Librerías

In [ ]:
```
# Cargamos las librerías necesarias

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#!pip install shap
```

1. Carga y Visualización Inicial del Dataset

Propósito: Cargar el conjunto de datos data.csv y visualizar las primeras 10 filas para entender su estructura y contenido.

Interpretación: Permite verificar que los datos se han cargado correctamente y proporciona una visión preliminar de las columnas y algunos valores iniciales.

In [ ]:
```
# Cargar el conjunto de datos del Titanic del PC
# titanic_df = pd.read_csv('titanic.csv')
# titanic_df = sns.load_dataset("titanic")

# Colab
# Montar Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Ruta del archivo CSV
file_path = '/content/data.csv'

# Leer el archivo CSV
data = pd.read_csv(file_path)

# Ver las primeras filas del conjunto de datos
print(data.head())
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
e.mount("/content/drive", force_remount=True).
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622
1  ...          23.41           158.80      1956.0            0.1238
2  ...          25.53           152.50      1709.0            0.1444
3  ...          26.50            98.87       567.7            0.2098
4  ...          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

2. Estadísticas Descriptivas

Propósito: Calcular estadísticas descriptivas de las variables numéricas y categóricas.

Interpretación: Proporciona un resumen de las principales métricas (como media, desviación estándar, mínimos, máximos, etc.) de las características numéricas y una vista general de las variables categóricas.

In [ ]:
```python
# Estadísticas descriptivas
print(data.describe())

# Estadísticas de variables categóricas
print(data.describe(include=['object']))
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
count	569.000000	569.000000	569.000000	569.000000
mean	0.096360	0.104341	0.088799	0.048919
std	0.014064	0.052813	0.079720	0.038803
min	0.052630	0.019380	0.000000	0.000000
25%	0.086370	0.064920	0.029560	0.020310
50%	0.095870	0.092630	0.061540	0.033500
75%	0.105300	0.130400	0.130700	0.074000
max	0.163400	0.345400	0.426800	0.201200

	symmetry_mean	...	texture_worst	perimeter_worst	area_worst
count	569.000000	...	569.000000	569.000000	569.000000
mean	0.181162	...	25.677223	107.261213	880.583128
std	0.027414	...	6.146258	33.602542	569.356993
min	0.106000	...	12.020000	50.410000	185.200000
25%	0.161900	...	21.080000	84.110000	515.300000
50%	0.179200	...	25.410000	97.660000	686.500000
75%	0.195700	...	29.720000	125.400000	1084.000000
max	0.304000	...	49.540000	251.200000	4254.000000

	smoothness_worst	compactness_worst	concavity_worst
count	569.000000	569.000000	569.000000
mean	0.132369	0.254265	0.272188
std	0.022832	0.157336	0.208624
min	0.071170	0.027290	0.000000
25%	0.116600	0.147200	0.114500
50%	0.131300	0.211900	0.226700
75%	0.146000	0.339100	0.382900
max	0.222600	1.058000	1.252000

	concave points_worst	symmetry_worst	fractal_dimension_worst
count	569.000000	569.000000	569.000000
mean	0.114606	0.290076	0.083946
std	0.065732	0.061867	0.018061
min	0.000000	0.156500	0.055040
25%	0.064930	0.250400	0.071460
50%	0.099930	0.282200	0.080040
75%	0.161400	0.317900	0.092080
max	0.291000	0.663800	0.207500

	Unnamed: 32
count	0.0
mean	NaN
std	NaN
min	NaN
25%	NaN

```
50%            NaN
75%            NaN
max            NaN

[8 rows x 32 columns]
       diagnosis
count        569
unique         2
top            B
freq         357
```

3. Preprocesamiento de Datos

Propósito: Convertir la variable categórica diagnosis a numérica (1 para maligno y 0 para benigno) y descartar la columna de identificación id que no aporta información relevante para el análisis.

Interpretación: Facilita el análisis posterior, ya que la variable objetivo diagnosis está ahora en formato numérico, y elimina columnas innecesarias.

