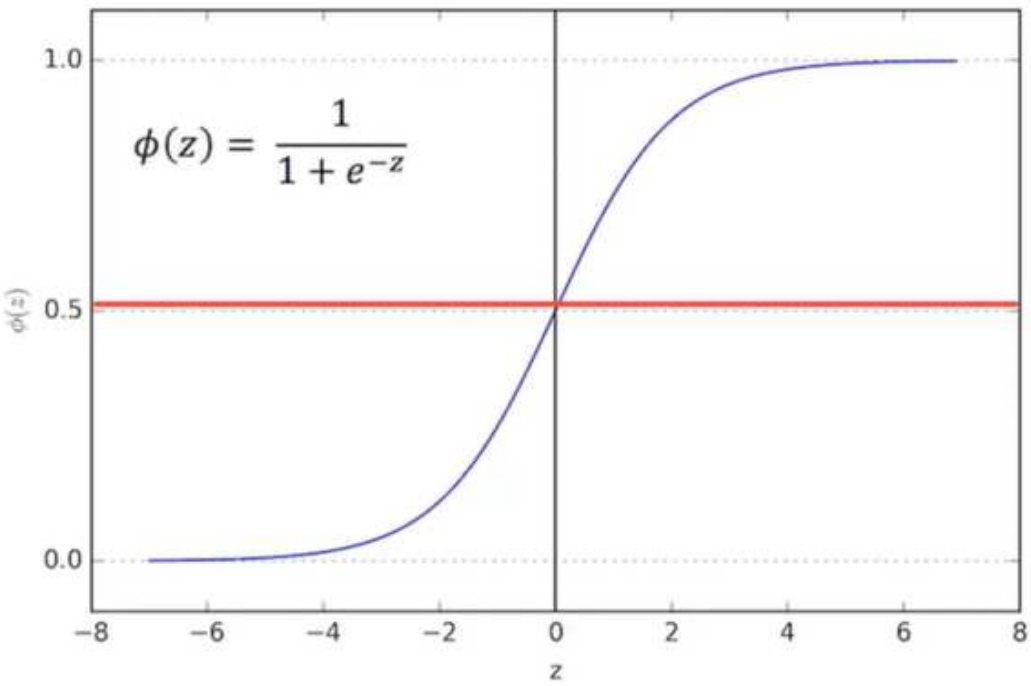
INSTRUCTIVO N° 12: REGRESION LOGISTICA

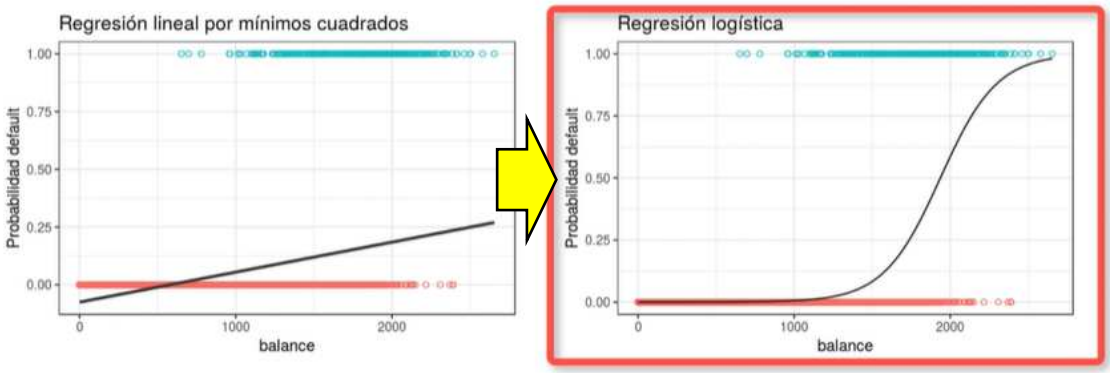
Base Teórica: Gráfico de la Regresión Logística



