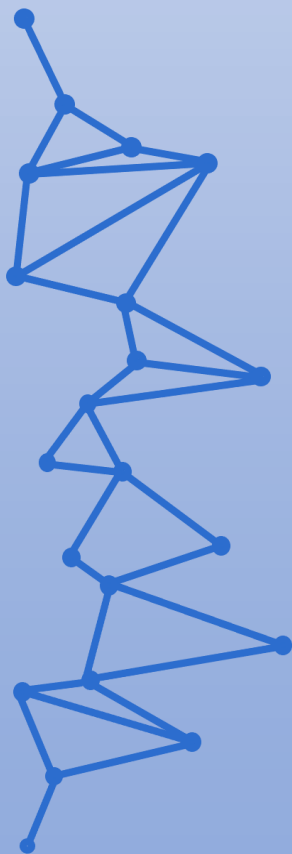




Curso de Especialización de Inteligencia Artificial y Big Data (IABD)



Programación de Inteligencia Artificial

UD04. Utilización de librerías de Ciencia de
Datos con Python.

Tarea Online.

JUAN ANTONIO GARCÍA MUELAS

INDICE

	Pag
1. Apartado 1: Explora los datos con Pandas	2
2. Apartado 2: Visualiza los datos con Pyplot	5
3. Apartado 3: Entrena modelos de aprendizaje automático con Scikit-Learn	6
4. Apartado 4: Entrena modelos de aprendizaje automático con pocas variables	10

Tarea para PIA04.

Título de la tarea: Utilización de librerías de Ciencia de Datos con Python

Curso de Especialización y módulo: Inteligencia Artificial y Big Data - Programación en Inteligencia Artificial.

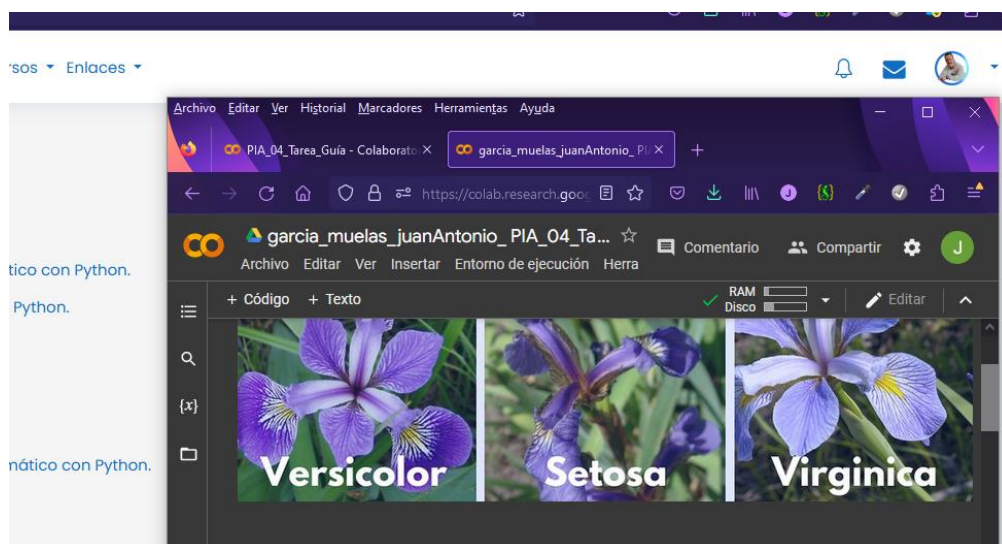
Curso académico: 2022-2023

¿Qué te pedimos que hagas?

✓ Apartado 1: Explora los datos con Pandas

- Inicia un nuevo notebook, preferiblemente en Google Colab. Para guiarte en el proceso, puedes utilizar este cuaderno-guía con los fragmentos de código indicados en las celdas de texto, pero tendrás que escribir el código en la celda de código correspondiente y ejecutarlo.

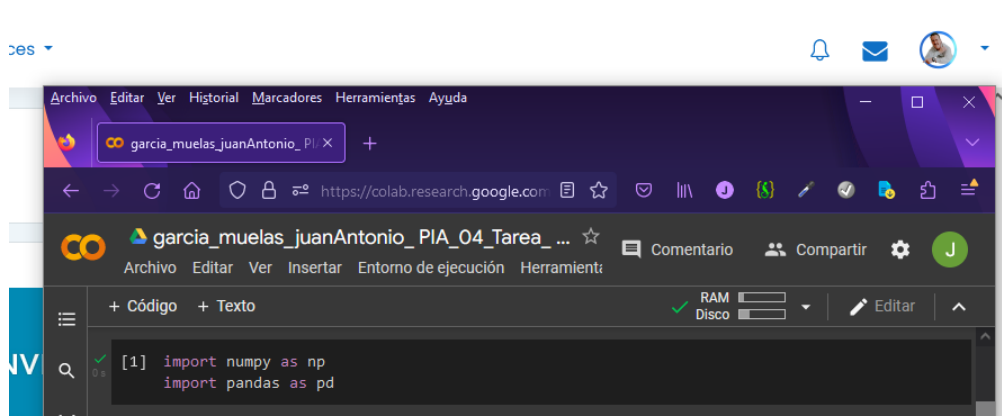
Creo una copia de la guía facilitada en mi drive.



