

▼ Clasificación y regresión de cáncer de mama en Wisconsin

- **Nombre del proyecto:** Redes Neuronales
- **Nombre de la universidad:** Universidad Panamericana - Campus Ciudad de México
- **Nombre de la materia:** Inteligencia Artificial
- **Integrantes del Equipo:** Sara Miranda, Ivanna Tinedo, Christian Matos, Dania Venegas
- **Fecha de entrega:** 31 de mayo de 2023
- **Versión de código:** 1.0
- Este archivo contiene el entrenamiento de la red neuronal con los datos, su graficación, su clasificación y su modelo de regresión.
- Para ejecutar el código, abra este archivo como un cuaderno de jupyter en un entorno como Anaconda o Google Collab y ejecute todas las celdas.

Tabla de Contenidos

- [Clasificación y regresión de cáncer de mama en Wisconsin](#)
- [Dependencias](#)
- [Sobre los datos](#)
- [Información de atributos](#)
- [Importación de datos](#)
- [Análisis de 5 números](#)
- [Gráfica del conjunto de datos](#)
- [Clasificación](#)
 - [Conjunto de datos para entrenamiento](#)
 - [Escalamiento de los datos](#)
 - [Definición del modelo base](#)
 - [Entrenamiento del modelo base](#)
 - [Pruebas del modelo base](#)
 - [Evaluación del modelo base](#)
 - [Definición de la búsqueda de hiper-parámetros](#)
 - [Ejecución de la búsqueda de hiper-parámetros](#)
 - [Entrenamiento del mejor modelo](#)
 - [Prueba del mejor modelo](#)
 - [Evaluación del mejor modelo](#)
- [Regresión](#)
 - [Conjunto de datos para entrenamiento](#)
 - [Escalamiento de los datos](#)
 - [Definición del modelo base](#)
 - [Entrenamiento del modelo base](#)
 - [Prueba del modelo base](#)
 - [Evaluación del modelo base](#)
 - [Definición de la búsqueda de hiper-parámetros](#)
 - [Ejecución de la búsqueda de hiper-parámetros](#)
 - [Entrenamiento del mejor modelo](#)
 - [Prueba del mejor modelo](#)
 - [Evaluación del mejor modelo](#)

▼ Dependencias

```
# Bibliotecas para manipular los datos y visualizarlos
import pandas as pd
import numpy as np
from math import sqrt
```

```
# Bibliotecas para la graficación
import matplotlib.pyplot as plt
import seaborn as sn
```

```
# Biblioteca para la manipulación de datos y creación de redes neuronales
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
# Métricas de evaluación para la clasificación
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report
```

```
# Métricas de evaluación para regresión
from sklearn.metrics import r2_score
from sklearn.metrics import max_error
from sklearn.metrics import explained_variance_score
```

```
# Para la sintonización de los hiperparámetros
from sklearn.model_selection import GridSearchCV
```

ModuleNotFoundError Traceback (most recent call last)

Cell In[2], line 2

```
1 # Bibliotecas para manipular los datos y visualizarlos
----> 2 import pandas as pd
      3 import numpy as np
      4 from math import sqrt
```

ModuleNotFoundError: No module named 'pandas'

SEARCH STACK OVERFLOW

Sobre los datos

Los datos usados para este modelo de red neuronal corresponden a un estudio de 569 diagnósticos de cáncer de mama en hospitales de Universidad de Wisconsin hasta noviembre de 1995. Los datos fueron computados a partir de imágenes digitalizadas del procedimiento de diagnóstico Fine-Needle Aspiration (FNA). Los datos fueron donados por Nick Street al Machine Learning Repository de UCI en

<https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>.

Sus creadores son:

1. Dr. William H. Wolberg, General Surgery Dept. University of Wisconsin, Clinical Sciences Center Madison, WI 53792 wolberg '@' eagle.surgery.wisc.edu
2. W. Nick Street, Computer Sciences Dept. University of Wisconsin, 1210 West Dayton St., Madison, WI 53706 street '@' cs.wisc.edu 608-262-6619
3. Olvi L. Mangasarian, Computer Sciences Dept. University of Wisconsin, 1210 West Dayton St., Madison, WI 53706 olvi '@' cs.wisc.edu

Estos datos fueron usados originalmente para los siguientes estudios:

1. O. L. Mangasarian and W. H. Wolberg: "Cancer diagnosis via linear programming", SIAM News, Volume 23, Number 5, September 1990, pp 1 & 18.
2. William H. Wolberg and O.L. Mangasarian: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193-9196.
3. O. L. Mangasarian, R. Setiono, and W.H. Wolberg: "Pattern recognition via linear programming: Theory and application to medical diagnosis", in: "Large-scale numerical optimization", Thomas F. Coleman and Yuying Li, editors, SIAM Publications, Philadelphia 1990, pp 22-30.
4. K. P. Bennett & O. L. Mangasarian: "Robust linear programming discrimination of two linearly inseparable sets", Optimization Methods and Software 1, 1992, 23-34 (Gordon & Breach Science Publishers).

Información de atributos

1. Número de identificación
2. Diagnóstico (M = maligno, B = benigno)

Se calculan diez características de valor real para cada núcleo celular:

1. Radio (promedio de distancias desde el centro hasta los puntos en el perímetro)
2. Textura (desviación estándar de los valores en escala de grises)
3. Perímetro
4. Área
5. Suavidad (variación local en las longitudes de radio)
6. Compacidad ($\text{perímetro}^2 / \text{área} - 1.0$)
7. Concavidad (gravedad de las porciones cóncavas del contorno)
8. Puntos cóncavos (número de porciones cóncavas del contorno)
9. Simetría
10. Dimensión fractal ("aproximación de la línea costera" - 1)

▼ Importación de datos

```
# Generar un dataframe de pandas con el archivo que contiene el conjunto de datos
dataset = pd.read_csv(filepath_or_buffer = 'breast-cancer-wisconsin.data', names =

# Eche un vistazo a los datos
print('The dataset contains {} observations and {} features\n'.format(dataset.shape

dataset.head()
```

The dataset contains 699 observations and 11 features

	sample_code	clump_thickness	cell_size	cell_shape	marginal_adhesion	epit
0	1000025	5	1	1	1	
1	1002945	5	4	4	5	
2	1015425	3	1	1	1	
3	1016277	6	8	8	1	
4	1017023	4	1	1	3	

```
# Convierte los dos valores de classd a cadenas de caracteres
diagnostics_mapping = {'2': 'benign', '4': 'malignant'}
dataset['classd'] = dataset['classd'].astype(str)
dataset['classd'] = dataset['classd'].replace(diagnostics_mapping)

dataset.head()
```

	sample_code	clump_thickness	cell_size	cell_shape	marginal_adhesion	epit
0	1000025	5	1	1	1	
1	1002945	5	4	4	5	
2	1015425	3	1	1	1	
3	1016277	6	8	8	1	
4	1017023	4	1	1	3	

```
# Verifique cuántas observaciones tenemos para cada categoría de benigno y maligno.
dataset.groupby('classd').size()
```

```
classd
benign      458
malignant   241
dtype: int64
```

▼ Análisis de 5 números

```
# Use el análisis de cinco números para explorar estadísticamente el conjunto de datos
dataset.describe()
```

	sample_code	clump_thickness	cell_size	cell_shape	marginal_adhesion
count	6.990000e+02	699.000000	699.000000	699.000000	699.000000
mean	1.071704e+06	4.417740	3.134478	3.207439	2.806867
std	6.170957e+05	2.815741	3.051459	2.971913	2.855379
min	6.163400e+04	1.000000	1.000000	1.000000	1.000000
25%	8.706885e+05	2.000000	1.000000	1.000000	1.000000
50%	1.171710e+06	4.000000	1.000000	1.000000	1.000000
75%	1.238298e+06	6.000000	5.000000	5.000000	4.000000
max	1.345435e+07	10.000000	10.000000	10.000000	10.000000

▼ Gráfica del conjunto de datos

```
# Grafique el tamaño de células y la forma de células con diagnósticos benignos y malignos
benign = dataset[dataset.classd == 'benign']
malign = dataset[dataset.classd == 'malignant']
```

```
fig, ax = plt.subplots()
fig.set_size_inches(13, 7) # adjusting the length and width of plot
```

