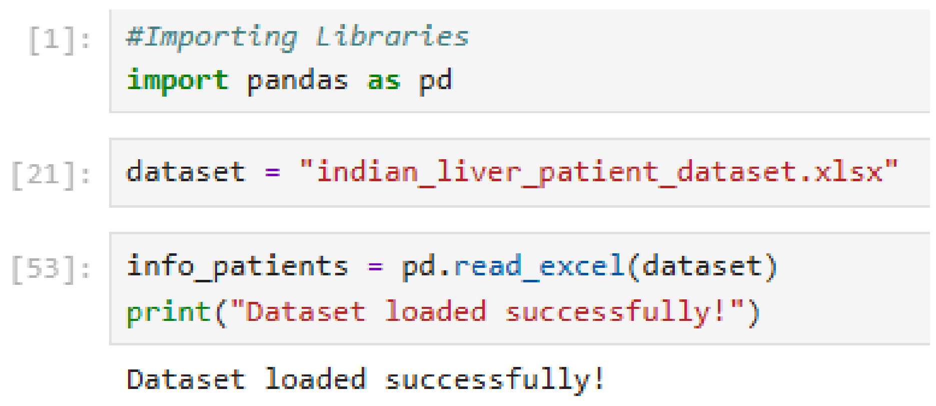
Task 1: Analyze the dataset

1. **Activity 1: Load the Dataset**
    - Load dataset.
    - Print the message: 'Dataset loaded successfully!'.

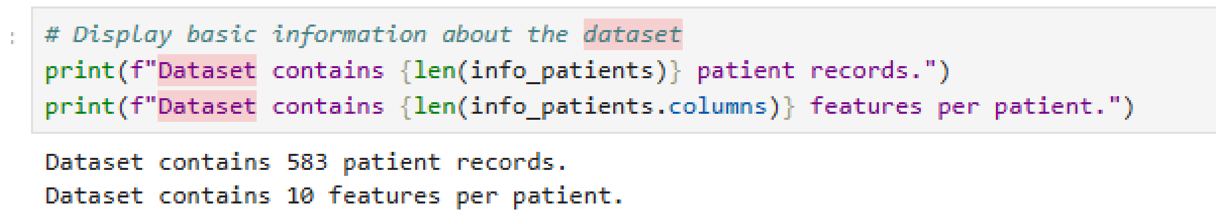


```
[1]: #Importing Libraries
     import pandas as pd
```

```
[21]: dataset = "indian_liver_patient_dataset.xlsx"
```

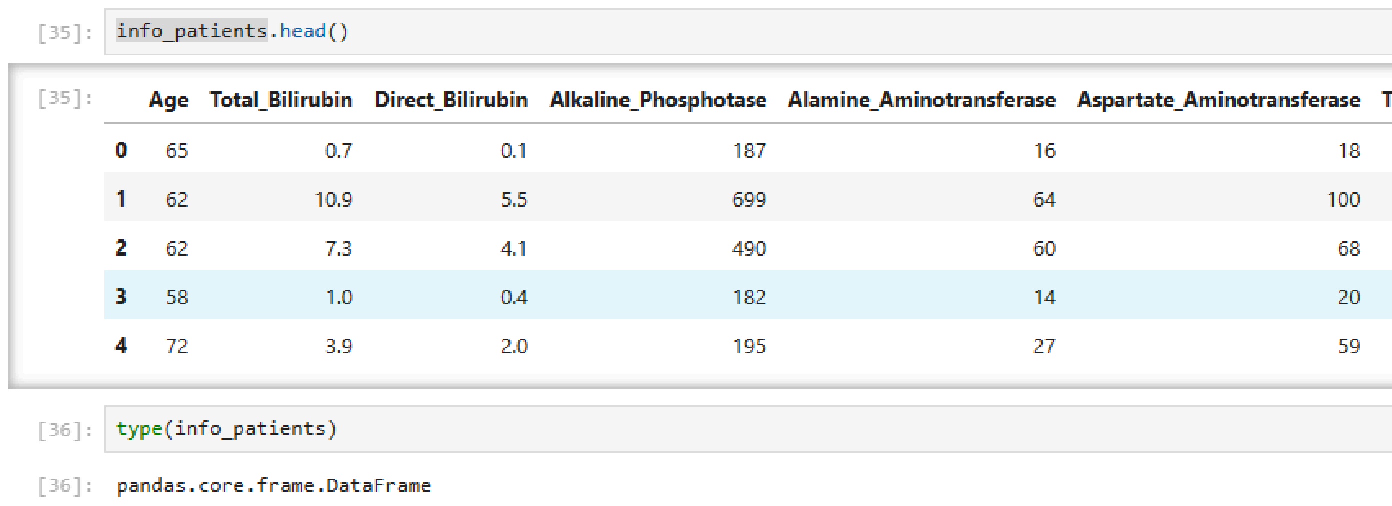
```
[53]: info_patients = pd.read_excel(dataset)
      print("Dataset loaded successfully!")

      Dataset loaded successfully!
```