- Importa las librerías **Numpy** y **Pandas**.

Importo las librerías y ejecuto.

```
import numpy as np
import pandas as pd
```



- Importa la función `load_iris` de la biblioteca de datasets de **Scikit-learn** y echa un vistazo rápido a sus principales elementos `'data'`, `'target'` y `'target_names'`.

Importamos la función y comprobamos los elementos a través de los arrays volcados.

```
from sklearn.datasets import load_iris
```

```
data = load_iris()
data['data']
```

The screenshot displays a Google Colab notebook in a web browser. The browser's address bar shows the URL 'https://colab.research.google.com'. The notebook's title bar indicates the file name 'garcia_muelas_juanAntonio_PIA_04_Tarea...'. The notebook interface includes a menu bar with options like 'Archivo', 'Editar', 'Ver', 'Insertar', 'Entorno de ejecución', and 'Herramientas'. The main workspace shows a code cell with the following Python code:

```
[2]: from sklearn.datasets import load_iris

data = load_iris()
data['data']
```

The output of the code cell is visible at the bottom, showing the first few rows of the Iris dataset:

```
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5. , 2.8, 4.1, 1.3]
```

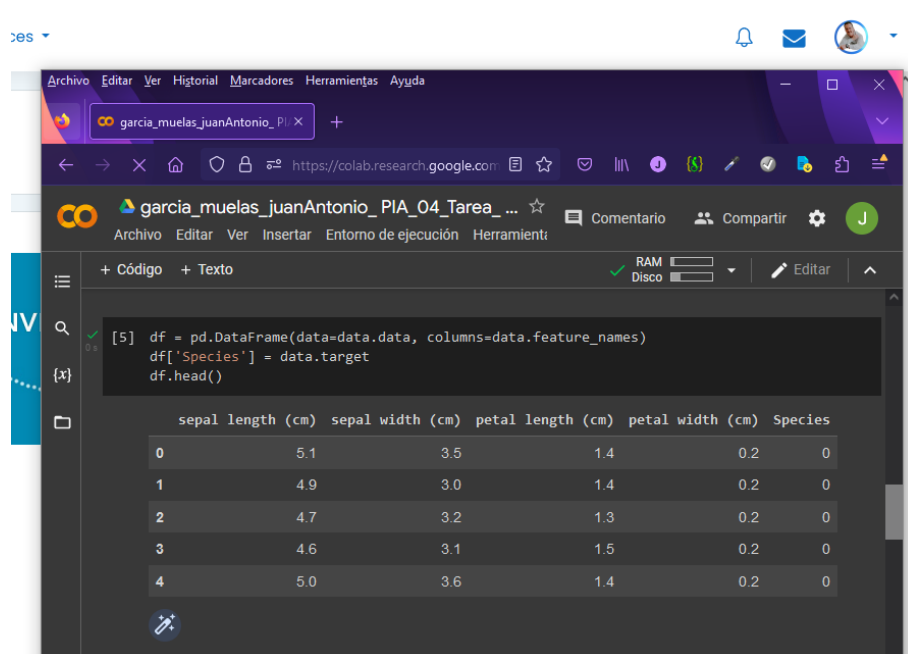
```
data['target']
```

```
data['target_names']
```

[illegible]

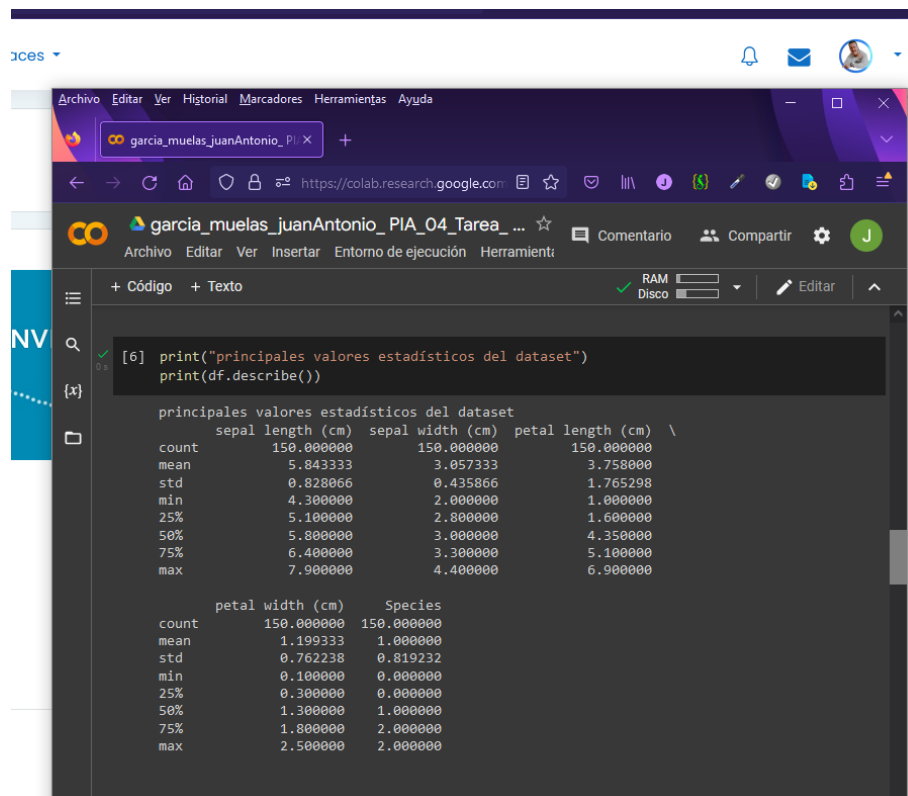
- Utiliza la clase *DataFrame* de Pandas para crear el dataset *df* y añade la columna "Species" a partir de la secuencia de datos *target*. Utiliza la función *head* para ver los primeros registros del nuevo dataset.

```
df = pd.DataFrame(data=data.data, columns=data.feature_names)
df['Species'] = data.target
df.head()
```