In [ ]:
```
# Sustituir 'M' por 1 y 'B' por 0
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})

# Ver las primeras filas del conjunto de datos
print(data.head(10))

# Descartar la columna del identificador
data = data.drop(columns=['id'])

# Ver las primeras filas del conjunto de datos
print(data.head(10))
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	1	17.99	10.38	122.80	1001.0	
1	842517	1	20.57	17.77	132.90	1326.0	
2	84300903	1	19.69	21.25	130.00	1203.0	
3	84348301	1	11.42	20.38	77.58	386.1	
4	84358402	1	20.29	14.34	135.10	1297.0	
5	843786	1	12.45	15.70	82.57	477.1	
6	844359	1	18.25	19.98	119.60	1040.0	
7	84458202	1	13.71	20.83	90.20	577.9	
8	844981	1	13.00	21.82	87.50	519.8	
9	84501001	1	12.46	24.04	83.97	475.9	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.30010	0.14710	
1	0.08474	0.07864	0.08690	0.07017	
2	0.10960	0.15990	0.19740	0.12790	
3	0.14250	0.28390	0.24140	0.10520	
4	0.10030	0.13280	0.19800	0.10430	
5	0.12780	0.17000	0.15780	0.08089	
6	0.09463	0.10900	0.11270	0.07400	
7	0.11890	0.16450	0.09366	0.05985	
8	0.12730	0.19320	0.18590	0.09353	
9	0.11860	0.23960	0.22730	0.08543	

	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	\
0	...	17.33	184.60	2019.0	0.1622	
1	...	23.41	158.80	1956.0	0.1238	
2	...	25.53	152.50	1709.0	0.1444	
3	...	26.50	98.87	567.7	0.2098	
4	...	16.67	152.20	1575.0	0.1374	
5	...	23.75	103.40	741.6	0.1791	
6	...	27.66	153.20	1606.0	0.1442	
7	...	28.14	110.60	897.0	0.1654	
8	...	30.73	106.20	739.3	0.1703	
9	...	40.68	97.65	711.4	0.1853	

	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	\
0	0.6656	0.7119	0.2654	0.4601	
1	0.1866	0.2416	0.1860	0.2750	
2	0.4245	0.4504	0.2430	0.3613	
3	0.8663	0.6869	0.2575	0.6638	
4	0.2050	0.4000	0.1625	0.2364	
5	0.5249	0.5355	0.1741	0.3985	
6	0.2576	0.3784	0.1932	0.3063	
7	0.3682	0.2678	0.1556	0.3196	
8	0.5401	0.5390	0.2060	0.4378	
9	1.0580	1.1050	0.2210	0.4366	

	fractal_dimension_worst	Unnamed: 32
0	0.11890	NaN
1	0.08902	NaN
2	0.08758	NaN
3	0.17300	NaN
4	0.07678	NaN
5	0.12440	NaN
6	0.08368	NaN

```
7                 0.11510          NaN
8                 0.10720          NaN
9                 0.20750          NaN

[10 rows x 33 columns]
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38          122.80     1001.0
1          1        20.57         17.77          132.90     1326.0
2          1        19.69         21.25          130.00     1203.0
3          1        11.42         20.38           77.58      386.1
4          1        20.29         14.34          135.10     1297.0
5          1        12.45         15.70           82.57      477.1
6          1        18.25         19.98          119.60     1040.0
7          1        13.71         20.83           90.20      577.9
8          1        13.00         21.82           87.50      519.8
9          1        12.46         24.04           83.97      475.9

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760         0.30010              0.14710
1          0.08474           0.07864         0.08690              0.07017
2          0.10960           0.15990         0.19740              0.12790
3          0.14250           0.28390         0.24140              0.10520
4          0.10030           0.13280         0.19800              0.10430
5          0.12780           0.17000         0.15780              0.08089
6          0.09463           0.10900         0.11270              0.07400
7          0.11890           0.16450         0.09366              0.05985
8          0.12730           0.19320         0.18590              0.09353
9          0.11860           0.23960         0.22730              0.08543

   symmetry_mean  ...  texture_worst  perimeter_worst  area_worst  \
0         0.2419  ...          17.33           184.60      2019.0
1         0.1812  ...          23.41           158.80      1956.0
2         0.2069  ...          25.53           152.50      1709.0
3         0.2597  ...          26.50            98.87       567.7
4         0.1809  ...          16.67           152.20      1575.0
5         0.2087  ...          23.75           103.40       741.6
6         0.1794  ...          27.66           153.20      1606.0
7         0.2196  ...          28.14           110.60       897.0
8         0.2350  ...          30.73           106.20       739.3
9         0.2030  ...          40.68            97.65       711.4

   smoothness_worst  compactness_worst  concavity_worst  concave points_worst  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625
5            0.1791             0.5249           0.5355                0.1741
6            0.1442             0.2576           0.3784                0.1932
7            0.1654             0.3682           0.2678                0.1556
8            0.1703             0.5401           0.5390                0.2060
9            0.1853             1.0580           1.1050                0.2210

   symmetry_worst  fractal_dimension_worst  Unnamed: 32
0          0.4601                  0.11890          NaN
1          0.2750                  0.08902          NaN
```