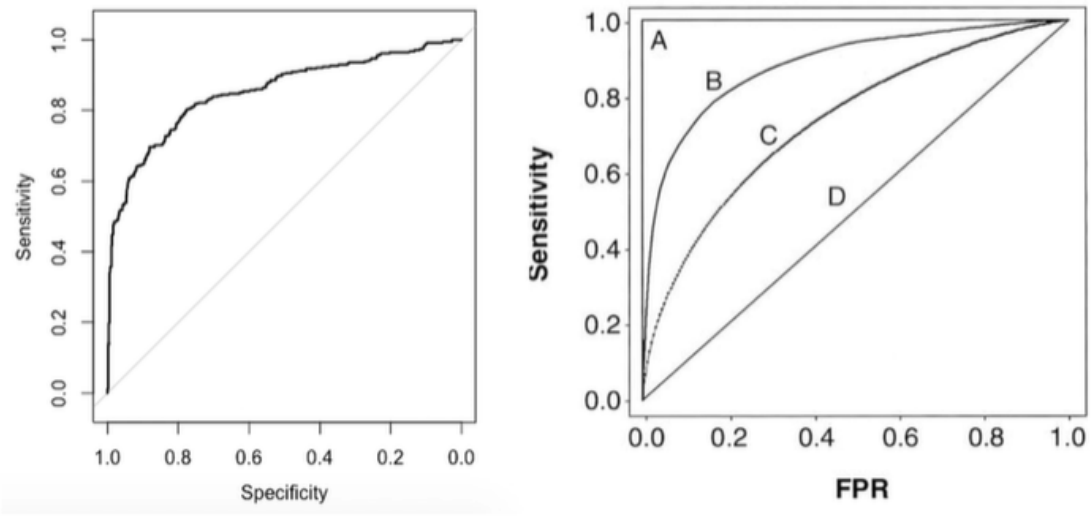
Usos:

- Diagnóstico de enfermedades.
- Detección si es spam o no
- Si o No de las devoluciones.

Observemos esta transformación de las curvas:



Curva ROC:



Cuánto más área abarca la curva es mejor por la mayor precisión.

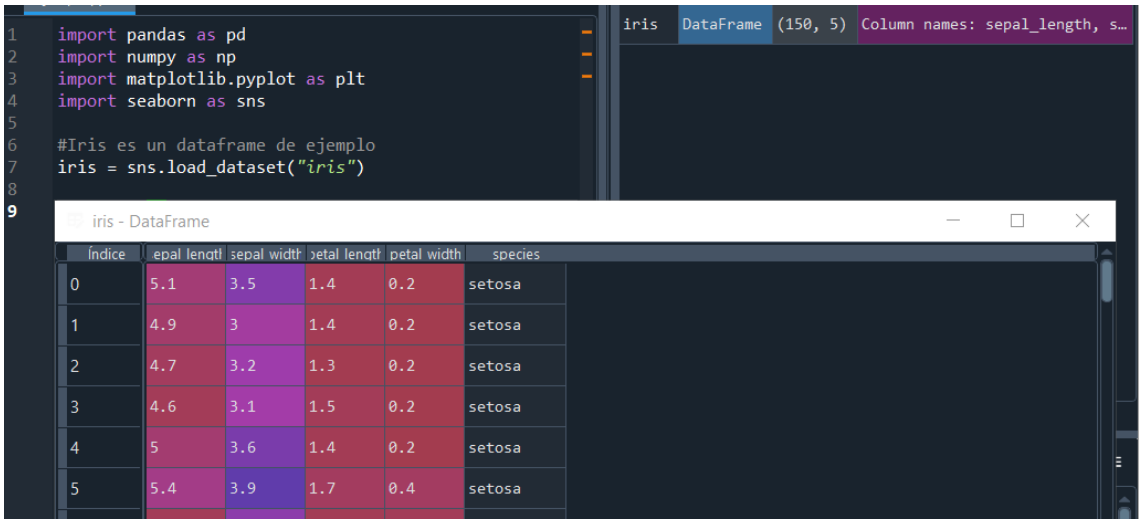
CONCLUSION: Es la representación de la razón o proporción de verdaderos positivos (VPR = Razón de Verdaderos Positivos) frente a la razón o proporción de falsos positivos (FPR = Razón de Falsos Positivos)