- Utiliza la función *describe* para ver los principales valores estadísticos del dataset.

```
print("principales valores estadísticos del dataset")
print(df.describe())
```

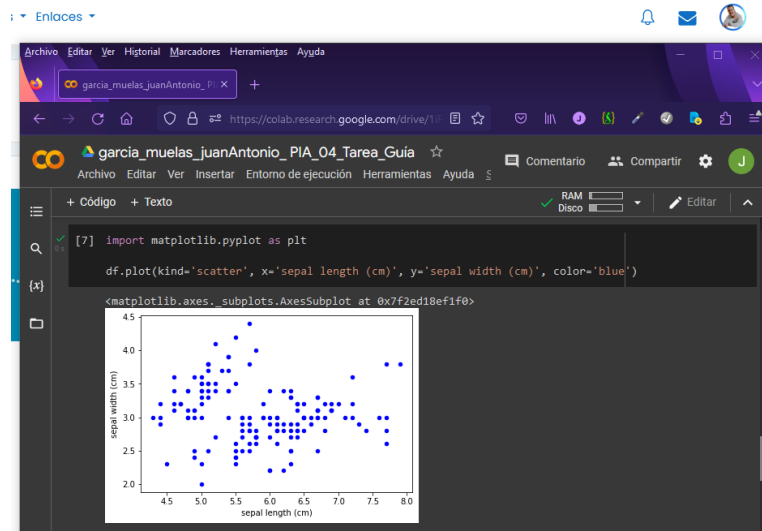


✓ Apartado 2: Visualiza los datos con Pyplot.

- Utiliza el paquete **Pyplot** para hacer representaciones gráficas de los datos. Importa **Pyplot** de la librería **Matplotlib** y crea una figura tipo "dispersión de puntos" (Scatter plot) con la variable *sepal length (cm)* en el eje x y la variable *sepal width (cm)* en el eje y.

```
import matplotlib.pyplot as plt
```

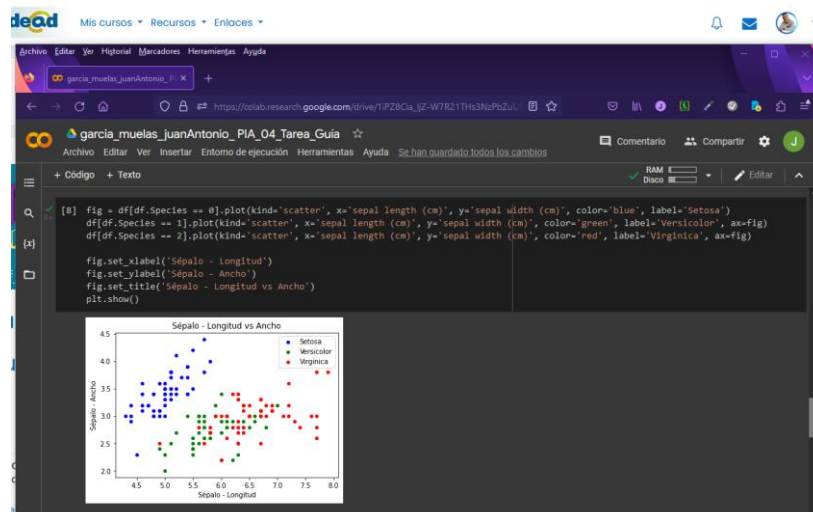
```
df.plot(kind='scatter', x='sepal length (cm)', y='sepal width (cm)', color='blue')
```



- Crea otra figura distinguiendo con el color azul la especie "Setosa", con el color verde la especie "Versicolor" y con color rojo la especie "Virginica".

```
fig = df[df.Species == 0].plot(kind='scatter', x='sepal length (cm)', y='sepal width (cm)', color='blue', label='Setosa')
df[df.Species == 1].plot(kind='scatter', x='sepal length (cm)', y='sepal width (cm)', color='green', label='Versicolor', ax=fig)
df[df.Species == 2].plot(kind='scatter', x='sepal length (cm)', y='sepal width (cm)', color='red', label='Virginica', ax=fig)

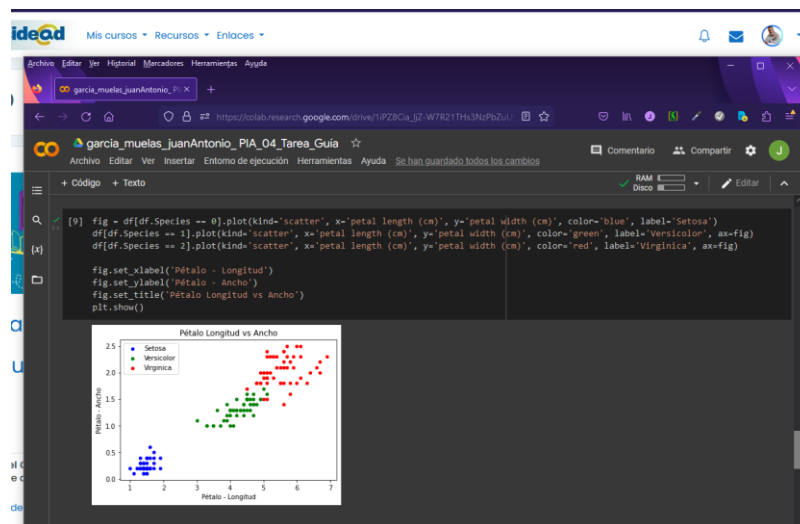
fig.set_xlabel('Sépalo - Longitud')
fig.set_ylabel('Sépalo - Ancho')
fig.set_title('Sépalo - Longitud vs Ancho')
plt.show()
```