Activity 2: Print Total Number of Rows



```
: # Display basic information about the dataset
  print(f"Dataset contains {len(info_patients)} patient records.")
  print(f"Dataset contains {len(info_patients.columns)} features per patient.")

  Dataset contains 583 patient records.
  Dataset contains 10 features per patient.
```

Activity 3: Print Top 5 Row



```
[35]: info_patients.head()
```

[35]:

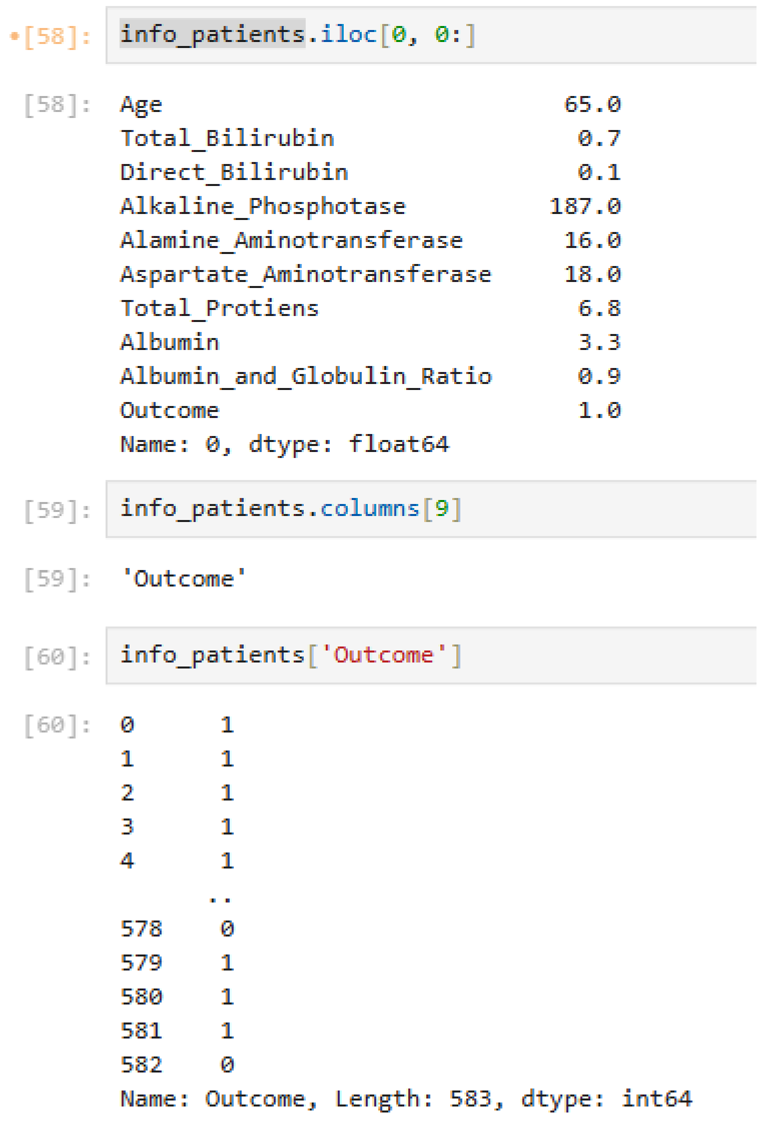
	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase
0	65	0.7	0.1	187	16	18
1	62	10.9	5.5	699	64	100
2	62	7.3	4.1	490	60	68
3	58	1.0	0.4	182	14	20
4	72	3.9	2.0	195	27	59

```
[36]: type(info_patients)

[36]: pandas.core.frame.DataFrame
```