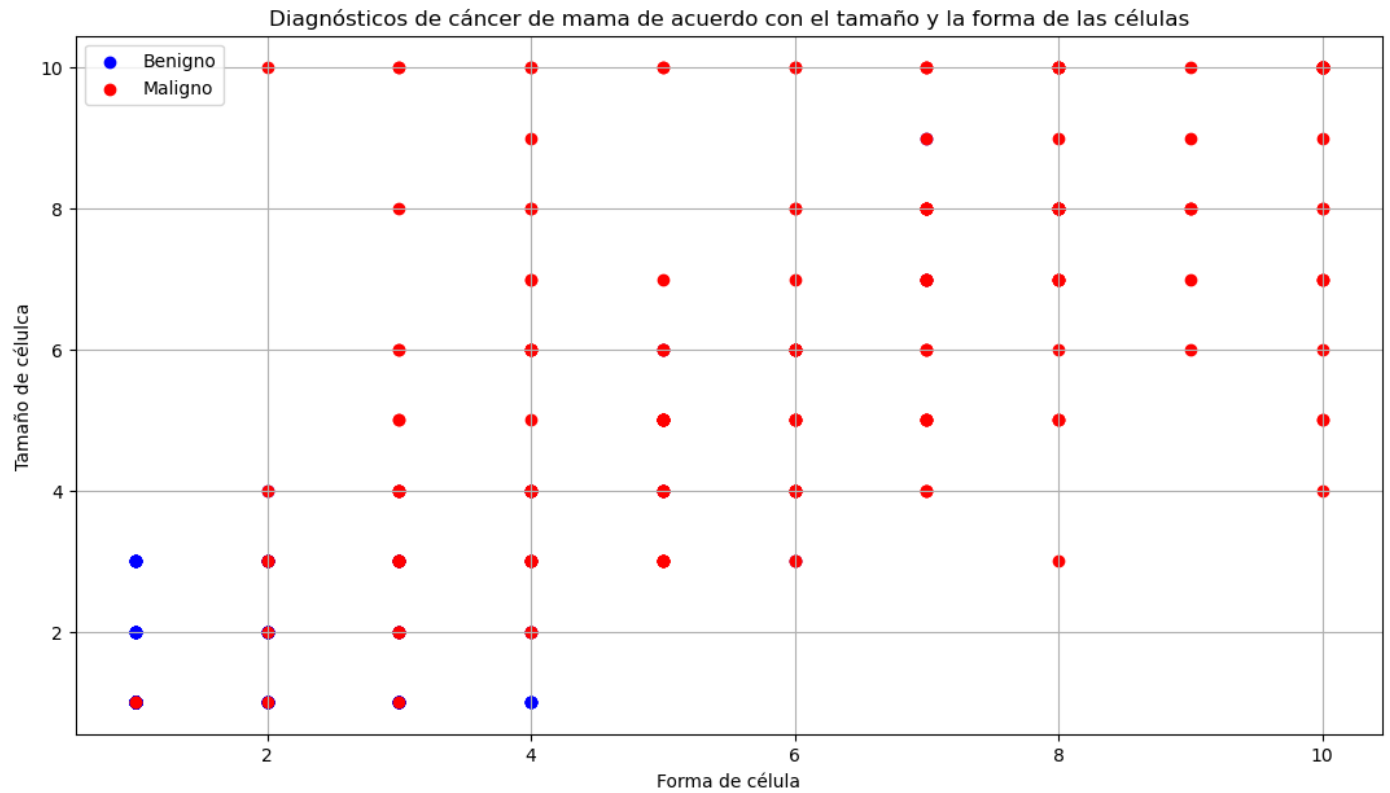
```
# Etiquetas y puntos de dispersión
ax.scatter(benign['cell_shape'], benign['cell_size'], label="Benigno", facecolor="lightblue")
ax.scatter(malign['cell_shape'], malign['cell_size'], label="Maligno", facecolor="lightcoral")
```

```

ax.set_xlabel("Forma de célula")
ax.set_ylabel("Tamaño de célula")
ax.grid()
ax.set_title("Diagnósticos de cáncer de mama de acuerdo con el tamaño y la forma de
ax.legend()

```

<matplotlib.legend.Legend at 0x16b9c864fa0>



▼ Clasificación

▼ Conjunto de datos para entrenamiento

Junto con dataset, estas conforman las variables globales usadas dentro del código.

```
# Tomaremos classd como nuestra característica de interés.
# X será nuestro conjunto de datos sin la característica de interés.
X = dataset.drop('classd', axis=1)

# Convertir en una matriz numpy y asignar el tamaño y la forma de las células como
X = X.to_numpy()[:(2,3)]

# Y será nuestra característica de interés.
y = dataset['classd']

# Dividir para entrenamiento y prueba en proporciones de la mitad.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5,random_state=

print('Los valores min del conjunto de datos son: \n{}'.format(dataset.min()))
print('Los valores máximos del conjunto de datos son: \n{}'.format(dataset.max()))
```

Los valores min del conjunto de datos son:

sample_code	61634
clump_thickness	1
cell_size	1
cell_shape	1
marginal_adhesion	1
epithelial_size	1
bare_nuclei	1
bland_chromatin	1
normal_nucleoli	1
mitoses	1
classd	benign

dtype: object

Los valores máximos del conjunto de datos son:

sample_code	13454352
clump_thickness	10
cell_size	10
cell_shape	10
marginal_adhesion	10
epithelial_size	10
bare_nuclei	?
bland_chromatin	10
normal_nucleoli	10
mitoses	10
classd	malignant

dtype: object

▼ Escalamiento de los datos

```
# Dado que los valores de las características son de diferentes rangos,
# es necesario escalarlos ya que los modelos de ML son sensibles a estas variaciones
scaler = StandardScaler()

# scaler = sc.fit(X_train)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

print('Los valores min del conjunto de datos son: \n{}'.format(X_train_scaled.min()))
print('Los valores máximos del conjunto de datos son: \n{}'.format(X_train_scaled.max()))

Los valores min del conjunto de datos son:
-0.7449149782167317
Los valores máximos del conjunto de datos son:
2.320275080096454
```

▼ Definición del modelo base

```
# Definición de un clasificador de perceptrones múltiples.
# Es una red neuronal artificial que utiliza la función de activación logística.
mlp_clf = MLPClassifier(
    # Definiendo el número de capas ocultas y el número de neuronas en cada capa.
    hidden_layer_sizes=(2),
    # Máximo número de iteraciones para el solucionador de optimización.
    max_iter = 2000,
    # Tolerancia para la optimización.
    learning_rate_init = 0.001,
    # Función de activación para la capa oculta. relu = la unidad lineal rectificada
    activation = 'relu',
    # Optimizador de los pesos. adam = optimizador basado en gradiente estocástico
    solver = 'adam'
)
```

▼ Entrenamiento del modelo base

```
# Entrene el modelo con los datos de entrenamiento.
```

```
mlp_clf.fit(X_train_scaled, y_train)
```

```
MLPClassifier(hidden_layer_sizes=2, max_iter=2000)
```

▼ Pruebas del modelo base

```
# Pasar el conjunto de datos que hemos preparado para las pruebas
```

```
y_pred = mlp_clf.predict(X_test_scaled)
```

```
y_pred
```

```
array(['benign', 'benign', 'benign', 'malignant', 'benign', 'benign',  
      'benign', 'malignant', 'malignant', 'benign', 'benign',  
      'malignant', 'benign', 'benign', 'benign', 'malignant', 'benign',  
      'malignant', 'malignant', 'malignant', 'malignant', 'benign',  
      'malignant', 'benign', 'malignant', 'benign', 'benign',  
      'malignant', 'benign', 'malignant', 'benign', 'benign', 'benign',  
      'malignant', 'benign', 'benign', 'benign', 'malignant', 'benign',  
      'malignant', 'benign', 'benign', 'malignant', 'benign', 'benign',  
      'benign', 'benign', 'malignant', 'benign', 'benign', 'benign',  
      'benign', 'malignant', 'benign', 'malignant', 'benign', 'benign',  
      'malignant', 'benign', 'benign', 'benign', 'benign', 'malignant',  
      'benign', 'benign', 'benign', 'malignant', 'malignant', 'benign',  
      'benign', 'benign', 'benign', 'malignant', 'benign', 'benign',  
      'benign', 'benign', 'benign', 'benign', 'benign', 'benign',  
      'benign', 'benign', 'benign', 'benign', 'benign', 'malignant',  
      'benign', 'benign', 'benign', 'benign', 'malignant', 'malignant',  
      'benign', 'benign', 'benign', 'benign', 'benign', 'malignant',  
      'benign', 'benign', 'malignant', 'benign', 'malignant', 'benign',  
      'benign', 'benign', 'benign', 'benign', 'benign', 'benign',  
      'malignant', 'benign', 'malignant', 'benign', 'malignant',  
      'malignant', 'benign', 'benign', 'benign', 'benign', 'benign',  
      'malignant', 'benign', 'benign', 'benign', 'benign', 'benign',  
      'malignant', 'benign', 'malignant', 'benign', 'benign',  
      'malignant', 'benign', 'benign', 'benign', 'benign', 'benign',  
      'malignant', 'malignant', 'benign', 'benign', 'malignant',  
      'malignant', 'benign', 'benign', 'benign', 'benign', 'malignant',  
      'benign', 'benign', 'benign', 'benign', 'benign', 'benign',  
      'benign', 'benign', 'benign', 'benign', 'benign', 'malignant',  
      'benign', 'benign', 'benign', 'malignant', 'malignant',  
      'malignant', 'malignant', 'malignant', 'malignant', 'benign',  
      'benign', 'benign', 'benign', 'benign', 'benign', 'malignant',  
      'malignant', 'benign', 'benign', 'malignant', 'malignant',
```

```

'benign', 'benign', 'malignant', 'benign', 'benign', 'benign',
'benign', 'malignant', 'malignant', 'malignant', 'malignant',
'benign', 'benign', 'benign', 'malignant', 'benign', 'benign',
'malignant', 'benign', 'benign', 'benign', 'benign', 'benign',
'benign', 'malignant', 'benign', 'benign', 'malignant',
'malignant', 'benign', 'malignant', 'malignant', 'benign',
'benign', 'malignant', 'benign', 'benign', 'benign', 'benign',
'benign', 'benign', 'benign', 'benign', 'malignant', 'malignant',
'benign', 'benign', 'benign', 'benign', 'malignant', 'benign',
'malignant', 'malignant', 'benign', 'malignant', 'malignant',
'benign', 'malignant', 'malignant', 'benign', 'malignant',
'malignant', 'benign', 'malignant', 'benign', 'benign', 'benign',
'malignant', 'malignant', 'benign', 'benign', 'benign', 'benign',
'malignant', 'benign', 'benign', 'malignant', 'malignant',
'malignant', 'malignant', 'benign', 'malignant', 'benign',
'malignant', 'benign', 'benign', 'benign', 'malignant', 'benign',
'benign', 'malignant', 'benign', 'malignant', 'benign',
'malignant', 'malignant', 'benign', 'malignant', 'benign',
'benign', 'benign', 'benign', 'malignant', 'benign', 'malignant',
'benign', 'benign', 'malignant', 'benign', 'benign', 'benign',
'malignant', 'benign', 'benign', 'malignant', 'benign', 'benign',
'benign', 'malignant', 'benign', 'benign', 'benign', 'malignant'

