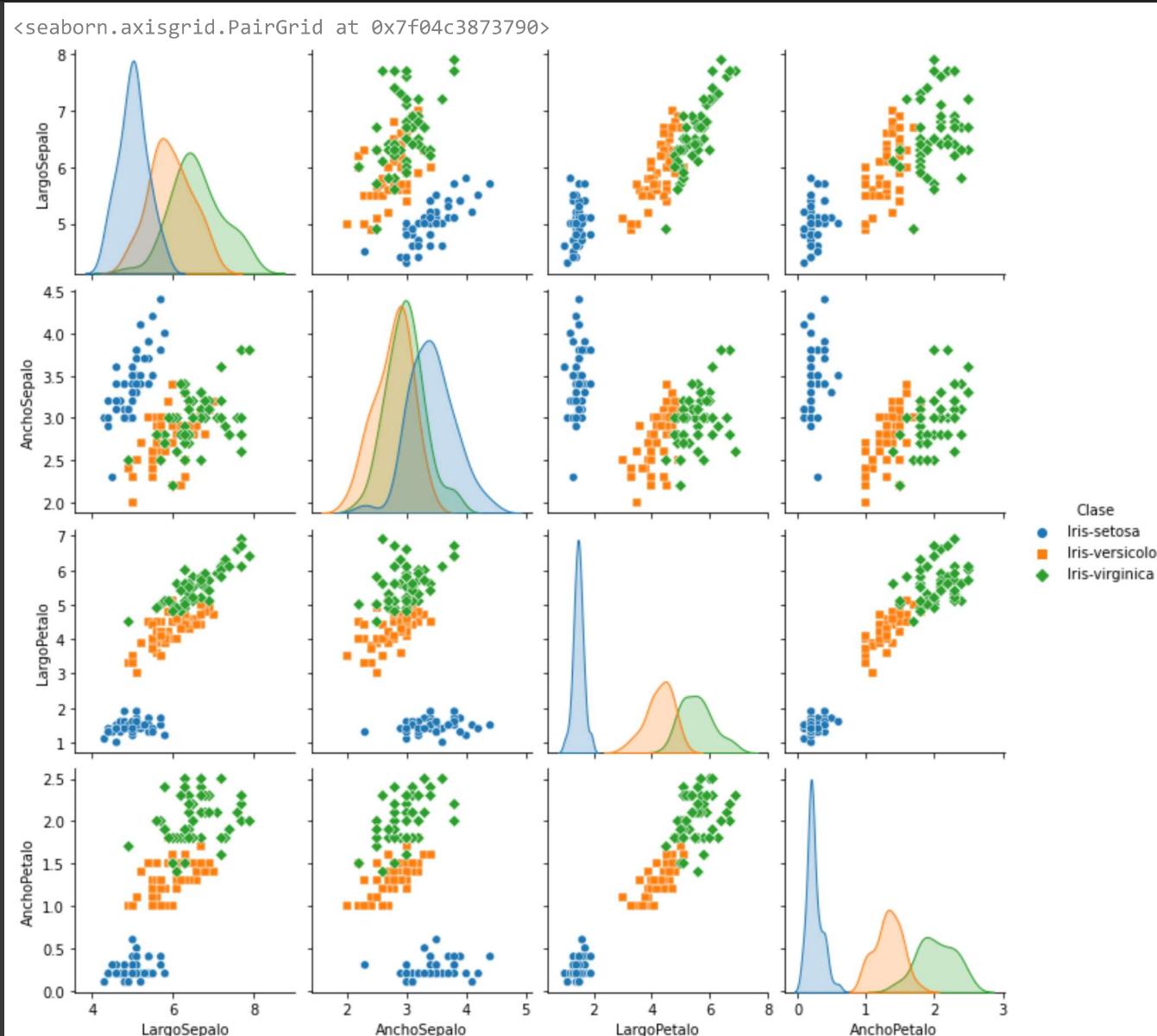
4. Obtenga los histogramas de las variables SepalLength, SepalWidth, PetalLength y PetalWidth.

```
plt.hist(df["LargoSepalo"],50,color="blue")
plt.title("Longitud Sepalo")
plt.show()
plt.hist(df["AnchoSepalo"],50,color="red")
plt.title("Ancho Sepalo")
plt.show()
plt.hist(df["LargoPetaloo"],50,color="black")
plt.title("Longitud Petalo")
plt.show()
plt.hist(df["AnchoPetaloo"],50,color="purple")
plt.title("Ancho Petalo")
plt.show()
```