- Analiza cómo se distribuyen los casos si, en vez de representar según las variables de las dimensiones de los sépalos, utilizas las variables basadas en las dimensiones de los pétalos. Representa los datos en una figura tipo scatter, utilizando en el eje x la variable "petal length (cm)" y en el eje y la variable "petal width (cm)". De nuevo, distingue las especies con tres colores: color azul la especie "Setosa", con color verde la especie "Versicolor" y con color rojo la especie "Virginica".

```
fig = df[df.Species == 0].plot(kind='scatter', x='petal length (cm)',
y='petal width (cm)', color='blue', label='Setosa')
df[df.Species == 1].plot(kind='scatter', x='petal length (cm)', y='petal
width (cm)', color='green', label='Versicolor', ax=fig)
df[df.Species == 2].plot(kind='scatter', x='petal length (cm)', y='petal
width (cm)', color='red', label='Virginica', ax=fig)

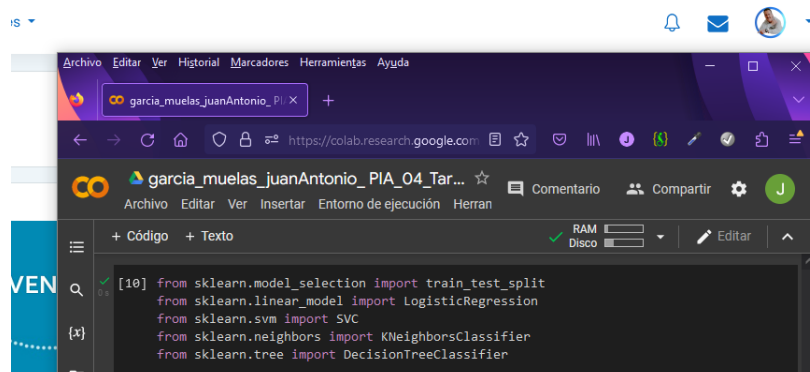
fig.set_xlabel('Pétalo - Longitud')
fig.set_ylabel('Pétalo - Ancho')
fig.set_title('Pétalo Longitud vs Ancho')
plt.show()
```



✓ Apartado 3: Entrena modelos de aprendizaje automático con Scikit-learn.

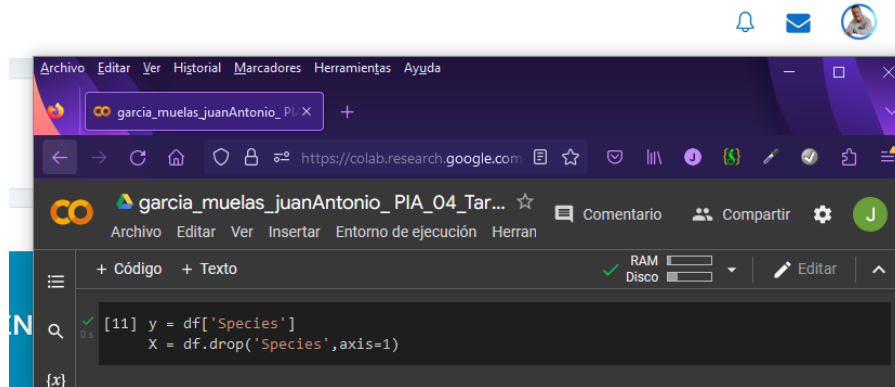
- Importa los módulos de la librería **Scikit-learn** que nos permiten hacer un modelo de **regresión lineal**, un modelo de **máquina de vectores soporte**, un modelo de tipo los **K vecinos más cercanos (KNN)**, y un modelo de tipo **árbol de decisión**.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```