```

▼ Evaluación del modelo base

https://scikit-learn.org/stable/modules/model_evaluation.html

Exactitud

```
print('Exactitud: {:.2f}'.format(accuracy_score(y_test, y_pred)))
```

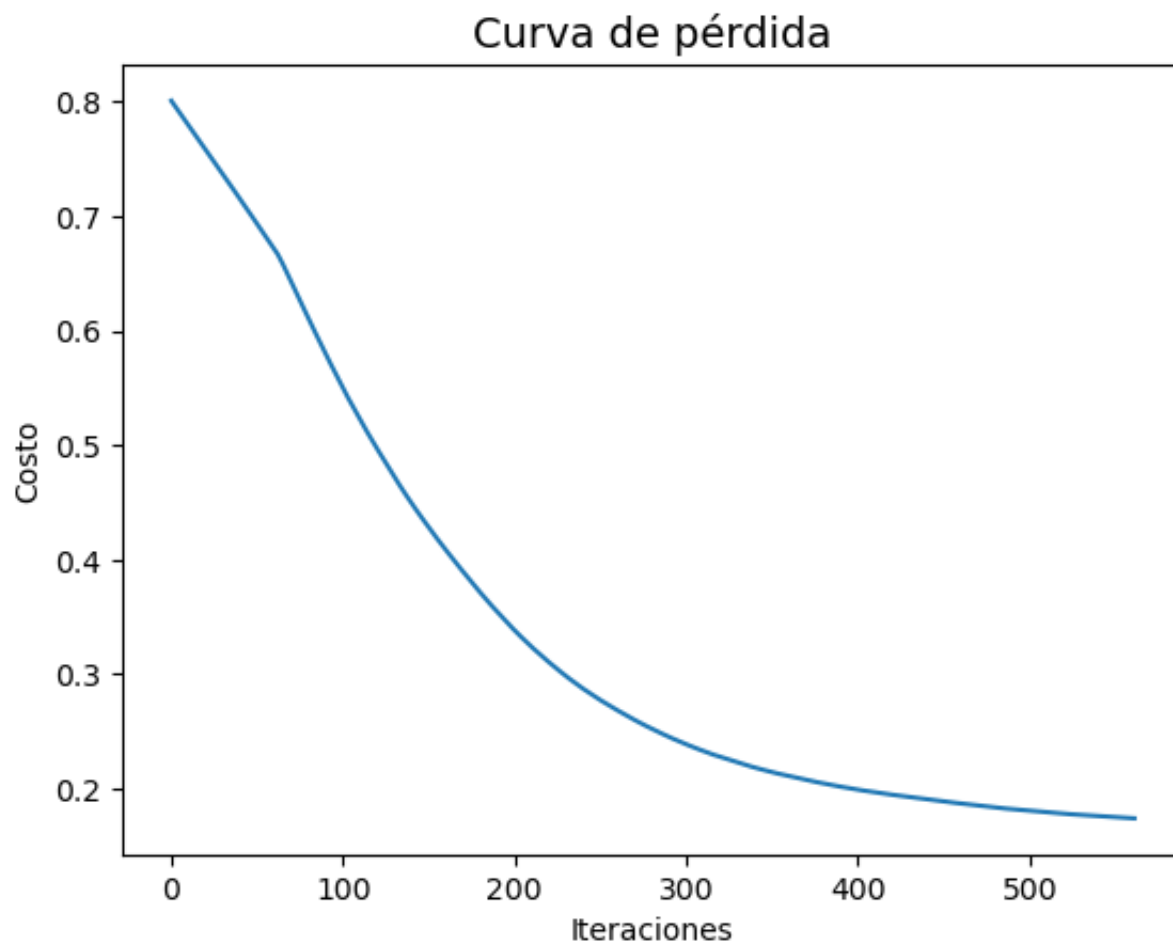
Precisión, recuperación, puntaje F1 y soporte

```
print(classification_report(y_test, y_pred))
```

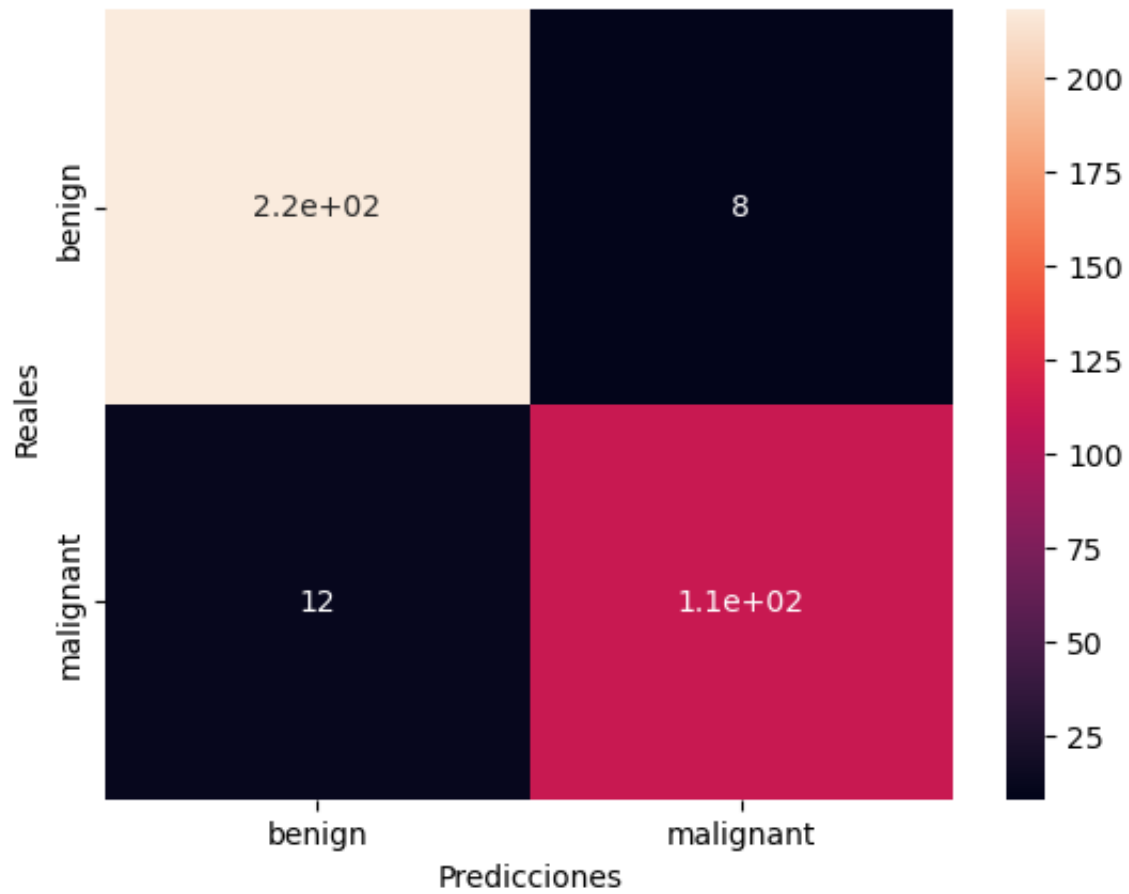
Exactitud: 0.94

	precision	recall	f1-score	support
benign	0.95	0.96	0.96	226
malignant	0.93	0.90	0.92	124
accuracy			0.94	350
macro avg	0.94	0.93	0.94	350
weighted avg	0.94	0.94	0.94	350

```
# Vea qué tan bien fue el entrenamiento del modelo
plt.plot(mlp_clf.loss_curve_)
plt.title("Curva de pérdida", fontsize=14)
plt.xlabel('Iteraciones')
plt.ylabel('Costo')
plt.show()
```



```
# Generar la matriz de confusión
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Reales'], colnames=['Predicciones'])
sns.heatmap(confusion_matrix, annot=True)
plt.show()
```



```
# Eche un vistazo a las predicciones
y_pred_pd = pd.DataFrame(y_pred)
print('Predicciones \n{}\n'.format(y_pred_pd.describe()))

# Eche un vistazo a los verdaderos ejemplos
print('Ejemplos Verdaderos \n{}\n'.format(y_test.describe()))
```