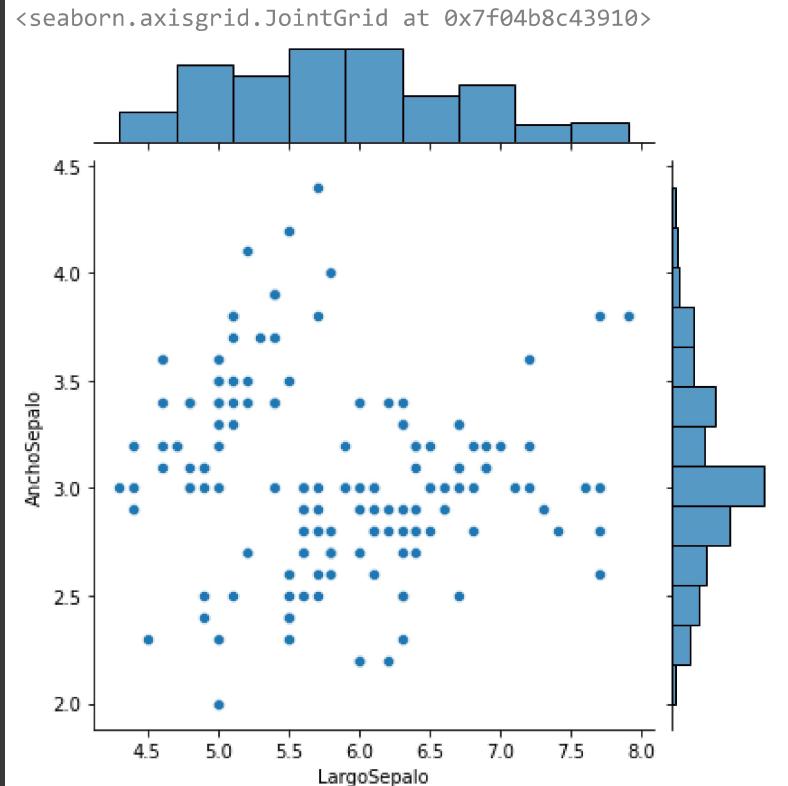
5. Cree gráficas de dispersión usando pairplot de seaborn y muestre con distintos colores las tres especies en las gráficas de dispersión.

```
sns.pairplot(df,hue="Clase", markers=[ "o", "s", "D"])
#https://seaborn.pydata.org/generated/seaborn.pairplot.html
```



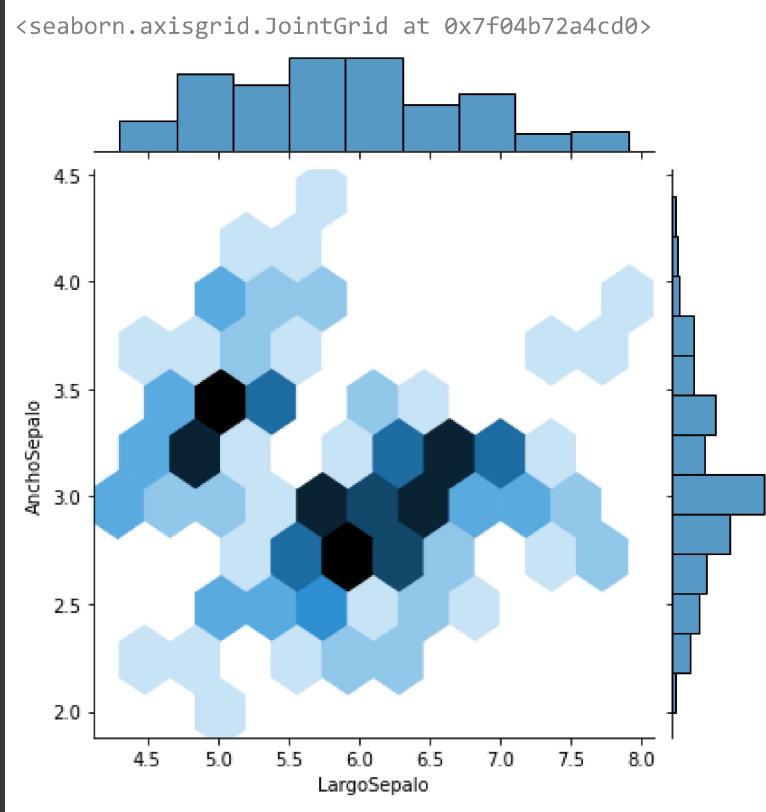
6. Cree una gráfica usando joinplot de seaborn para mostrar la dispersión entre la longitud y ancho del sépalo y las distribuciones de estas

```
sns.jointplot(data=df, x="LargoSepalo", y="AnchoSepalo")
https://seaborn.pydata.org/generated/seaborn.jointplot.html
```