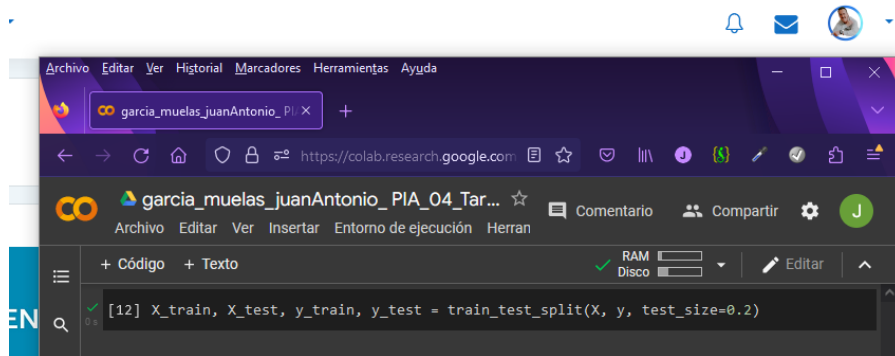
- Genera el conjunto de datos *X* con las variables de entrada a los modelos, y el conjunto *y* de las etiquetas o variable de salida del modelo, eligiendo para este último, la variable "Species".

```
y = df['Species']  
X = df.drop('Species',axis=1)
```



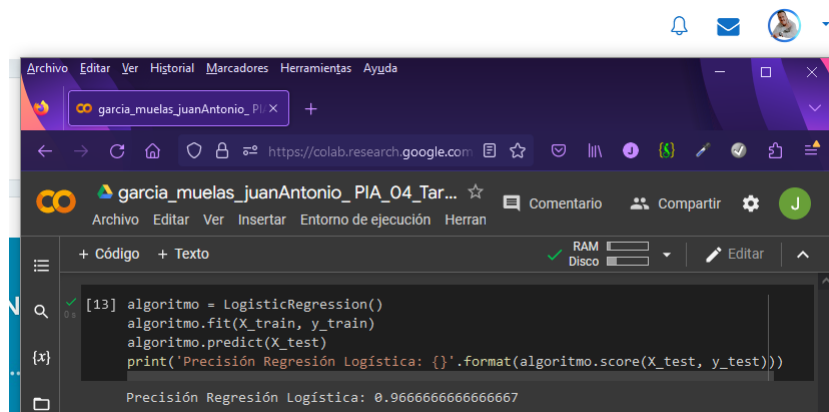
- Utiliza la función *train_test_split* para separar los datos en el conjunto train y test según el ejemplo.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```



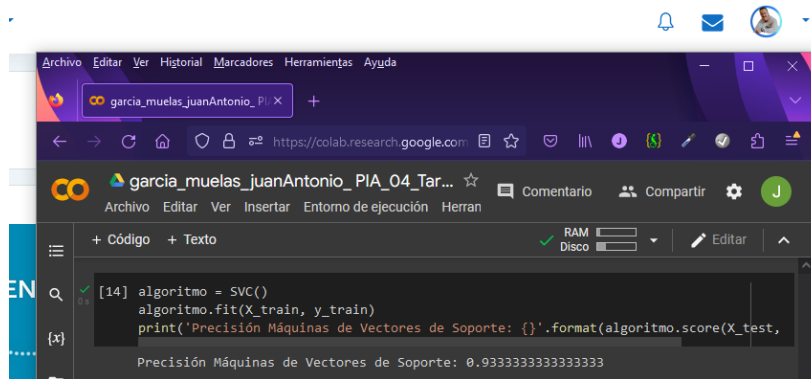
- Crea un modelo de Regresión Logística. Utiliza la función *fit* para entrenarlo y utiliza la función *predict* sobre los datos de test para medir la precisión del modelo. Muestra el valor de dicha precisión con *print*.

```
algoritmo = LogisticRegression()  
algoritmo.fit(X_train, y_train)  
algoritmo.predict(X_test)  
print('Precisión Regresión Logística: {}'.format(algoritmo.score(X_test,  
y_test)))
```



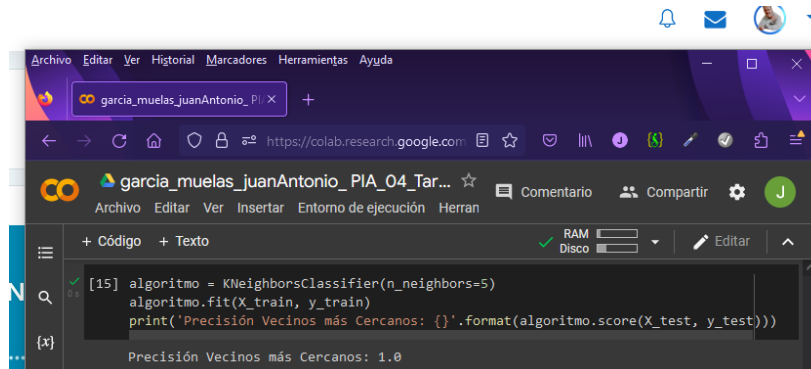
- Crea un modelo de SVC o Máquinas de Vectores de Soporte, entrénalo y calcula la precisión utilizando los datos de test. Muestra la precisión.

```
algoritmo = SVC()
algoritmo.fit(X_train, y_train)
print('Precisión Máquinas de Vectores de Soporte: {}'.format(algoritmo.score(X_test, y_test)))
```