Task 2: Identify the features and target columns

Activity 1: Identify the Target (y) Column

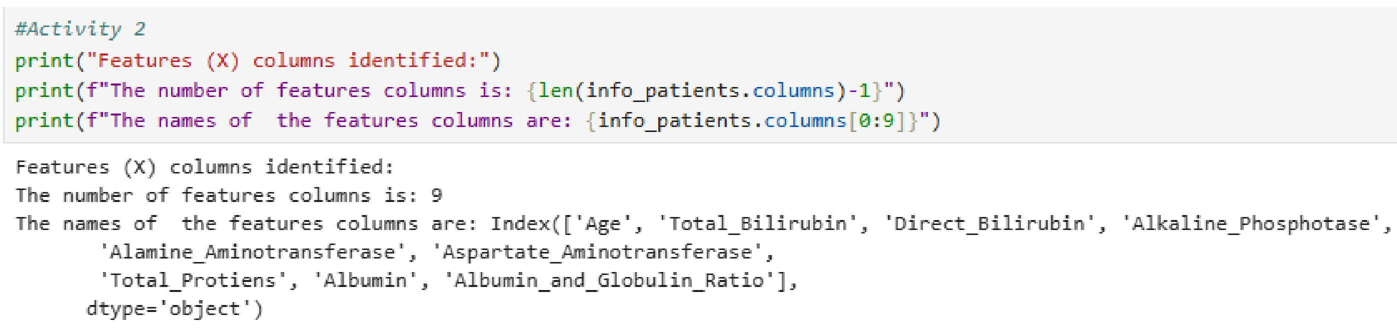
- Print all columns.
- Print the name of the target column.
- Print the values of the target column.



```
•[58]: info_patients.iloc[0, 0:]

 [58]: Age                           65.0
       Total_Bilirubin                0.7
       Direct_Bilirubin               0.1
       Alkaline_Phosphotase         187.0
       Alamine_Aminotransferase      16.0
       Aspartate_Aminotransferase    18.0
       Total_Protiens                 6.8
       Albumin                        3.3
       Albumin_and_Globulin_Ratio     0.9
       Outcome                        1.0
       Name: 0, dtype: float64

 [59]: info_patients.columns[9]

 [59]: 'Outcome'

 [60]: info_patients['Outcome']

 [60]: 0      1
       1      1
       2      1
       3      1
       4      1
             ..
       578    0
       579    1
       580    1
       581    1
       582    0
       Name: Outcome, Length: 583, dtype: int64
```

Activity 2: Identify the Feature (X) Columns

- Display the message: 'Features (X) columns identified'.
- Print the number of feature columns.
- Print the feature column names.

