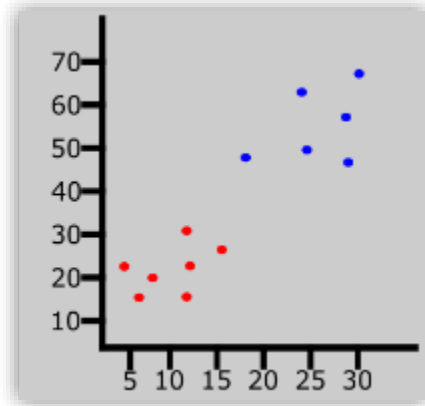


## Contenerización de una aplicación

Tenemos un dataset con datos de la saturación y el brillo y como respuesta, el color:



Con estos datos, vamos a entrenar un modelo que prediga el dato del color dependiendo del dato de la saturación y el brillo.

### A. Generación del modelo

- Elección del modelo:** Como tenemos 2 atributos y solo tenemos que decidir si es Azul o Rojo, vamos a utilizar un modelo SVM.
- Carga del dataset:** Cargamos el dataset desde google drive en un dataframe.
- Normalización de los datos:** para esto, utilizamos MinMaxScaler de sklearn. Agregamos 3 nuevas columnas al dataframe, NBrillo, NSaturacion y NColor. En el caso de este último, hemos cambiado la variable cualitativa, por una cuantitativa (Azul = 1 y Rojo = 0).
- Entrenamos el modelo:** usamos la función fit del modelo para entrenarlo con el total de datos.
- Análisis de métricas:** utilizamos cross\_val\_score, precision\_score, recall\_score, f1\_score para analizar el rendimiento del modelo, con los siguientes resultados:
  - Precisión de la validación cruzada: [1. 0.5 0.5 1.]
  - Precisión media: 0.75
  - Precisión: 1.0
  - Recall: 1.0
  - F1-score: 1.0
- Exportar modelo:** Con el modelo listo, lo exportamos. Usando la librería pickle. Lo denominamos: modelo\_entrenado.pkl

## B. Generación de la interfaz:

1. **Librería para interfaz:** Para la interfaz se utilizó la librería streamlit, que te permite generar páginas con pocas líneas, centrándose en la lógica, más que en el maquetado.
2. **Carga del modelo:** Para cargar el modelo, se utilizó la librería joblib.
3. **Normalización de datos:** Los datos de entrada deben ser normalizados antes de ser utilizados para el modelo. Se ha creado una función que realiza esta tarea, y devuelve el dato normalizado. En la pantalla de la aplicación se puede ver, tanto el dato ingresado, como el normalizado.
4. **Etiqueta del color:** El modelo desarrollado, devuelve un 0 (cero) cuando el color es Rojo, y 1 cuando es Azul. Se ha desarrollado una función que dependiendo del número, devuelve la etiqueta del color.
5. **Inputs y botones de datos:** Se han creado dos inputs donde se ingresan los datos y un botón que se presiona para llamar a la función de predicción del modelo, con los datos normalizados
6. **Iniciar la interfaz:** Para inicializar la interfaz, se debe escribir: `streamlit run app.py` Esto con la finalidad de probar el funcionamiento de la interfaz antes de dockerizarla.

## C. Dockerización de la aplicación

Se han seguido los siguientes pasos:

1. **Creación del Dockerfile:** en este archivo especificamos la versión de Python, la carpeta de trabajo, las librerías y los archivos que se deben incluir en la imagen. Además, se anota el comando con el que se inicia nuestra aplicación.

```
# Python como base
FROM python:3.11.4

# carpeta /app
WORKDIR /app

# Copia los archivos de la aplicación a carpeta
COPY app.py modelo_entrenado.pkl /app/

# Instala las dependencias de la aplicación
RUN pip install streamlit joblib scikit-learn

# Exponer el puerto 8501 para Streamlit
EXPOSE 8501

# Ejecutar la aplicación cuando se inicie el contenedor
CMD ["streamlit", "run", "app.py"]
```

2. **Compilar la imagen de Docker:** se ejecutan las instrucciones del Dockerfile. Esto se hace con el comando:

```
docker build -t ml_cris:latest
```