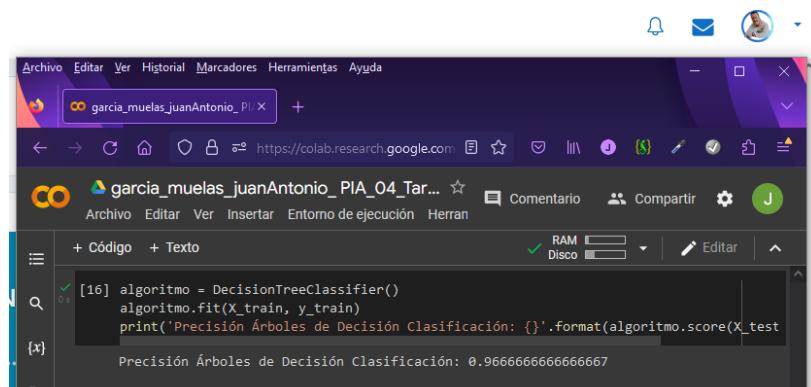
- Crea un modelo de KNN o K vecinos más cercanos, entrénalo y calcula la precisión utilizando los datos de test. Muestra la precisión.

```
algoritmo = KNeighborsClassifier(n_neighbors=5)
algoritmo.fit(X_train, y_train)
print('Precisión Vecinos más Cercanos: {}'.format(algoritmo.score(X_test, y_test)))
```



- Crea un modelo de árbol de decisión, entrénalo y calcula la precisión utilizando los datos de test. Muestra la precisión.

```
algoritmo = DecisionTreeClassifier()
algoritmo.fit(X_train, y_train)
print('Precisión Árboles de Decisión Clasificación: {}'.format(algoritmo.score(X_test, y_test)))
```



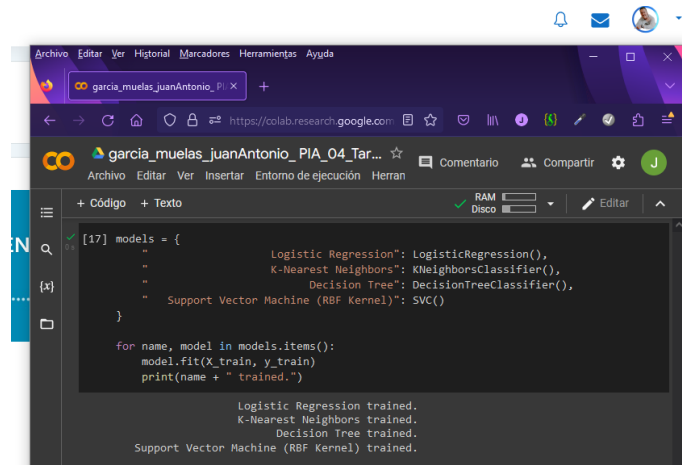
- Compara los valores de precisión que has ido consiguiendo en los diferentes modelos. ¿Cuál sería el mejor para este dataset?

Podría usar ya los datos según están, pero prefiero utilizar más recursos de los ofrecidos en el temario.

Aprovechando el ejemplo facilitado en el tema, lo ajusto a los modelos creados para este ejercicio:

```
models = {
    "Logistic Regression": LogisticRegression(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Support Vector Machine (RBF Kernel)": SVC()
}

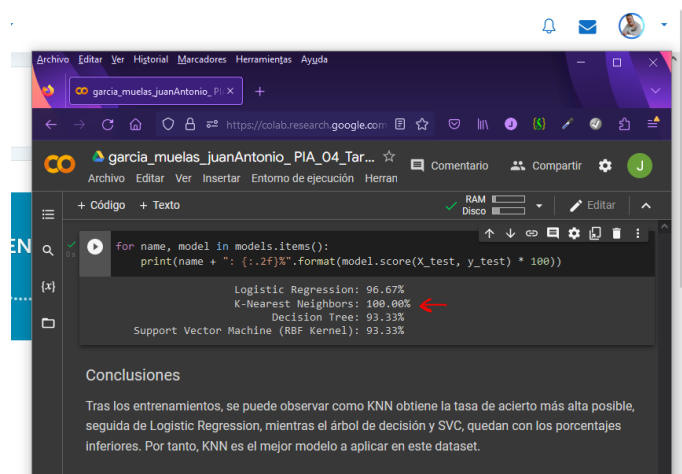
for name, model in models.items():
    model.fit(X_train, y_train)
    print(name + " trained.")
```



En caso de mostrarnos un warning, es relativo a la configuración estándar de este proceso y por la cantidad escasa de datos, por lo que podemos obviarlo.

Con un bucle `for`, recogemos los datos unidos para poder mostrarlos en porcentaje:

```
for name, model in models.items():
    print(name + ": {:.2f}%".format(model.score(X_test, y_test) * 100))
```

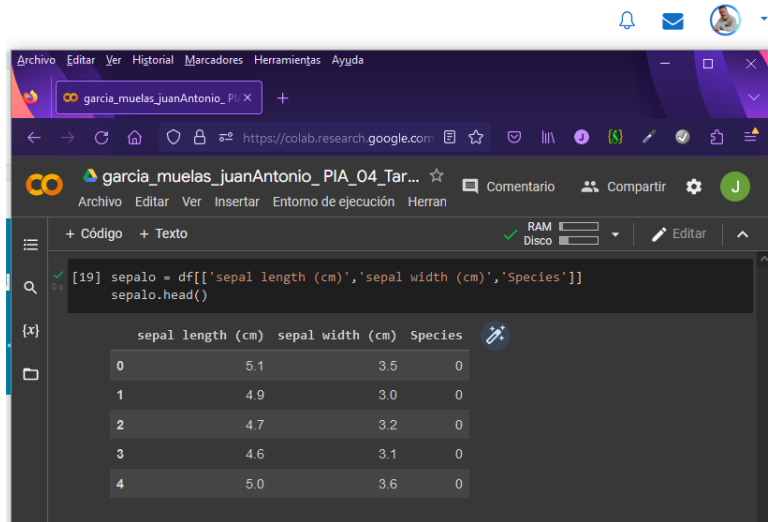


Como se ve en la captura, en un nuevo campo de texto añadido las conclusiones de análisis para que sean observables al abrir el documento. En él, apuntamos a KNN como el mejor modelo, al conseguir un porcentaje del 100%.