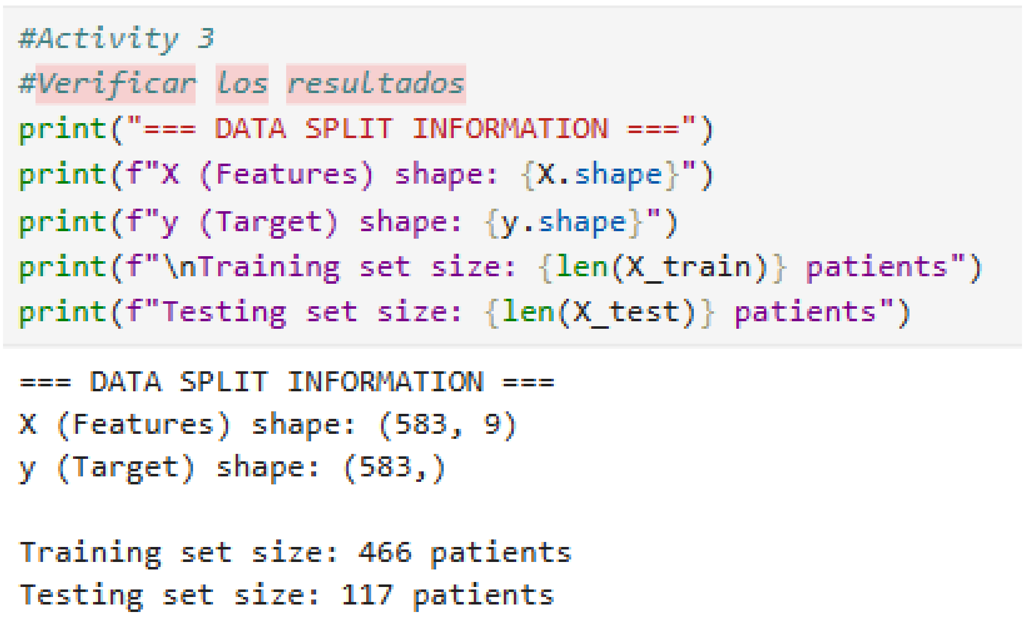


```
#Activity 2
print("Features (X) columns identified:")
print(f"The number of features columns is: {len(info_patients.columns)-1}")
print(f"The names of  the features columns are: {info_patients.columns[0:9]}")
```

```
Features (X) columns identified:
The number of features columns is: 9
The names of  the features columns are: Index(['Age', 'Total_Bilirubin', 'Direct_Bilirubin', 'Alkaline_Phosphotase',
       'Alamine_Aminotransferase', 'Aspartate_Aminotransferase',
       'Total_Protiens', 'Albumin', 'Albumin_and_Globulin_Ratio'],
      dtype='object')
```

Activity 3: Split the Data into Test and Train Sets

- Print the training set size.
- Print the testing set size.
- Print the training features shape.
- Print the testing features shape.



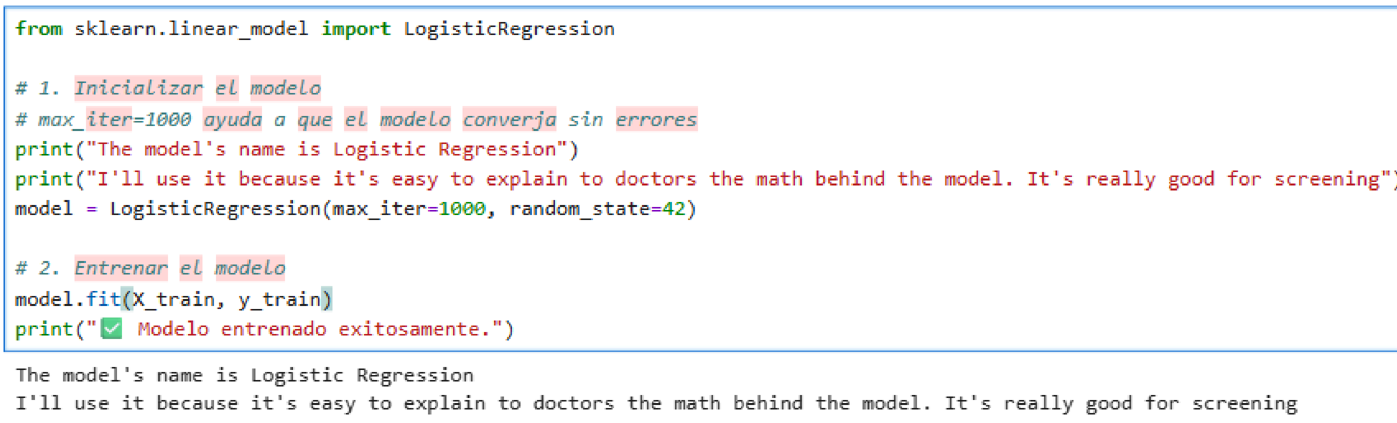
```
#Activity 3
#Verificar los resultados
print("=== DATA SPLIT INFORMATION ===")
print(f"X (Features) shape: {X.shape}")
print(f"y (Target) shape: {y.shape}")
print(f"\nTraining set size: {len(X_train)} patients")
print(f"Testing set size: {len(X_test)} patients")
```

```
=== DATA SPLIT INFORMATION ===
X (Features) shape: (583, 9)
y (Target) shape: (583,)

Training set size: 466 patients
Testing set size: 117 patients
```