Donde ml\_cris es el nombre de la imagen que estoy creando. Este dato es variable.

```
PS C:\trabajo\personal\cursos\2023\maestriaBD\modulo14\clase2> docker build -t ml_cris:latest .
2024/05/10 09:10:44 http2: server: error reading preface from client //./pipe/docker_engine: file has already been closed
[+] Building 202.1s (9/9) FINISHED
=> [internal] load build definition from dockerfile
=> => transferring dockerfile: 436B
=> [internal] load metadata for docker.io/library/python:3.11.4
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/python:3.11.4@sha256:85b3d192dddbc96588b719e86991e472b390805a754681a38132de1977d8e429
=> [internal] load build context
=> => transferring context: 68B
=> CACHED [2/4] WORKDIR /app
=> CACHED [3/4] COPY app.py modelo_entrenado.pkl /app/
=> [4/4] RUN pip install streamlit joblib
=> exporting to image
=> => exporting layers
=> => writing image sha256:3618571bf017422beaaecbb85e2b10a37e3377c1998630320ea130abab0ce573
```

Una vez que el proceso termina, la consola nos permite escribir. Ahora procedemos a probar que se ha creado efectivamente con el comando:

```
docker images
```

```
PS C:\trabajo\personal\cursos\2023\maestriaBD\modulo14\clase2> docker images
>>
REPOSITORY    TAG       IMAGE ID      CREATED        SIZE
ml_cris       latest    3618571bf017  2 minutes ago  1.53GB
```

Ahí podemos ver que si esta creada la imagen.

3. **Correr la imagen de Docker:** Se ejecuta el comando:

```
docker run -p 8501:8501 ml_cris
```

Donde se especifica el nombre y puerto de la imagen y el puerto en el equipo físico.

```
PS C:\trabajo\personal\cursos\2023\maestriaBD\modulo14\clase2> docker run -p 8501:8501 ml_cris
Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.

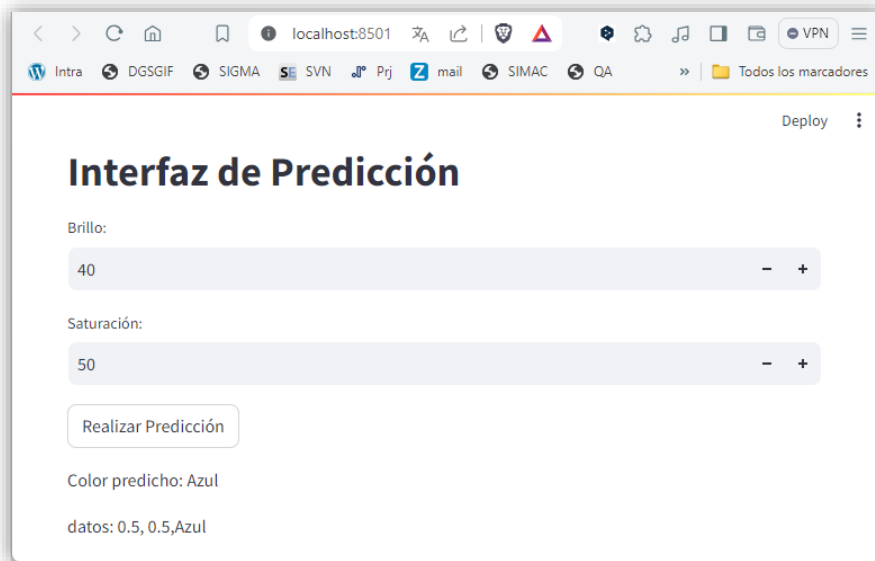
You can now view your Streamlit app in your browser.

Network URL: http://172.17.0.2:8501
External URL: http://192.168.99.80:8501

2024-05-10 13:17:47.474 Uncaught app exception
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/site-packages/streamlit/runtime/scriptrunner/script_runner.py", line 600, in _run_script
    exec(code, module.__dict__)
  File "/app/app.py", line 5, in <module>
    modelo = joblib.load('modelo_entrenado.pkl')
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/site-packages/joblib/numpy_pickle.py", line 658, in load
```

Accedemos a su URL:

```
http://localhost:8501/
```



Funciona!