Predicciones

```
0
count      350
unique       2
top      benign
freq       230
```

Ejemplos Verdaderos

```
count      350
unique       2
top      benign
freq       226
Name: class, dtype: object
```

▼ Definición de la búsqueda de hiper-parámetros

```
# Definir un conjunto de parámetros que estamos interesados en entrenar con nuestros
param_grid = {
    'hidden_layer_sizes': [(150,100,50), (120,80), (100)],
    'max_iter': [600, 1200, 1800],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    # Fuerza del término de regularización de L2, que agrega una penalización a los
    'alpha': [0.0001, 0.05],
    # Un valor que funciona como la temperatura en el algoritmo de recocido simulado
    'learning_rate': ['constant', 'adaptive'],
}
```

▼ Ejecución de la búsqueda de hiper-parámetros

```

grid_clf = GridSearchCV(
    # El modelo que estamos entrenando, en este caso un clasificador de perceptrone
    mlp_clf,
    # Los diferentes hiperparámetros que queremos entrenar
    param_grid,
    # Número de procesos que se ejecutarán en paralelo. Si se establece en -1, se u
    n_jobs = -1,
    # Número de veces que se repite la validación cruzada con diferentes conjuntos
    cv = 5
)

```

```

# Ajustar el modelo a los datos de entrenamiento (puede demorar)
grid_clf.fit(X_train_scaled, y_train)

```

```

GridSearchCV(cv=5, estimator=MLPClassifier(hidden_layer_sizes=2,
max_iter=2000),
            n_jobs=-1,
            param_grid={'activation': ['tanh', 'relu'],
                        'alpha': [0.0001, 0.05],
                        'hidden_layer_sizes': [(150, 100, 50), (120, 80),
100],
                        'learning_rate': ['constant', 'adaptive'],
                        'max_iter': [600, 1200, 1800],
                        'solver': ['sgd', 'adam']})

```

```

# Consigue los mejores valores para los hiper-parámetros entrenados
print('Los mejores valores de hiperparámetros son:\n{}'.format(grid_clf.best_params_))

```

```

Los mejores valores de hiperparámetros son:
{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (150, 100, 50),

```

```

# Guarde los resultados como Pandas DataFrame para interactuar fácilmente con los c
grid_results_clf = pd.DataFrame(grid_clf.cv_results_)

```

```

grid_results_clf.sort_values('rank_test_score')

```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_activa
3	1.542868	0.734412	0.002838	0.000546	
15	1.347760	0.863983	0.002006	0.000636	

110	4.999809	0.708491	0.002420	0.000212
118	6.007784	1.011112	0.004512	0.003590
80	7.016711	0.570813	0.001934	0.000650
...
81	4.059874	1.345144	0.002285	0.000746
79	3.303391	1.676667	0.004613	0.004738
77	4.345523	1.575447	0.004043	0.002848
83	5.103198	1.314619	0.002652	0.000734
75	6.536128	1.021818	0.003413	0.001353

144 rows x 19 columns

```
# Generar un codificador directo
def coder(classd):
    if classd == 'benign':
        return 0
    elif classd == 'malignant':
        return 1

# Aplicar el codificador a la característica de interés
dataset['classd'] = dataset['classd'].apply(coder)

dataset.groupby('classd').size()

classd
0      458
1      241
dtype: int64
```

▼ [Entrenamiento del mejor modelo](#)

```
# Definición de un clasificador de perceptrones múltiples.
# Se usan los mejores valores de hiperparámetros encontrados en la sintonización de
mlp_clf_opt = MLPClassifier(
    activation = 'tanh',
    alpha = 0.0001,
    hidden_layer_sizes = (150, 100, 50),
    learning_rate = 'adaptive',
    max_iter = 1200,
    solver = 'adam'
)

# Entrene el modelo con los datos de entrenamiento.
mlp_clf_opt.fit(X_train_scaled, y_train)

MLPClassifier(activation='tanh', hidden_layer_sizes=(150, 100, 50),
              learning_rate='adaptive', max_iter=1200)
```

▼ [Prueba del mejor modelo](#)

```
# Pasar el conjunto de datos que hemos preparado para las pruebas
y_pred = mlp_clf_opt.predict(X_test_scaled)
y_pred
```

[illegible]

```
'benign', 'malignant', 'benign', 'malignant', 'benign',
'malignant', 'malignant', 'benign', 'malignant', 'benign',
'benign', 'benign', 'benign', 'malignant', 'benign', 'malignant',
'benign', 'benign', 'malignant', 'benign', 'benign', 'benign',
'malignant', 'benign', 'benign', 'malignant', 'benign', 'benign',
'benign', 'malignant', 'benign', 'benign', 'benign', 'malignant'
```

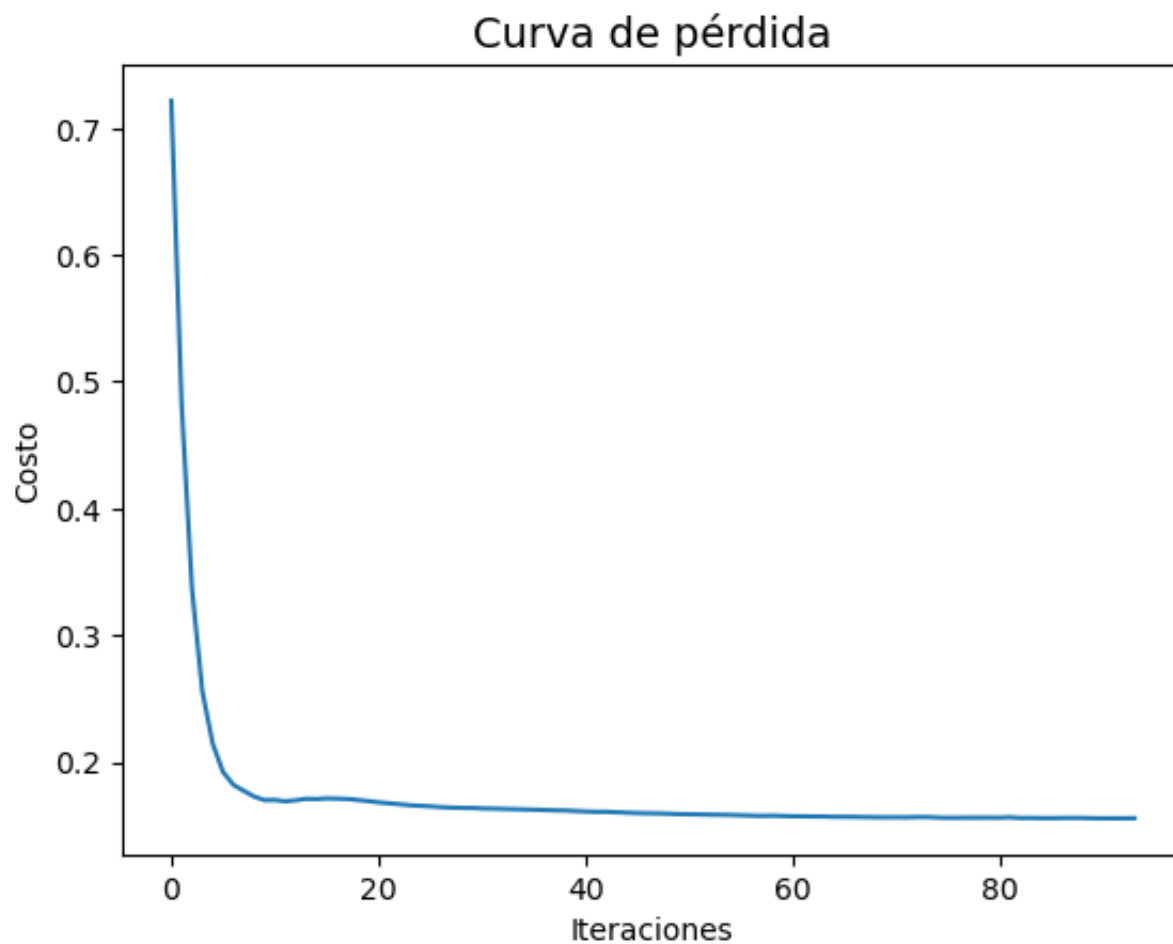
▼ Evaluación del mejor modelo

```
# Exactitud
print('Exactitud: {:.2f}'.format(accuracy_score(y_test, y_pred)))
# Precisión, recuperación, puntaje F1 y soporte
print(classification_report(y_test, y_pred))
```