Task 3: Choose the classifier and train the model

Activity 1: Choose the Classifier

- Choose the classifier (Use LogisticRegression).
- Print the chosen classifier name.
- Print the reason for choosing it.

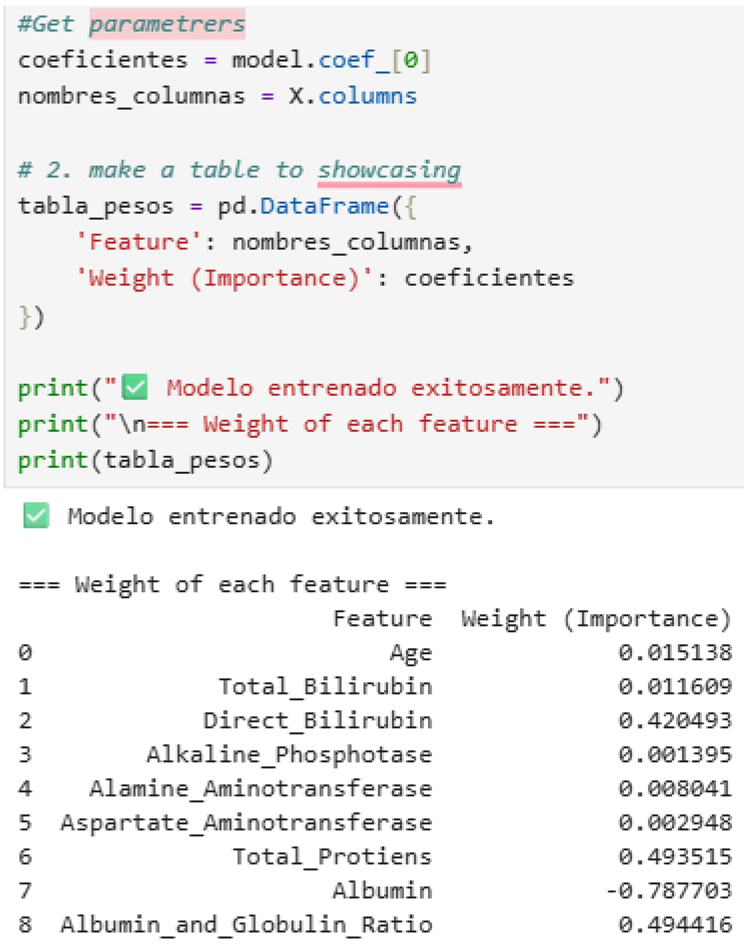


```
from sklearn.linear_model import LogisticRegression

# 1. Inicializar el modelo
# max_iter=1000 ayuda a que el modelo converja sin errores
print("The model's name is Logistic Regression")
print("I'll use it because it's easy to explain to doctors the math behind the model. It's really good for screening")
model = LogisticRegression(max_iter=1000, random_state=42)

# 2. Entrenar el modelo
model.fit(X_train, y_train)
print("✅ Modelo entrenado exitosamente.")
```

```
The model's name is Logistic Regression
I'll use it because it's easy to explain to doctors the math behind the model. It's really good for screening
```

Activity 2: Create the Model

- Create the model.
- Display confirmation that model is created.
- Print the model parameters.