✓ **Apartado 4: Entrena modelos de aprendizaje automático con pocas variables.**

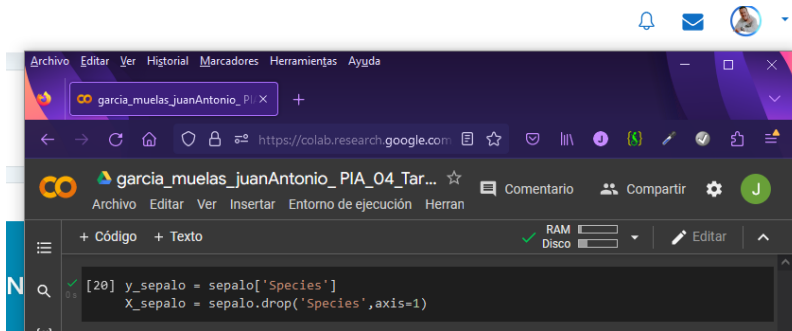
- Imagina que no has podido tener todas las variables, y que solo has conseguido los valores de las medidas de los sépalos, y con esos datos debes entrenar un modelo que acierte con el tipo de especie de flor de iris. Para ello, crea un nuevo dataset que tenga solo las columnas de las dimensiones de los sépalos y la de la especie.

```
sepal = df[['sepal length (cm)', 'sepal width (cm)', 'Species']]
sepal.head()
```



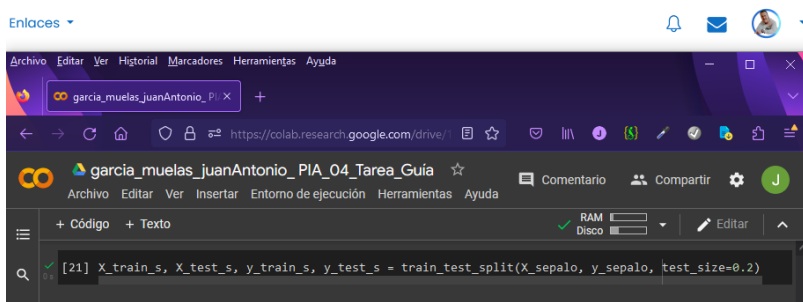
- Separa los datos en `X_sepalo` para las variables de entrada e `y_sepalo` para la variable de salida.

```
y_sepalo = sepal['Species']
X_sepalo = sepal.drop('Species', axis=1)
```



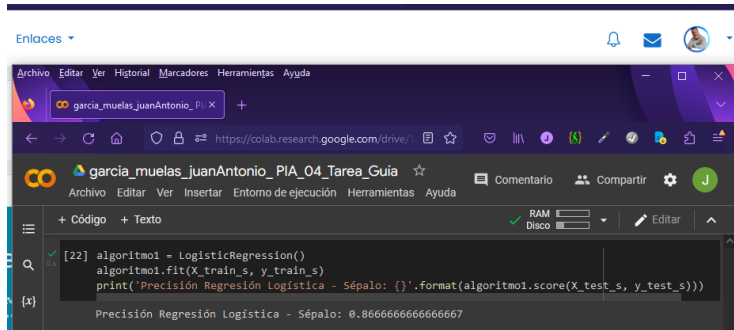
- Separa los datos en un conjunto `X_train_s`, `X_test_s`, `y_train_s`, `y_test_s`, para entrenamiento y test del modelo.

```
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X_sepalo,
y_sepalo, test_size=0.2)
```



- Crea, entrena y mide la precisión de un modelo de Regresión Logística. Muestra la precisión. ¿Es muy diferente al mismo modelo del apartado anterior?

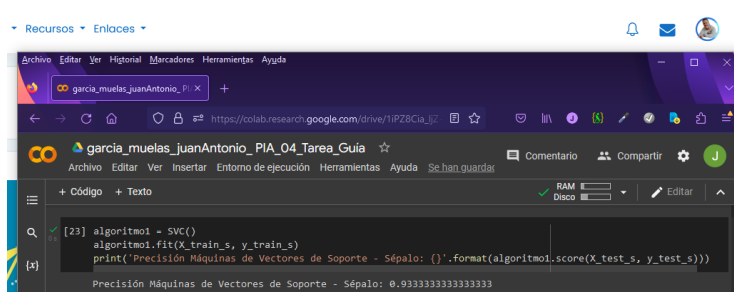
```
algoritmo = LogisticRegression()
algoritmo.fit(X_train_s, y_train_s)
print('Precisión Regresión Logística - Sépalo: {}'.format(algoritmo.score(X_test_s, y_test_s)))
```



Se puede observar que arroja un porcentaje menor respecto al apartado anterior (0.87 frente a 0.97 del apartado 1).