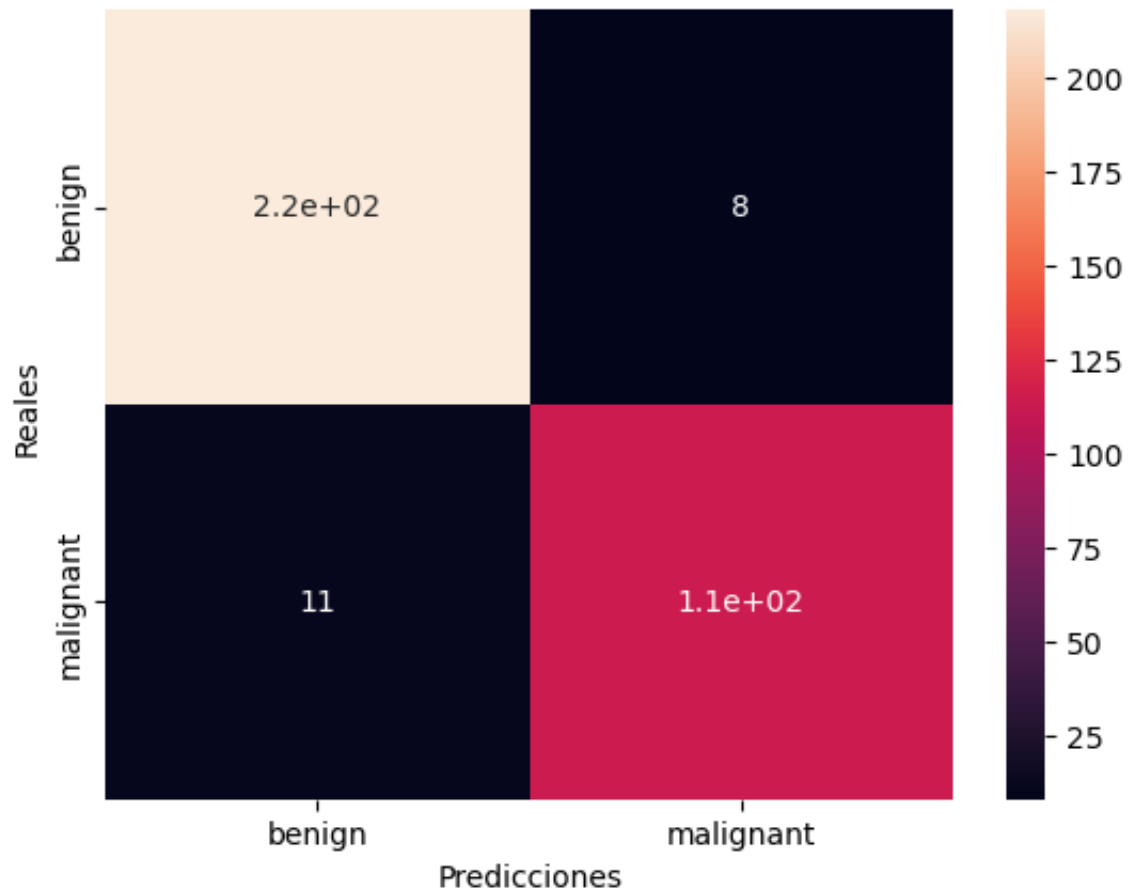
Exactitud: 0.95

	precision	recall	f1-score	support
benign	0.95	0.96	0.96	226
malignant	0.93	0.91	0.92	124
accuracy			0.95	350
macro avg	0.94	0.94	0.94	350
weighted avg	0.95	0.95	0.95	350

```
# Vea qué tan bien fue el entrenamiento del modelo
plt.plot(mlp_clf_opt.loss_curve_)
plt.title("Curva de pérdida", fontsize=14)
plt.xlabel('Iteraciones')
plt.ylabel('Costo')
plt.show()
```



```
# Generar la matriz de confusión
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Reales'], colnames=['Predicciones'])
sns.heatmap(confusion_matrix, annot=True)
plt.show()
```



```
# Eche un vistazo a las predicciones
y_pred_pd = pd.DataFrame(y_pred)
print('Predicciones \n{}\n'.format(y_pred_pd.describe()))

# Eche un vistazo a los verdaderos ejemplos
print('Ejemplos Verdaderos \n{}\n'.format(y_test.describe()))
```

Predicciones

	0
count	350
unique	2
top	benign
freq	229

Ejemplos Verdaderos

count	350
unique	2
top	benign
freq	226

Name: classd, dtype: object

▼ [Regresión](#)

▼ [Conjunto de datos para entrenamiento](#)

Junto con dataset, estas conforman las variables globales usadas dentro del código.

```

# Tomaremos cell_size como nuestra característica de interés.
# X será nuestro conjunto de datos sin la característica de interés.
X = dataset.drop('cell_size', axis=1)

# Convertir en una matriz numpy y asignar el grosor y la adhesión marginal como car
X = X.to_numpy()[:(1,4)]

# Y será nuestra característica de interés.
y = dataset['cell_size']

# Dividir para entrenamiento y prueba en proporciones de la mitad.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5,random_state=

print('Los valores min del conjunto de datos son: \n{}'.format(dataset.min()))
print('Los valores máximos del conjunto de datos son: \n{}'.format(dataset.max()))

```

Los valores min del conjunto de datos son:

```

sample_code      61634
clump_thickness   1
cell_size         1
cell_shape       1
marginal_adhesion 1
epithelial_size   1
bare_nuclei       1
bland_chromatin   1
normal_nucleoli   1
mitoses          1
classd           0
dtype: object

```

Los valores máximos del conjunto de datos son:

```

sample_code      13454352
clump_thickness   10
cell_size         10
cell_shape       10
marginal_adhesion 10
epithelial_size   10
bare_nuclei       ?
bland_chromatin   10
normal_nucleoli   10
mitoses          10
classd           1
dtype: object

```

▼ Escalamiento de los datos


```
# Dado que los valores de las características son de diferentes rangos,
# es necesario escalarlos ya que los modelos de ML son sensibles a estas variaciones
scaler = StandardScaler()

# scaler = sc.fit(X_train)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

print('Los valores min del conjunto de datos son: \n{}'.format(X_train_scaled.min()))
print('Los valores máximos del conjunto de datos son: \n{}'.format(X_train_scaled.max()))

Los valores min del conjunto de datos son:
-1.2403382940641132
Los valores máximos del conjunto de datos son:
3.1795693103863085
```

▼ Definición del modelo base

```
# Definición de un clasificador de perceptrones múltiples.
# Usa los mismos parámetros que el bloque de MLPClassifier anterior.
mlp_reg = MLPRegressor(
    hidden_layer_sizes=(2),
    max_iter = 2000,
    learning_rate_init = 0.001,
    activation = 'relu',
    solver = 'adam'
)
```

▼ Entrenamiento del modelo base

```
# Entrenar el modelo
mlp_reg.fit(X_train_scaled, y_train)

c:\Users\chbas\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_
warnings.warn(
MLPRegressor(hidden_layer_sizes=2, max_iter=2000)
```

▼ Prueba del modelo base

```
# Pasar el conjunto de datos que hemos preparado para las pruebas
```

```
y_pred = mlp_reg.predict(X_test_scaled)
y_pred
```

```
array([ 1.29219537,  1.69580459,  1.29219537,  3.04248831,  1.29219537,
        1.29219537,  2.08886146,  5.6520992 ,  5.34279888,  3.04248831,
        1.29219537,  7.08253947,  1.29219537,  1.29219537,  1.29219537,
        8.03616632,  1.29219537,  7.91498074,  5.51274146,  3.91235861,
        7.76477876,  1.29219537,  5.6520992 ,  1.29219537,  3.12624486,
        2.17261801,  1.69580459,  9.46640627,  1.69580459,  3.99611516,
        1.29219537,  1.69580459,  1.69580459,  3.51930173,  1.29219537,
        2.17261801,  1.29219537,  9.56645822,  1.29219537,  6.46360716,
        1.29219537,  1.29219537,  9.56645822,  1.29219537,  1.29219537,
        2.17261801,  2.17261801,  4.91218164,  1.29219537,  5.26248752,
        3.04248831,  1.29219537,  3.91235861,  1.29219537,  5.25904233,
        1.29219537,  2.17261801,  4.47292858,  1.29219537,  1.29219537,
        1.29219537,  1.29219537,  8.26528663,  2.17261801,  1.29219537,
        1.29219537,  5.34279888,  6.29642573,  2.17261801,  1.29219537,
        4.55668513,  1.29219537,  4.47292858,  1.69580459,  1.69580459,
        2.08886146,  1.29219537,  4.7822289 ,  1.29219537,  1.29219537,
        1.29219537,  1.29219537,  1.29219537,  2.17261801,  1.29219537,
        7.16629602,  3.04248831,  1.61204804,  1.29219537,  1.29219537,
        1.29219537,  1.29219537,  9.46640627,  2.08886146,  2.17261801,
        2.56567489,  1.29219537,  1.29219537,  1.30274772,  1.69580459,
        1.29219537,  2.17261801,  1.29219537,  2.17261801,  5.42655543,
        1.29219537,  4.86598546,  2.00510491,  2.17261801,  3.96131593,
        4.7822289 ,  1.69580459,  1.29219537,  1.29219537,  2.17261801,
        5.8196123 ,  1.29219537,  1.69580459,  2.87497521,  1.29219537,
        1.29219537,  4.31162181,  1.29219537,  1.29219537,  4.56187575,
        3.82860206,  1.29219537,  2.17261801,  6.21266918,  5.42655543,
        1.29219537,  1.29219537,  4.47292858,  1.29219537,  3.51930173,
        1.29219537,  6.12891263,  3.91235861,  1.29219537,  1.29219537,
        2.17261801,  2.17261801,  1.29219537,  2.17261801,  1.29219537,
        9.81671216,  1.29219537,  6.21266918,  1.29219537,  6.29642573,
        1.29219537,  5.42655543,  1.29219537,  8.36533858,  1.29219537,
        8.90603662,  1.29219537,  1.29219537,  2.64943144,  1.29219537,
        1.29219537,  7.16629602,  1.69580459,  3.91235861,  3.04248831,
        1.29219537,  8.03616632,  1.29219537,  4.55668513,  2.95873176,
        1.29219537,  1.29219537,  6.12891263,  4.38917203,  1.29219537,
        1.29219537,  10.16701804,  7.08253947,  1.29219537,  1.69580459,
        1.29219537,  1.29219537,  10.16701804,  1.29219537,  1.29219537,
        2.17261801,  4.38917203,  1.69580459,  1.29219537,  1.61204804,
        2.64943144,  1.29219537,  2.56567489,  2.64943144,  1.69580459,
        1.69580459,  1.29219537,  1.29219537,  7.16421893,  7.06416699,
        7.5593529 ,  8.03616632,  6.21266918,  5.42655543,  1.69580459,
        1.29219537,  1.29219537,  1.69580459,  1.69580459,  1.29219537,
        8.9658984 ,  7.16421893,  1.29219537,  1.29219537,  6.46360716,
        7.16629602,  3.04248831,  1.29219537,  5.16243557,  1.29219537,
        1.29219537,  2.17261801,  1.61204804,  3.60305828,  6.60572605,
        6.29642573,  3.60305828,  1.29219537,  1.69580459,  1.29219537,
        3.43554518,  1.29219537,  1.29219537,  6.21266918,  1.29219537,
        1.29219537,  1.29219537,  1.69580459,  1.29219537,  1.69580459,
        10.16701804,  1.29219537,  1.29219537,  4.47292858,  5.26248752,
        1.29219537,  3.60305828,  4.38917203,  1.29219537,  2.17261801,
        5.34279888,  1.29219537,  1.29219537,  1.29219537,  1.29219537]
```