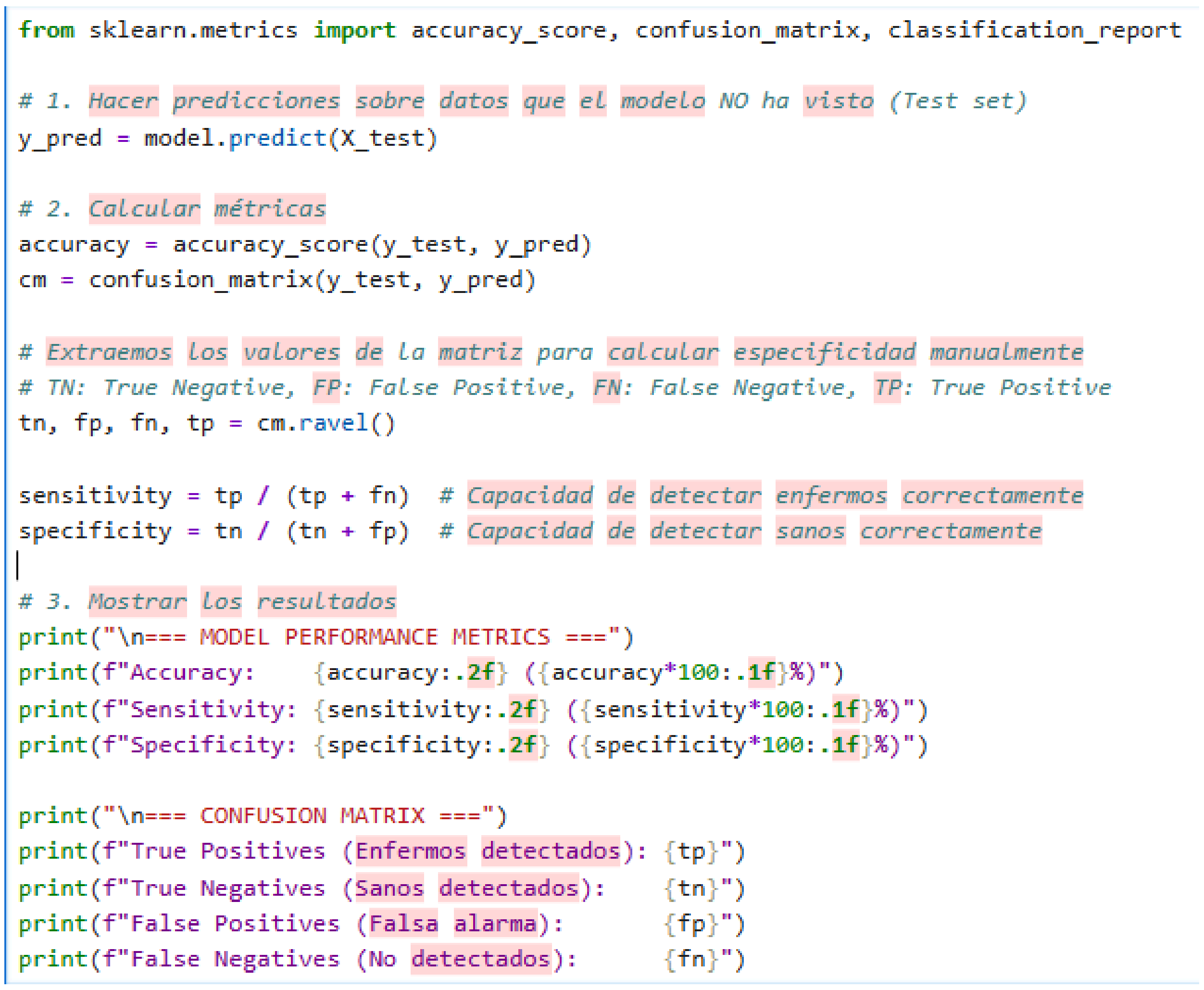


```python
#Get parametrers
coeficientes = model.coef_[0]
nombres_columnas = X.columns

# 2. make a table to showcasing
tabla_pesos = pd.DataFrame({
    'Feature': nombres_columnas,
    'Weight (Importance)': coeficientes
})

print("✅ Modelo entrenado exitosamente.")
print("\n=== Weight of each feature ===")
print(tabla_pesos)
```