7. Repita el ejercicio anterior, pero esta vez usando joinplot con kind="hexbin".

```
sns.jointplot(data=df, x="LargoSepalo", y="AnchoSepalo", kind="hex")
```



▼ C) Ejercicios de regresión logística

1. Muestre los percentiles, media y desviación estándar de cada especie ('Iris-setosa', 'Iris-versicolor' e 'Iris-virginica').

```
copia = df.copy()
copia.set_index("Clase", inplace = True)
ISe = copia.loc["Iris-setosa"]
IVe = copia.loc["Iris-versicolor"]
IVi = copia.loc["Iris-virginica"]

for especie in ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]:
    print("Especie:", especie)
    aux = copia.loc[especie]
    print(aux.describe().iloc[[4,5,6,1,2]], "\n")
```

Especie: Iris-setosa

	LargoSepalo	AnchoSepalo	LargoPetalo	AnchoPetalo
25%	4.80000	3.12500	1.40000	0.20000
50%	5.00000	3.40000	1.50000	0.20000
75%	5.20000	3.67500	1.57500	0.30000
mean	5.00600	3.41800	1.46400	0.24400
std	0.35249	0.381024	0.173511	0.10721

Especie: Iris-versicolor

	LargoSepalo	AnchoSepalo	LargoPetalo	AnchoPetalo
25%	5.60000	2.52500	4.00000	1.20000
50%	5.90000	2.80000	4.35000	1.30000
75%	6.30000	3.00000	4.60000	1.50000
mean	5.93600	2.77000	4.26000	1.32600
std	0.516171	0.313798	0.469911	0.197753

Especie: Iris-virginica

	LargoSepalo	AnchoSepalo	LargoPetalo	AnchoPetalo
25%	6.22500	2.80000	5.10000	1.80000

```

50%      6.50000  3.000000  5.550000  2.00000
75%      6.90000  3.175000  5.875000  2.30000
mean     6.58800  2.974000  5.552000  2.02600
std      0.63588  0.322497  0.551895  0.27465

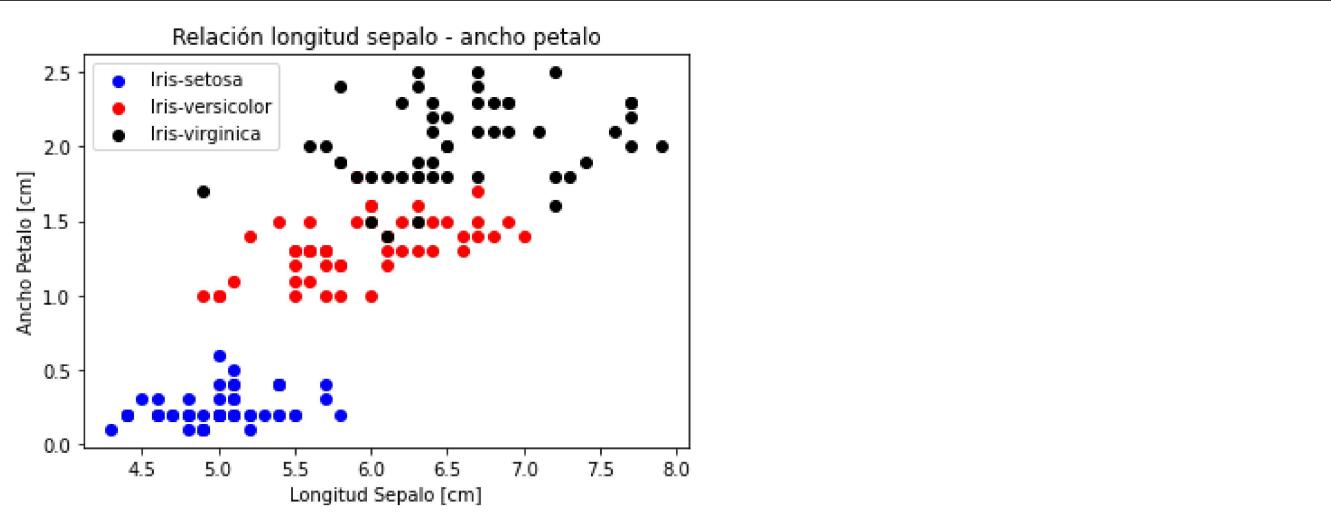
```