2.17261801,	1.29219537,	2.17261801,	1.29219537,	5.34279888,
8.90603662,	2.17261801,	1.69580459,	1.29219537,	2.17261801,
5.34279888,	1.69580459,	4.7822289,	9.46640627,	1.69580459,
3.43554518,	3.43554518,	1.29219537,	3.99611516,	7.08253947,
2.17261801,	3.91235861,	4.86598546,	2.64943144,	7.16421893,
1.29219537,	3.60305828,	1.29219537,	2.56567489,	5.16243557,
1.29219537,	1.29219537,	1.29219537,	1.29219537,	2.17261801,
2.17261801,	2.64943144,	5.42655543,	8.03616632,	2.95873176,
7.16421893,	1.60590450,	2.42554510,	1.20210537,	4.06136700,

▼ Evaluación del modelo base

https://scikit-learn.org/stable/modules/model_evaluation.html

Prueba de r2. La mejor puntuación posible es 1.0 y puede ser negativa (porque el
 print('r2: {:.2f}'.format(r2_score(y_test, y_pred)))

Error cuadrático promedio. La mejor puntuación posible es 0.0
 print('Error cuadrático promedio: {:.2f}'.format(mean_squared_error(y_test, y_pred))

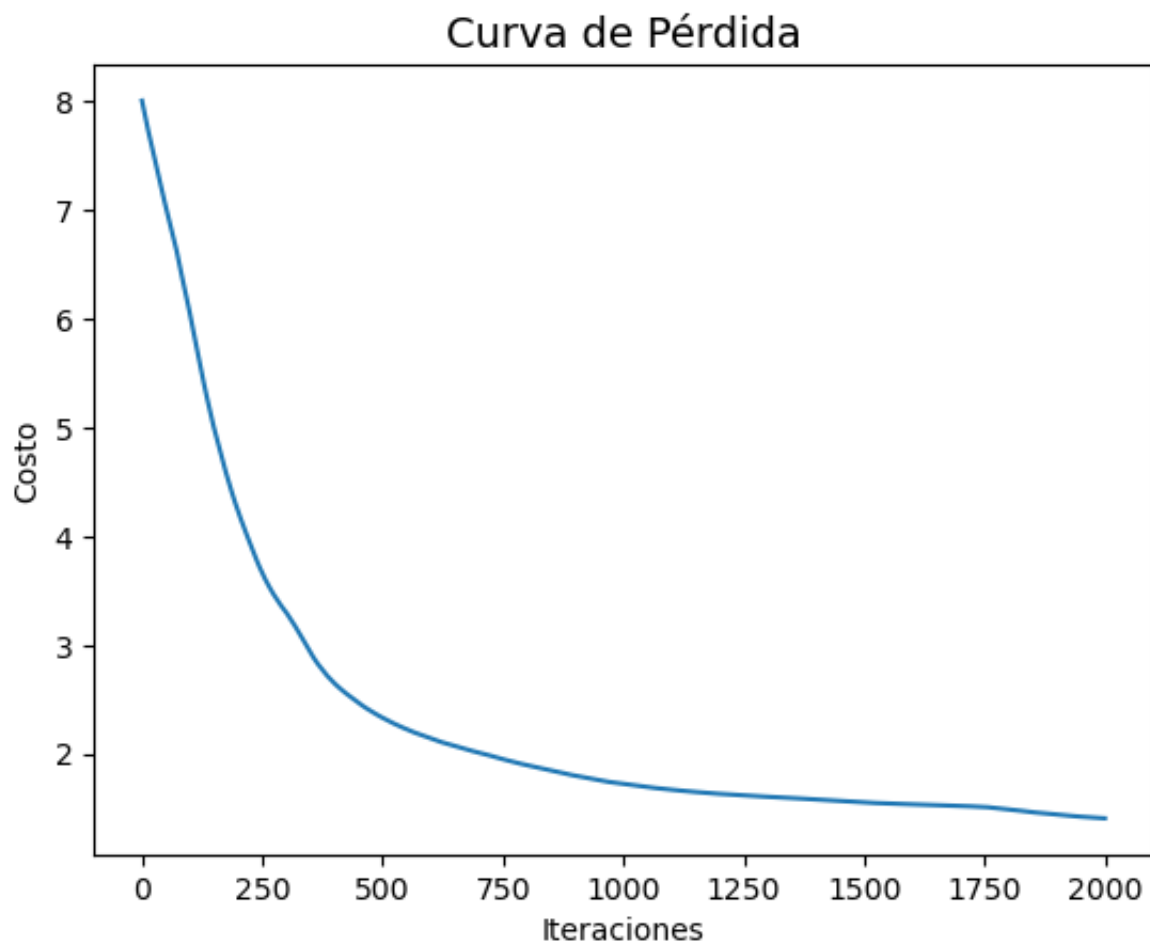
Puntuación de varianza explicada. La mejor puntuación posible es 1.0, los valores
 print('Varianza explicada: {:.2f}'.format(explained_variance_score(y_test, y_pred))

```

r2: 0.67
Error cuadrático promedio: 3.18
Varianza explicada: 0.67

```

```
# Vea qué tan bien fue el entrenamiento del modelo
plt.plot(mlp_reg.loss_curve_)
plt.title("Curva de Pérdida", fontsize=14)
plt.xlabel('Iteraciones')
plt.ylabel('Costo')
plt.show()
```



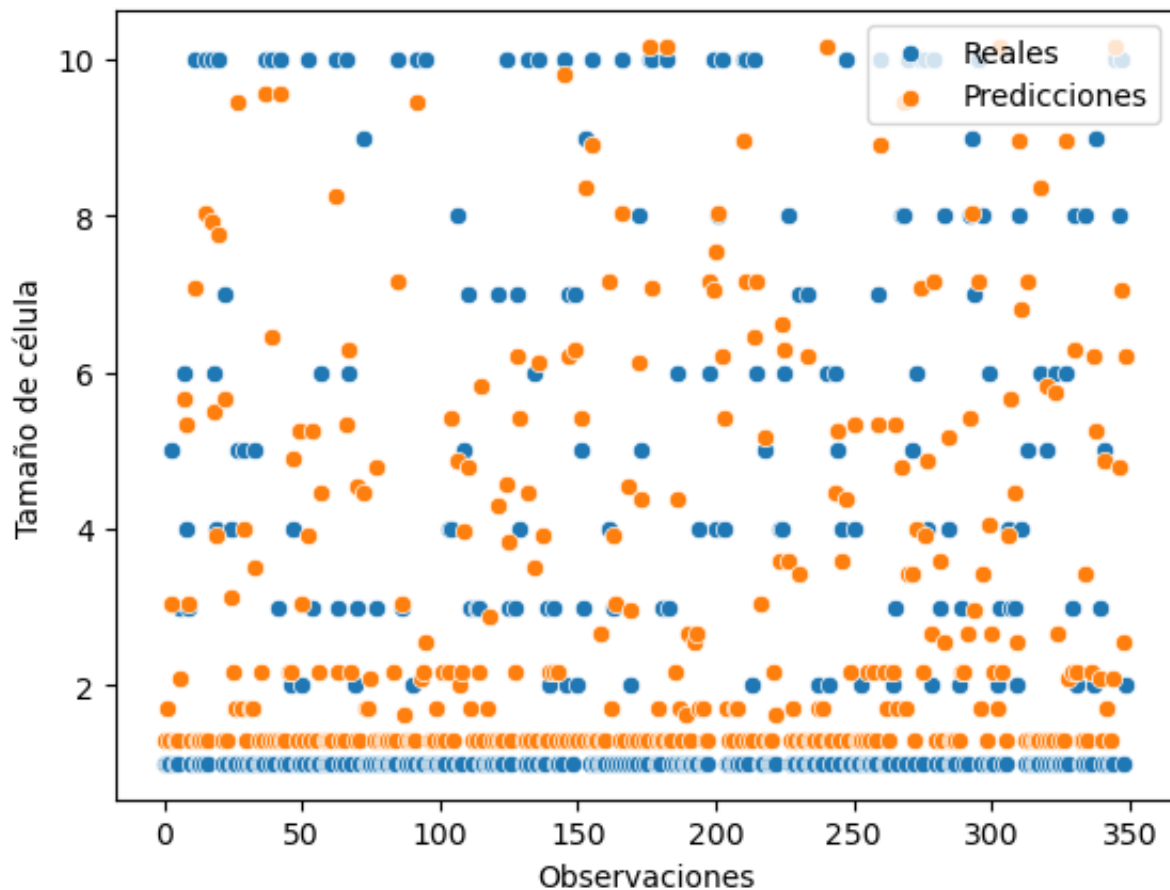
```
# Vea qué tan bien fue la prueba del modelo
regression_res_df = pd.DataFrame({'Reales': y_test, 'Predicciones': y_pred})
regression_res_df.head()
```