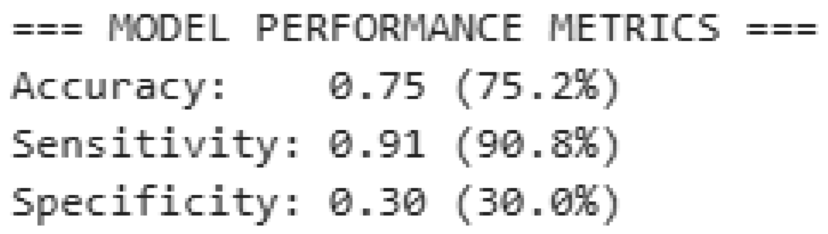
```
✅ Modelo entrenado exitosamente.

=== Weight of each feature ===
                      Feature  Weight (Importance)
0                         Age             0.015138
1             Total_Bilirubin             0.011609
2            Direct_Bilirubin             0.420493
3        Alkaline_Phosphotase             0.001395
4    Alamine_Aminotransferase             0.008041
5  Aspartate_Aminotransferase             0.002948
6              Total_Protiens             0.493515
7                     Albumin            -0.787703
8  Albumin_and_Globulin_Ratio             0.494416
```

Task 4: Evaluate the model

- **Activity 1: Calculate Accuracy**
    - Calculate accuracy.
    - Print accuracy score.
    - Print accuracy percentage.
- **Activity 2: Calculate Sensitivity**
    - Calculate sensitivity.
    - Print sensitivity score.
    - Print sensitivity percentage.
- **Activity 3: Calculate Specificity**
    - Calculate specificity.
    - Print specificity score.
    - Print specificity percentage.



```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# 1. Hacer predicciones sobre datos que el modelo NO ha visto (Test set)
y_pred = model.predict(X_test)

# 2. Calcular métricas
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

# Extraemos los valores de la matriz para calcular especificidad manualmente
# TN: True Negative, FP: False Positive, FN: False Negative, TP: True Positive
tn, fp, fn, tp = cm.ravel()

sensitivity = tp / (tp + fn)  # Capacidad de detectar enfermos correctamente
specificity = tn / (tn + fp)  # Capacidad de detectar sanos correctamente

# 3. Mostrar los resultados
print("\n=== MODEL PERFORMANCE METRICS ===")
print(f"Accuracy:    {accuracy:.2f} ({accuracy*100:.1f}%)")
print(f"Sensitivity: {sensitivity:.2f} ({sensitivity*100:.1f}%)")
print(f"Specificity: {specificity:.2f} ({specificity*100:.1f}%)")

print("\n=== CONFUSION MATRIX ===")
print(f"True Positives (Enfermos detectados): {tp}")
print(f"True Negatives (Sanos detectados):    {tn}")
print(f"False Positives (Falsa alarma):       {fp}")
print(f"False Negatives (No detectados):      {fn}")
```



```
=== MODEL PERFORMANCE METRICS ===
Accuracy:    0.75 (75.2%)
Sensitivity: 0.91 (90.8%)
Specificity: 0.30 (30.0%)
```