2. Cree una gráfica de dispersión de la longitud del sépalo y ancho del pétalo mostrando en la gráfica las tres especies con distintos colores.

```

# Encontrar los datos necesarios
sepalLength = list(df["LargoSepalo"])
petalWidth = list(df["AnchoPetalo"])
# creando las graficas
plt.scatter(sepalLength[:50],petalWidth[:50],label="Iris-setosa",color="blue")
plt.scatter(sepalLength[50:100],petalWidth[50:100],label="Iris-versicolor",color="red")
plt.scatter(sepalLength[100:150],petalWidth[100:150],label="Iris-virginica",color="black")
# para que se vea bonito
plt.legend()
plt.title("Relación longitud sepalo - ancho petalo")
plt.xlabel("Longitud Sepalo [cm]")
plt.ylabel("Ancho Petalo [cm]")
plt.show()

```



3. En el modelado estadístico, el análisis de regresión es un proceso para estimar la relación entre variables. Investigue y describa la regresión logística.

```
#https://realpython.com/logistic-regression-python/#classification
```

4. Clasifique los datos mediante regresión logística y mida el desempeño de su modelo. Describa la medida usada para medir el desempeño de su modelo.

```

# importamos las herramientas necesarias para crear los modelos
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# inicializamos los 3 modelos
modeloTodos = LogisticRegression(max_iter=1000)
modeloLPAP = LogisticRegression()
modeloLSAP = LogisticRegression()

# Definición de los datasets
X = df[["LargoSepalo","AnchoSepalo","LargoPetalo","AnchoPetalo"]]
# definimos las clases para cada conjunto de "features"
Y = df[["Clase"]]
# dividimos el dataset en train y test set
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.2)
# es necesario redimensionar los sets de Y ya que la función anterior
# regresa solo una lista, es necesario que sea un arreglo de tamaño
# (n_samples,). Usamos np.ravel por que no se puede redimensionar
# un Dataframe con reshape() o alterando el atributo shape
y_train = np.ravel(y_train)
y_test = np.ravel(y_test)

# Entrenamiento del modelo *****
# Aquí escogemos que características usaremos
modeloTodos.fit(x_train[["LargoSepalo","AnchoSepalo","LargoPetalo","AnchoPetalo"]],
                 y_train)
# Evaluación del modelo *****
y_predecida_todos = modeloTodos.predict(x_test)
y_test_todos = y_test
cal = modeloTodos.score(x_test,y_test) # buscar que hace esta función
print("Precisión del modelo: {}%".format(cal*100))

Precisión del modelo: 93.33333333333333%
```

```

# Entrenamiento del modelo *****
# Aquí escogemos que características usaremos
modeloLPAP.fit(x_train[["LargoPetalo","AnchoPetalo"]],y_train)
# Evaluación del modelo *****
y_predecida_LPAP = modeloLPAP.predict(x_test[["LargoPetalo","AnchoPetalo"]])
y_test_LPAP = y_test
cal = modeloLPAP.score(x_test[["LargoPetalo","AnchoPetalo"]],
                       y_test) # buscar que hace esta función
print("Precisión del modelo: {}%".format(cal*100))