	Reales	Predicciones
158	1	1.292195
499	1	1.695805
396	1	1.292195
155	5	3.042488
321	1	1.292195

```
range(len(regression_res_df))
```

```
range(0, 350)
```

```
sn.scatterplot(x = list(range(len(regression_res_df))), y = 'Reales', data=regression_res_df)
sn.scatterplot(x = list(range(len(regression_res_df))), y = 'Predicciones', data=regression_res_df)
plt.xlabel('Observaciones')
plt.ylabel('Tamaño de célula')
plt.show()
```



Definición de la búsqueda de hiper-parámetros

Se usan los mismos valores de búsqueda declarados dentro de param_grid.

▼ Ejecución de la búsqueda de hiper-parámetros

```

grid_reg = GridSearchCV(
    # El modelo que estamos entrenando, en este caso un clasificador de perceptrone
    mlp_reg,
    # Los diferentes hiperparámetros que queremos entrenar
    param_grid,
    # Número de procesos que se ejecutarán en paralelo. Si se establece en -1, se u
    n_jobs = -1,
    # Número de veces que se repite la validación cruzada con diferentes conjuntos
    cv = 5
)

```

```

# Ajustar el modelo a los datos de entrenamiento (puede demorar)
grid_reg.fit(X_train_scaled, y_train)

```

```

GridSearchCV(cv=5, estimator=MLPRegressor(hidden_layer_sizes=2,
max_iter=2000),
            n_jobs=-1,
            param_grid={'activation': ['tanh', 'relu'],
                        'alpha': [0.0001, 0.05],
                        'hidden_layer_sizes': [(150, 100, 50), (120, 80),
100],
                        'learning_rate': ['constant', 'adaptive'],
                        'max_iter': [600, 1200, 1800],
                        'solver': ['sgd', 'adam']})

```

```

# Consigue los mejores valores para los hiper-parámetros entrenados
print('Los mejores valores de hiperparámetros son:\n{}'.format(grid_reg.best_params_))

```

```

Los mejores valores de hiperparámetros son:
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (150, 100, 50),

```

```

# Guarde los resultados como Pandas DataFrame para interactuar fácilmente con los c
grid_results_reg = pd.DataFrame(grid_reg.cv_results_)

```

```

grid_results_reg.sort_values('rank_test_score')

```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_activa
79	3.821337	0.532401	0.008079	0.010830	
11	7.688618	2.248379	0.007765	0.009080	

57	4.328221	0.513115	0.002117	0.000477
85	3.083079	0.454499	0.001853	0.000652
107	1.950336	0.627194	0.001107	0.000204
...
28	1.319313	0.877090	0.001433	0.000390
66	1.067453	0.386870	0.000705	0.000402
96	0.822446	0.385007	0.001318	0.000412
70	1.034798	1.001015	0.002127	0.001277
62	1.167524	0.719378	0.001307	0.000405

144 rows x 19 columns

```
def coder(classd):
    if classd == 'benign':
        return 0
    elif classd == 'malignant':
        return 1

# applying the coder to the feature of interest
dataset['classd'] = dataset['classd'].apply(coder)

dataset.groupby('classd').size()

Series([], dtype: int64)
```

▼ [Entrenamiento del mejor modelo](#)

```
# Definición de un clasificador de perceptrones múltiples.
# Usa los datos óptimos encontrados en la sintonización de hiperparámetros.
mlp_reg_opt = MLPRegressor(
    activation = 'relu',
    alpha = 0.0001,
    hidden_layer_sizes = (120, 80),
    learning_rate = 'constant',
    max_iter = 600,
    solver = 'adam'
)

# Entrenar el modelo
mlp_reg_opt.fit(X_train_scaled, y_train)

MLPRegressor(hidden_layer_sizes=(120, 80), max_iter=600)
```

▼ [Prueba del mejor modelo](#)

```
# Pasar el conjunto de datos que hemos preparado para las pruebas
y_pred = mlp_reg.predict(X_test_scaled)
y_pred

array([ 1.29219537,  1.69580459,  1.29219537,  3.04248831,  1.29219537,
        1.29219537,  2.08886146,  5.6520992 ,  5.34279888,  3.04248831,
        1.29219537,  7.08253947,  1.29219537,  1.29219537,  1.29219537,
        8.03616632,  1.29219537,  7.91498074,  5.51274146,  3.91235861,
        7.76477876,  1.29219537,  5.6520992 ,  1.29219537,  3.12624486,
```


2.17261801,	1.69580459,	9.46640627,	1.69580459,	3.99611516,
1.29219537,	1.69580459,	1.69580459,	3.51930173,	1.29219537,
2.17261801,	1.29219537,	9.56645822,	1.29219537,	6.46360716,
1.29219537,	1.29219537,	9.56645822,	1.29219537,	1.29219537,
2.17261801,	2.17261801,	4.91218164,	1.29219537,	5.26248752,
3.04248831,	1.29219537,	3.91235861,	1.29219537,	5.25904233,
1.29219537,	2.17261801,	4.47292858,	1.29219537,	1.29219537,
1.29219537,	1.29219537,	8.26528663,	2.17261801,	1.29219537,
1.29219537,	5.34279888,	6.29642573,	2.17261801,	1.29219537,
4.55668513,	1.29219537,	4.47292858,	1.69580459,	1.69580459,
2.08886146,	1.29219537,	4.7822289 ,	1.29219537,	1.29219537,
1.29219537,	1.29219537,	1.29219537,	2.17261801,	1.29219537,
7.16629602,	3.04248831,	1.61204804,	1.29219537,	1.29219537,
1.29219537,	1.29219537,	9.46640627,	2.08886146,	2.17261801,
2.56567489,	1.29219537,	1.29219537,	1.30274772,	1.69580459,
1.29219537,	2.17261801,	1.29219537,	2.17261801,	5.42655543,
1.29219537,	4.86598546,	2.00510491,	2.17261801,	3.96131593,
4.7822289 ,	1.69580459,	1.29219537,	1.29219537,	2.17261801,
5.8196123 ,	1.29219537,	1.69580459,	2.87497521,	1.29219537,
1.29219537,	4.31162181,	1.29219537,	1.29219537,	4.56187575,
3.82860206,	1.29219537,	2.17261801,	6.21266918,	5.42655543,
1.29219537,	1.29219537,	4.47292858,	1.29219537,	3.51930173,
1.29219537,	6.12891263,	3.91235861,	1.29219537,	1.29219537,
2.17261801,	2.17261801,	1.29219537,	2.17261801,	1.29219537,
9.81671216,	1.29219537,	6.21266918,	1.29219537,	6.29642573,
1.29219537,	5.42655543,	1.29219537,	8.36533858,	1.29219537,
8.90603662,	1.29219537,	1.29219537,	2.64943144,	1.29219537,
1.29219537,	7.16629602,	1.69580459,	3.91235861,	3.04248831,
1.29219537,	8.03616632,	1.29219537,	4.55668513,	2.95873176,
1.29219537,	1.29219537,	6.12891263,	4.38917203,	1.29219537,
1.29219537,	10.16701804,	7.08253947,	1.29219537,	1.69580459,
1.29219537,	1.29219537,	10.16701804,	1.29219537,	1.29219537,
2.17261801,	4.38917203,	1.69580459,	1.29219537,	1.61204804,
2.64943144,	1.29219537,	2.56567489,	2.64943144,	1.69580459,
1.69580459,	1.29219537,	1.29219537,	7.16421893,	7.06416699,
7.5593529 ,	8.03616632,	6.21266918,	5.42655543,	1.69580459,
1.29219537,	1.29219537,	1.69580459,	1.69580459,	1.29219537,
8.9658984 ,	7.16421893,	1.29219537,	1.29219537,	6.46360716,
7.16629602,	3.04248831,	1.29219537,	5.16243557,	1.29219537,
1.29219537,	2.17261801,	1.61204804,	3.60305828,	6.60572605,
6.29642573,	3.60305828,	1.29219537,	1.69580459,	1.29219537,
3.43554518,	1.29219537,	1.29219537,	6.21266918,	1.29219537,
1.29219537,	1.29219537,	1.69580459,	1.29219537,	1.69580459,
10.16701804,	1.29219537,	1.29219537,	4.47292858,	5.26248752,
1.29219537,	3.60305828,	4.38917203,	1.29219537,	2.17261801,
5.34279888,	1.29219537,	1.29219537,	1.29219537,	1.29219537,
2.17261801,	1.29219537,	2.17261801,	1.29219537,	5.34279888,
8.90603662,	2.17261801,	1.69580459,	1.29219537,	2.17261801,
5.34279888,	1.69580459,	4.7822289 ,	9.46640627,	1.69580459,
3.43554518,	3.43554518,	1.29219537,	3.99611516,	7.08253947,
2.17261801,	3.91235861,	4.86598546,	2.64943144,	7.16421893,
1.29219537,	3.60305828,	1.29219537,	2.56567489,	5.16243557,
1.29219537,	1.29219537,	1.29219537,	1.29219537,	2.17261801,
2.17261801,	2.64943144,	5.42655543,	8.03616632,	2.95873176,