- Crea, entrena y mide la precisión de un modelo de Máquinas de Vectores Soporte. Muestra la precisión ¿Es muy diferente al mismo tipo de modelo del apartado anterior?

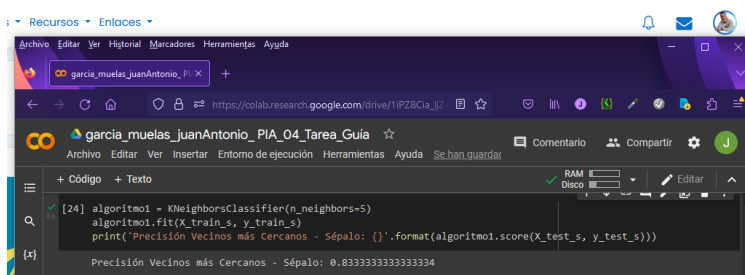
```
algoritmo = SVC()
algoritmo.fit(X_train_s, y_train_s)
print('Precisión Máquinas de Vectores de Soporte - Sépalo: {}'.format(algoritmo.score(X_test_s, y_test_s)))
```



Se puede observar que mantiene el porcentaje con respecto al apartado anterior.

- Crea, entrena y mide la precisión de un modelo de K vecinos más cercanos. Muestra la precisión ¿Es muy diferente al mismo tipo de modelo del apartado anterior?

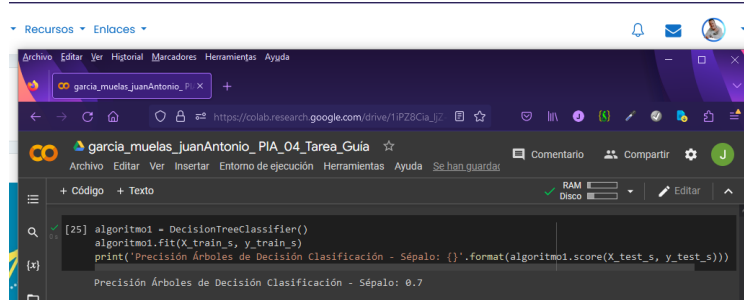
```
algoritmo = KNeighborsClassifier(n_neighbors=5)
algoritmo.fit(X_train_s, y_train_s)
print('Precisión Vecinos más Cercanos - Sépalo: {}'.format(algoritmo.score(X_test_s, y_test_s)))
```



Se puede observar que arroja un porcentaje menor respecto al apartado anterior (0.83 frente a 1 del apartado 1).

- Crea, entrena y mide la precisión de un modelo de Árbol de decisión. Muestra la precisión ¿Es muy diferente al mismo tipo de modelo del apartado anterior?

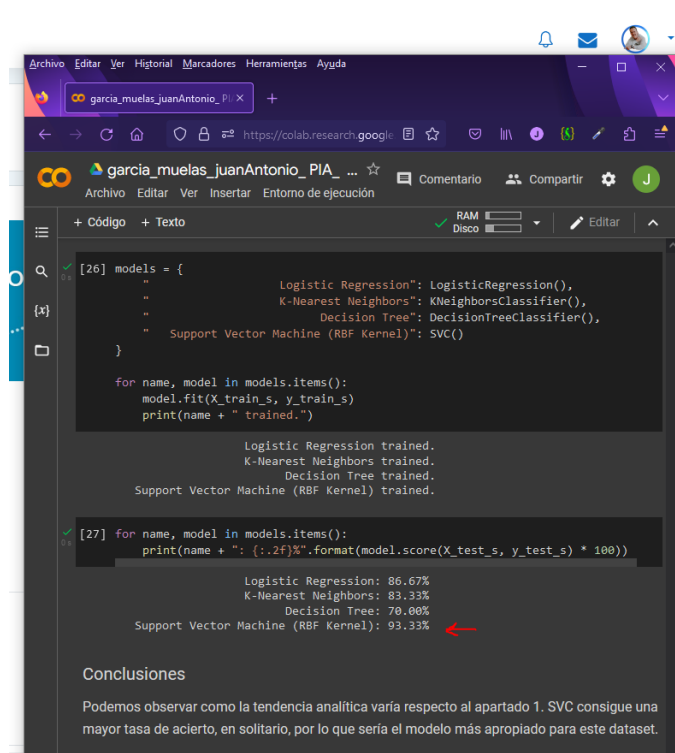
```
algoritmo = DecisionTreeClassifier()
algoritmo.fit(X_train_s, y_train_s)
print('Precisión Árboles de Decisión Clasificación - Sépalo: {}'.format(
    algoritmo.score(X_test_s, y_test_s)))
```



Se puede observar que arroja un porcentaje menor respecto al apartado anterior (0.7 frente a 0.93 del apartado 1).

- Compara los valores de precisión que has ido consiguiendo en los diferentes modelos. ¿Cuál sería el mejor para este dataset?

Vuelvo a realizar el mismo proceso que en el apartado 1 para el análisis.



Puede observarse que el modelo con mayor tasa de acierto pasa a ser Support Vector Machine, bajando tasa sensiblemente KNN, que antes obtuvo un 100% y el Árbol de Decisión seguiría por detrás.

Enlace abierto al notebook: https://colab.research.google.com/drive/1iPZ8Cia_ljZ-W7R21THs3NzPbZuUhghj?usp=sharing