```

Precisión del modelo: 93.333333333333%

```
# Entrenamiento del modelo ****
# Aquí escogemos que características usaremos
modeloLSAP.fit(x_train[["LargoSepalo","AnchoPetalo"]],y_train)
# Evaluación del modelo ****
y_predecida_LSAP = modeloLSAP.predict(x_test[["LargoSepalo","AnchoPetalo"]])
y_test_LSAP = y_test
cal = modeloLSAP.score(x_test[["LargoSepalo","AnchoPetalo"]],y_test) # buscar que hace esta función
print("Precisión del modelo: {}%".format(cal*100))
```

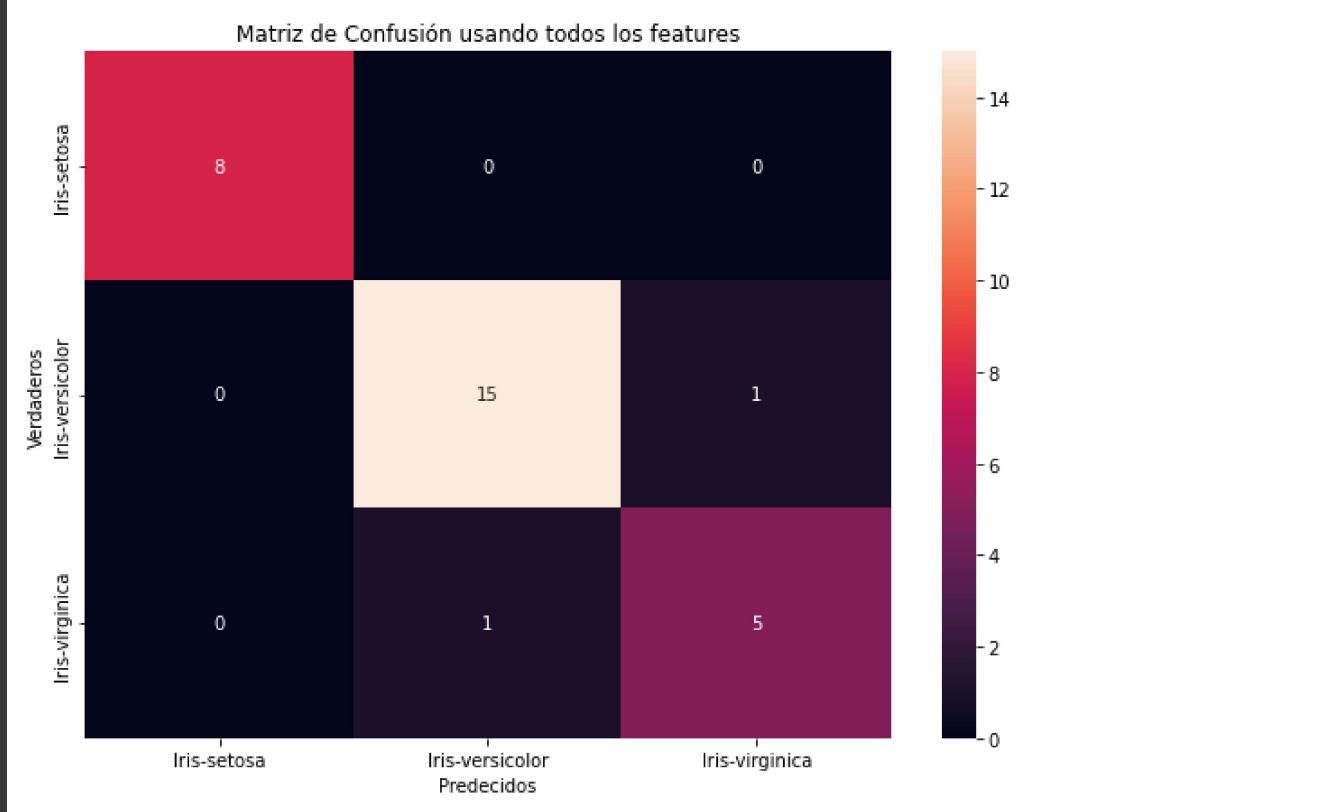
Precisión del modelo: 83.333333333334%

Matrices de confusión

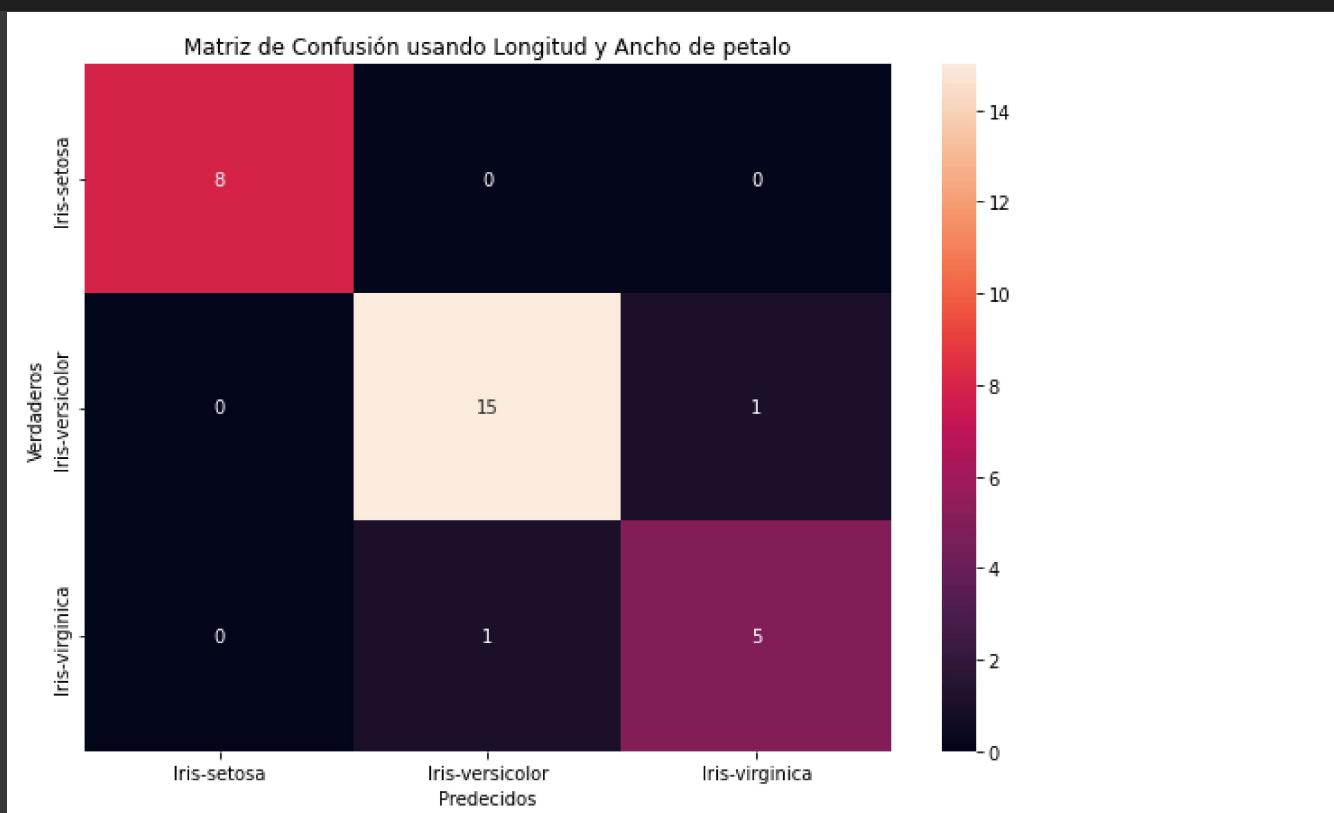
```
from sklearn.metrics import confusion_matrix
```

```
# determinamos las matrices de confusión
cm_Todos = confusion_matrix(y_test_todos,y_predecida_todos)
cm_LPAP = confusion_matrix(y_test_LPAP,y_predecida_LPAP)
cm_LSAP = confusion_matrix(y_test_LSAP,y_predecida_LSAP)
```