▼ Evaluación del mejor modelo

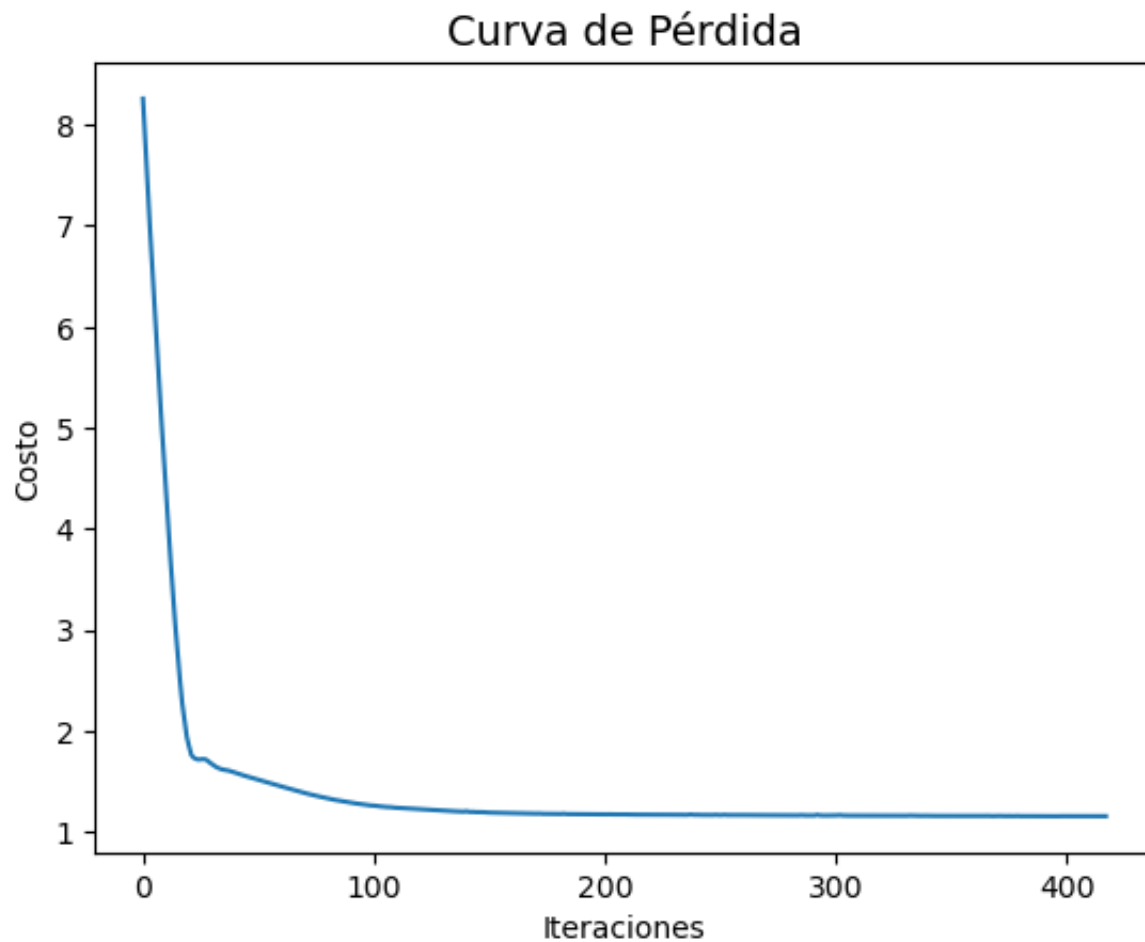
```
# Prueba de r2. La mejor puntuación posible es 1.0 y puede ser negativa (porque el
print('r2: {:.2f}'.format(r2_score(y_test, y_pred)))

# Error cuadrático promedio. La mejor puntuación posible es 0.0
print('Error cuadrático promedio: {:.2f}'.format(mean_squared_error(y_test, y_pred))

# Puntuación de varianza explicada. La mejor puntuación posible es 1.0, los valores
print('Varianza explicada: {:.2f}'.format(explained_variance_score(y_test, y_pred))

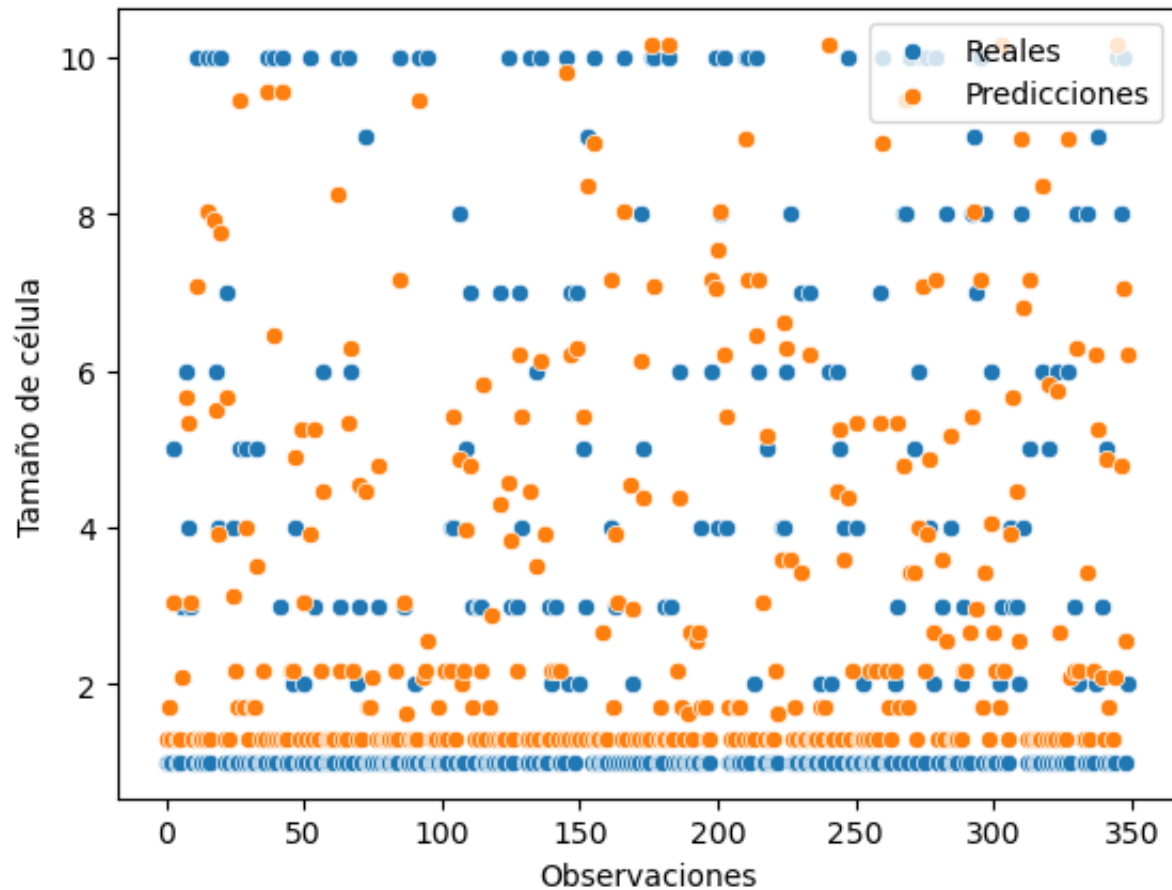
r2: 0.67
Error cuadrático promedio: 3.18
Varianza explicada: 0.67
```

```
# Vea qué tan bien fue el entrenamiento del modelo
plt.plot(mlp_reg_opt.loss_curve_)
plt.title("Curva de Pérdida", fontsize=14)
plt.xlabel('Iteraciones')
plt.ylabel('Costo')
plt.show()
```



```
# Veamos qué tan bien fue la prueba del modelo
regression_res_df_opt = pd.DataFrame({'Reales': y_test, 'Predicciones': y_pred})

sns.scatterplot(x = list(range(len(regression_res_df))), y = 'Reales', data=regression_res_df)
sns.scatterplot(x = list(range(len(regression_res_df))), y = 'Predicciones', data=regression_res_df)
plt.xlabel('Observaciones')
plt.ylabel('Tamaño de célula')
plt.show()
```



Colab paid products - [Cancel contracts here](#)