1. Importamos las librerías y el dataset de “iris” proveniente de seaborn.

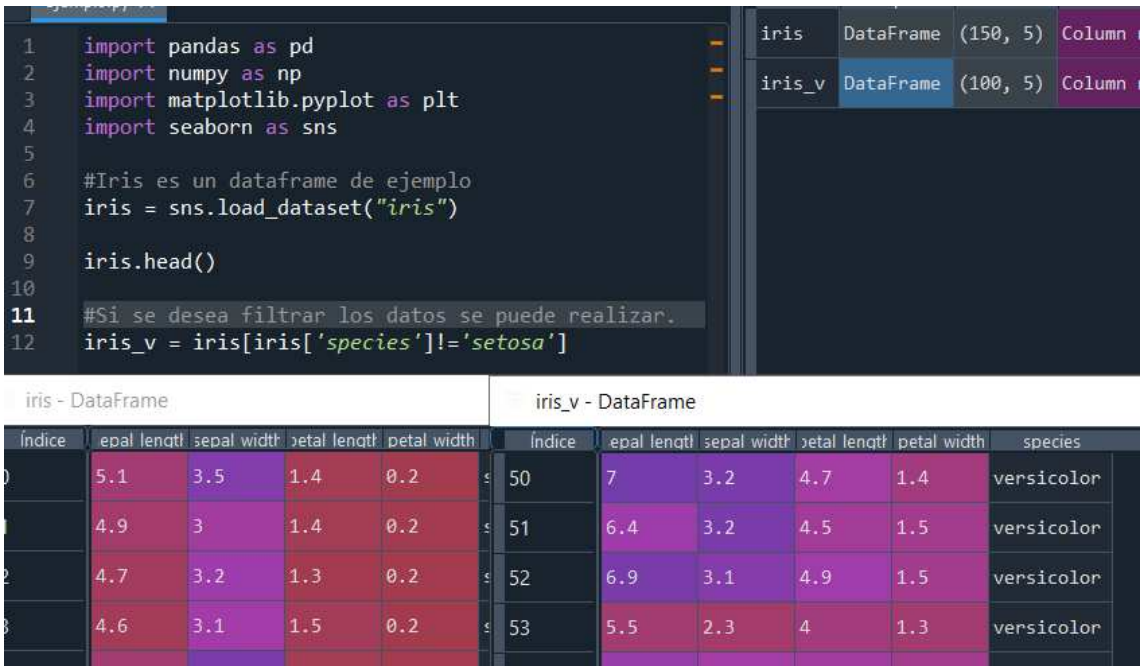
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Iris es un dataframe de ejemplo
iris = sns.load_dataset("iris")
```

2. Para observar los resultados y su contenido:

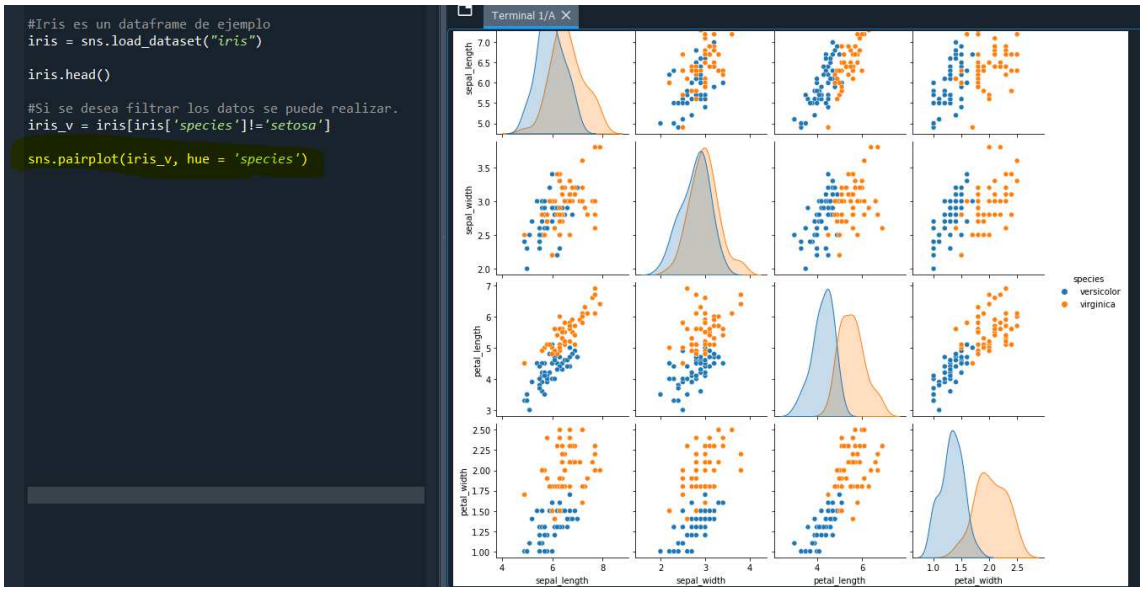


3. Si deseamos hacer consultas podemos utilizar el head para mostrar el contenido de iris, pero la regresión logística trabaja con 1 ó 0 así que especies hay 3 valores por tanto eliminamos un valor (setosa) del grupo (de la columna “especies”).



Por tanto, iris_v no contendrá la palabra setosa.

4. Ahora exploramos la data de iris_v para saber si es una población homogénea.



Observemos que puede que algunos puntos se ajusten a la curva de la regresión logística teóricamente hablando.

5. Construyamos los X como Y, eliminando la columna de las especies.