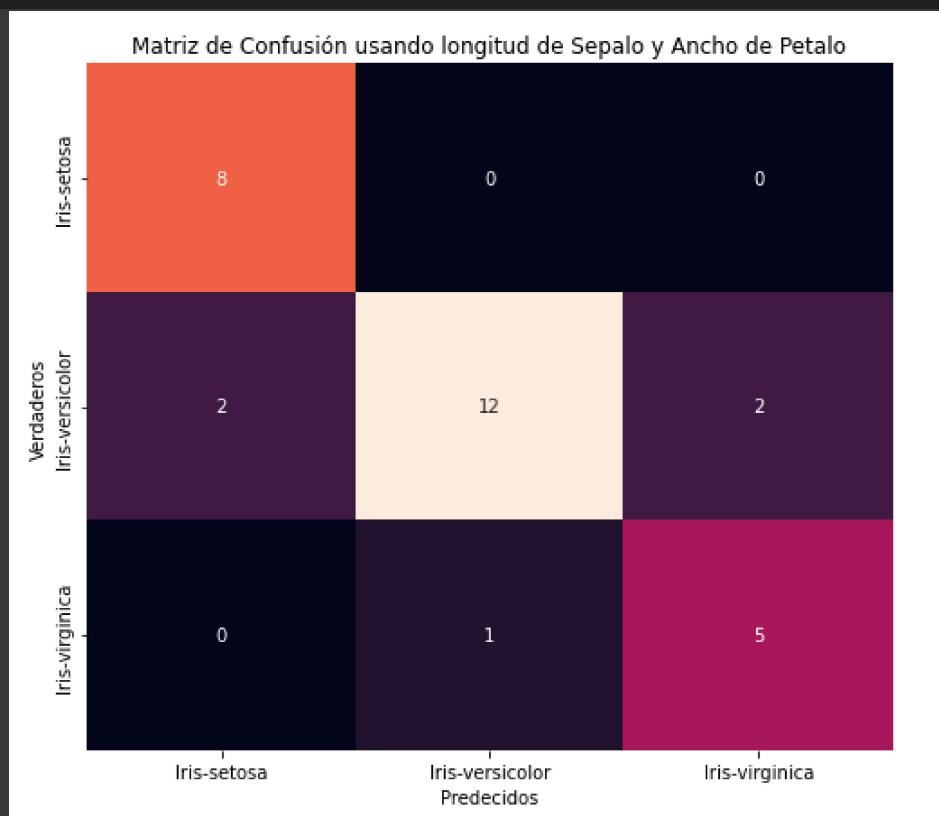
```
# Graficamos con el mapa de calor de seaborn
plt.figure(figsize=(10,7))
sns.heatmap(cm_Todos,annot=True)
# Edición para que sea más comprensible el gráfico
plt.xlabel("Predecidos")
plt.ylabel("Verdaderos")
plt.title("Matriz de Confusión usando todos los features")
# Sirve para cambiarle las anotaciones a los ejes
plt.xticks([0.5,1.5,2.5],labels=["Iris-setosa","Iris-versicolor","Iris-virginica"])
plt.yticks([0.5,1.5,2.5],labels=["Iris-setosa","Iris-versicolor","Iris-virginica"])
plt.show()
```



```
plt.figure(figsize=(10,7))
sns.heatmap(cm_LPAP,annot=True)
plt.xlabel("Predecidos")
plt.ylabel("Verdaderos")
plt.title("Matriz de Confusión usando Longitud y Ancho de petalo")
plt.xticks([0.5,1.5,2.5],labels=["Iris-setosa","Iris-versicolor","Iris-virginica"])
plt.yticks([0.5,1.5,2.5],labels=["Iris-setosa","Iris-versicolor","Iris-virginica"])
plt.show()
```



```
plt.figure(figsize=(10,7))
sns.heatmap(cm_LSAP, annot=True)
plt.xlabel("Predecidos")
plt.ylabel("Verdaderos")
plt.title("Matriz de Confusión usando longitud de Sepalo y Ancho de Petalo")
plt.xticks([0.5,1.5,2.5],labels=["Iris-setosa","Iris-versicolor","Iris-virginica"])
plt.yticks([0.5,1.5,2.5],labels=["Iris-setosa","Iris-versicolor","Iris-virginica"])
plt.show()
```



▶ Pruebas

↳ 24 cells hidden