#### D. Subiendo a Dockerhub

Ahora que nuestra aplicación ha funcionado correctamente, procedemos a subir la imagen a dockerhub

1. **Login en Dockerhub.** Para loguearnos en dockerhub, debemos ejecutar el comando:

`docker login`

```
PS C:\trabajo\personal\cursos\2023\maestriaBD\modulo14\clase2> docker login
Authenticating with existing credentials...
Login Succeeded
```

En mi caso, el login es automático, porque ya estaba registrado mediante la extensión de Docker en VSCode.

2. **Etiquetado de la imagen.** se utiliza el comando:

`docker tag ml_cris crisuruchi/repocris`

Donde `ml_cris` es el nombre de la imagen, `crisuruchi` es el nombre de mi usuario de dockerhub y `repocris`, es el nombre de mi repositorio en dockerhub.

```
PS C:\trabajo\personal\cursos\2023\maestriaBD\modulo14\clase2> docker tag ml_cris crisuruchi/repocris
```

Si no se tiene un repositorio, se crea automáticamente al ejecutar el comando.

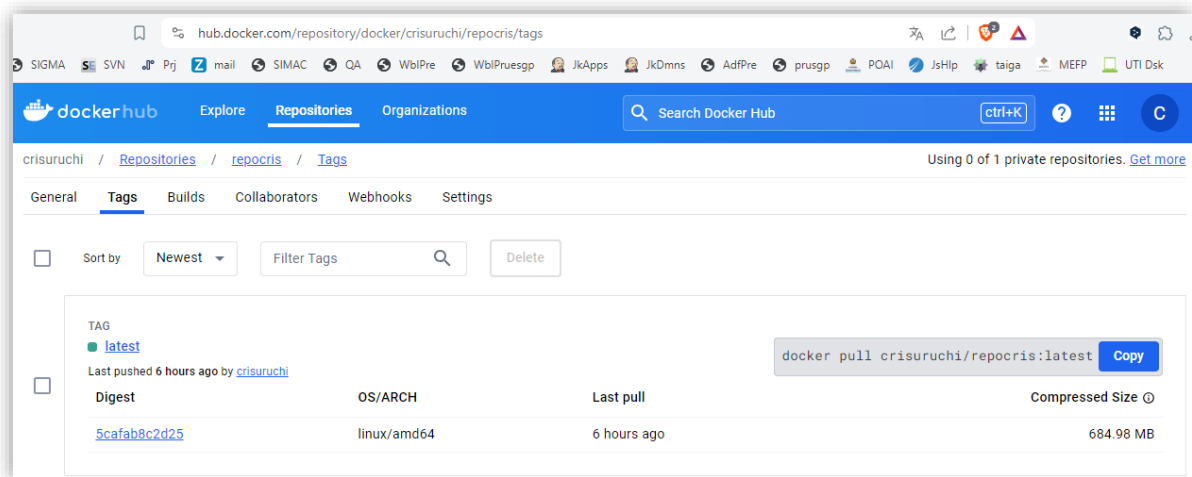
3. **Subir la imagen a dockerhub.** Se utiliza el comando:

`docker push crisuruchi/repocris`

Donde *crisuruchi* es mi nombre de usuario y *repocris* que es el repositorio creado anteriormente.

```
PS C:\trabajo\personal\cursos\2023\maestriaBD\modulo14\clase2> docker push crisuruchi/repocris
Using default tag: latest
The push refers to repository [docker.io/crisuruchi/repocris]
61ba23f317fe: Pushed
fb32fc5f4c6f: Pushed
a24198998210: Pushed
640c66c56f14: Mounted from library/python
76d42947d3a7: Mounted from library/python
854c9b0c3191: Mounted from library/python
b2e5b1eee192: Mounted from library/python
b485c6cd33a6: Mounted from library/python
6aa872026017: Mounted from library/python
43ba18a5eaf8: Mounted from library/python
ff61a9b258e5: Mounted from library/python
latest: digest: sha256:5cafab8c2d25491146a069cf0de8c6a045dd84e4be466506be182e0e62f5524e size: 2634
```

Finalizado, revisamos que se haya subido correctamente.



Para mayor referencia, se puede revisar el código en mi repositorio de github:

<https://github.com/CrisUruchi/appOnDocker/tree/master>