```
#Eliminamos la columna species
X = iris_v.drop('species', axis=1)
Y = iris_v['species']
```

6. Preparamos el modelo con los datos separados en el paso anterior.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30, random_state=30)
```

0.30 en test_size genera 30% de valores hacia la validación

0.30 en random_state con una aleatorización del 30%

7. Entrenamos el modelo con los datos separados en el paso anterior.

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train, y_train)
```

Nombre	Tipo	Tamaño	Valor
iris	DataFrame	(150, 5)	Column names: sepal_length, se
iris_v	DataFrame	(100, 5)	Column names: sepal_length, se
logmodel	linear_model._logistic.LogisticRegression	1	LogisticRegression object of s
X	DataFrame	(100, 4)	Column names: sepal_length, se
X_test	DataFrame	(30, 4)	Column names: sepal_length, se
X_train	DataFrame	(70, 4)	Column names: sepal_length, se
Y	Series	(100,)	Series object of pandas.core.series module
y_test	Series	(30,)	Series object of pandas.core.series module
y_train	Series	(70,)	Series object of pandas.core.series module

8. Ahora empezaremos al uso del modelo para generar las predicciones.

```
#Iniciamos los test para predicciones
predicciones = logmodel.predict(X_test)
print(predicciones)
```

```
In [6]: print(predicciones)
['versicolor' 'virginica' 'versicolor' 'virginica'
'versicolor'
 'virginica' 'versicolor' 'versicolor' 'virginica'
'virginica' 'virginica'
 'versicolor' 'versicolor' 'virginica' 'versicolor'
```

9. Luego analizamos las métricas asociadas a la precisión, sensibilidad, puntuación. La relación entre y_test y predicciones para analizar la cercanía.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, predicciones))
```

```
Terminal 4/A
'virginica' 'virginica'
 'versicolor' 'virginica' 'virginica']

In [7]: from sklearn.metrics import classification_report
   ...: print(classification_report(y_test, predicciones))
              precision    recall  f1-score   support

  versicolor       0.92      1.00      0.96        12
   virginica       1.00      0.94      0.97        18

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

10. Según la matriz de confusión

```
#Ahora mostramos la tabla de confusión
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predicciones)
```

```
In [8]:
   ...: from sklearn.metrics im
   ...: confusion_matrix(y_test
Out[8]:
array([[12,  0],
       [ 1, 17]], dtype=int64)
```

Matriz de confusión

Predicción

Actual		Positivo	Negativo	
	Positivo	48	2	Error tipo II
	Negativo	5	45	

Error tipo I

Por tanto, prácticamente no hay predicciones erradas.

11. Ahora vamos implementar la curva ROC para obtener quienes forman parte de 1 ó 0 en base a las predicciones. (1 es acierto y 0 es falla)

```
#Ahora vamos a obtener la curva ROC
from sklearn.metrics import roc_curve
y_pred_prob = logmodel.predict_proba(X_test)[:,1]

#Se debe escoger una de las 2 columnas virginica ó versicolor
#porque los valores fluctuarán entre 0 y 1
#donde 0 se asigna a una columna y 1 hacia otra columna
roc_curve(y_test, y_pred_prob, 'virginica')
```

```
error
  warnings.warn(f"Pass {args_msg} as keyword args. From
version "
Out[9]:
(array([0., 0., 0., 0., 0., 1.]),
 array([0.        , 0.05555556, 0.72222222, 0.83333333, 1.
,
        1.        ]),
 array([1.99797124, 0.99797124, 0.81634228, 0.75262194,
0.48809934,
        0.00314452]))
```

12. Obtenemos el Falso-Positivo, Verdadero-Positivo y threshold (umbral de cercanía). Representados como fpr, tpr, threshold respectivamente.

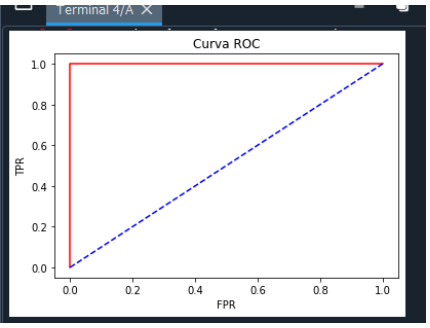
```
fpr, tpr, threshold = roc_curve(y_test, y_pred_prob, 'virginica')
print(fpr)
print(tpr)
print(threshold)
```

```
0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "

In [12]: print(fpr)
    ...: print(tpr)
    ...: print(threshold)
[0. 0. 0. 0. 0. 1.]
[0.         0.05555556 0.72222222 0.83333333 1.         1.        ]
[1.99797124 0.99797124 0.81634228 0.75262194 0.48809934 0.00314452]
```

13. Dibujamos la curva ROC.

```
#Dibujamos
plt.plot(fpr, tpr, color = 'red', label = 'Curva ROC')
plt.plot([0,1],[0,1], color = 'blue', linestyle = '--')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('Curva ROC')
```



No se nota la “curva” propia de la curva ROC, pero podemos realizar un cambio en el paso 6. Para luego aplicarlo en la ejecución de los scripts.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30, random_state=100)
```