```
2          0.3613                  0.08758          NaN
3          0.6638                  0.17300          NaN
4          0.2364                  0.07678          NaN
5          0.3985                  0.12440          NaN
6          0.3063                  0.08368          NaN
7          0.3196                  0.11510          NaN
8          0.4378                  0.10720          NaN
9          0.4366                  0.20750          NaN

[10 rows x 32 columns]
```

4. Verificación de Datos Faltantes

Propósito: Verificar si hay valores faltantes en el conjunto de datos y eliminar la columna Unnamed: 32 que probablemente no contiene información útil.

Interpretación: Garantiza que no hay valores faltantes en el dataset, lo que es crucial para un análisis preciso y confiable.

In [ ]:
```python
#Verifica los datos faltantes de los dataset
print('Datos faltantes:')
print(pd.isnull(data).sum())

data = data.drop(columns=['Unnamed: 32'])

#Verifica los datos faltantes de los dataset
print('Datos faltantes:')
print(pd.isnull(data).sum())

# Estadísticas descriptivas
statistics = data.describe()
```

```
Datos faltantes:
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
Unnamed: 32              569
dtype: int64
Datos faltantes:
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
```

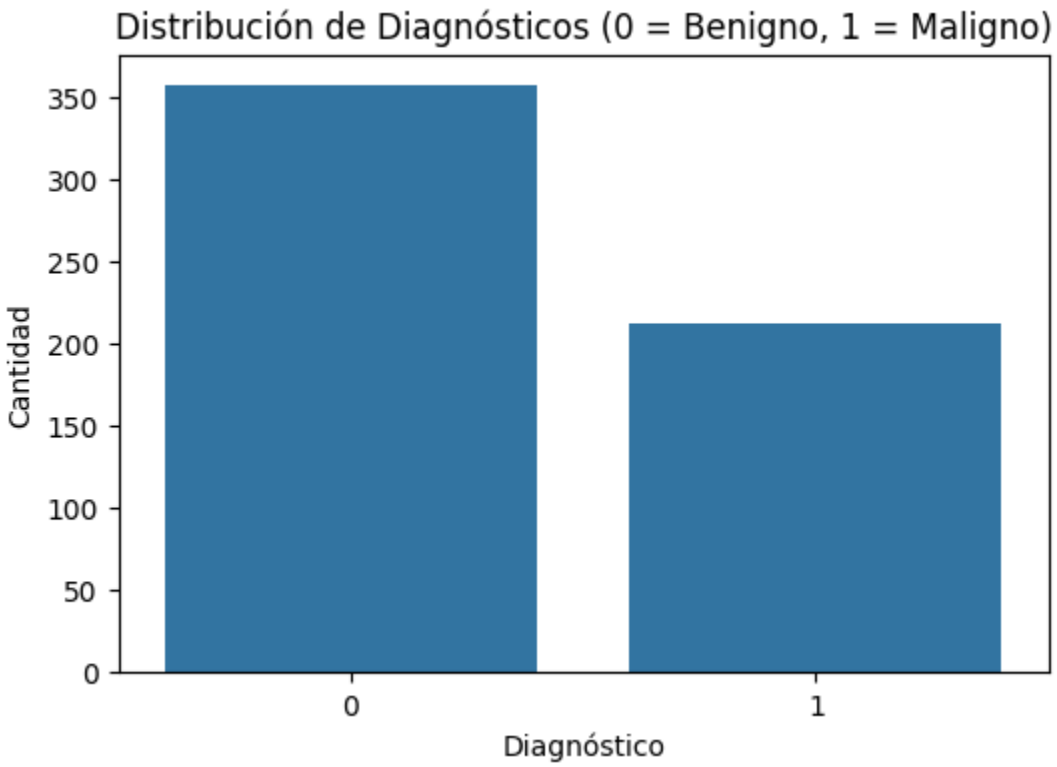
```
radius_worst              0
texture_worst             0
perimeter_worst           0
area_worst                0
smoothness_worst          0
compactness_worst         0
concavity_worst           0
concave points_worst      0
symmetry_worst            0
fractal_dimension_worst   0
dtype: int64
```

5. Visualización de la Distribución de Diagnósticos

Propósito: Visualizar la distribución de casos benignos y malignos.

Interpretación: Muestra la proporción de casos benignos y malignos en el conjunto de datos, indicando si el dataset está equilibrado o no.

In [ ]:
```
# Visualización de la distribución de la variable 'diagnosis'
plt.figure(figsize=(6, 4))
sns.countplot(x='diagnosis', data=data)
plt.title('Distribución de Diagnósticos (0 = Benigno, 1 = Maligno)')
plt.xlabel('Diagnóstico')
plt.ylabel('Cantidad')
plt.show()
```

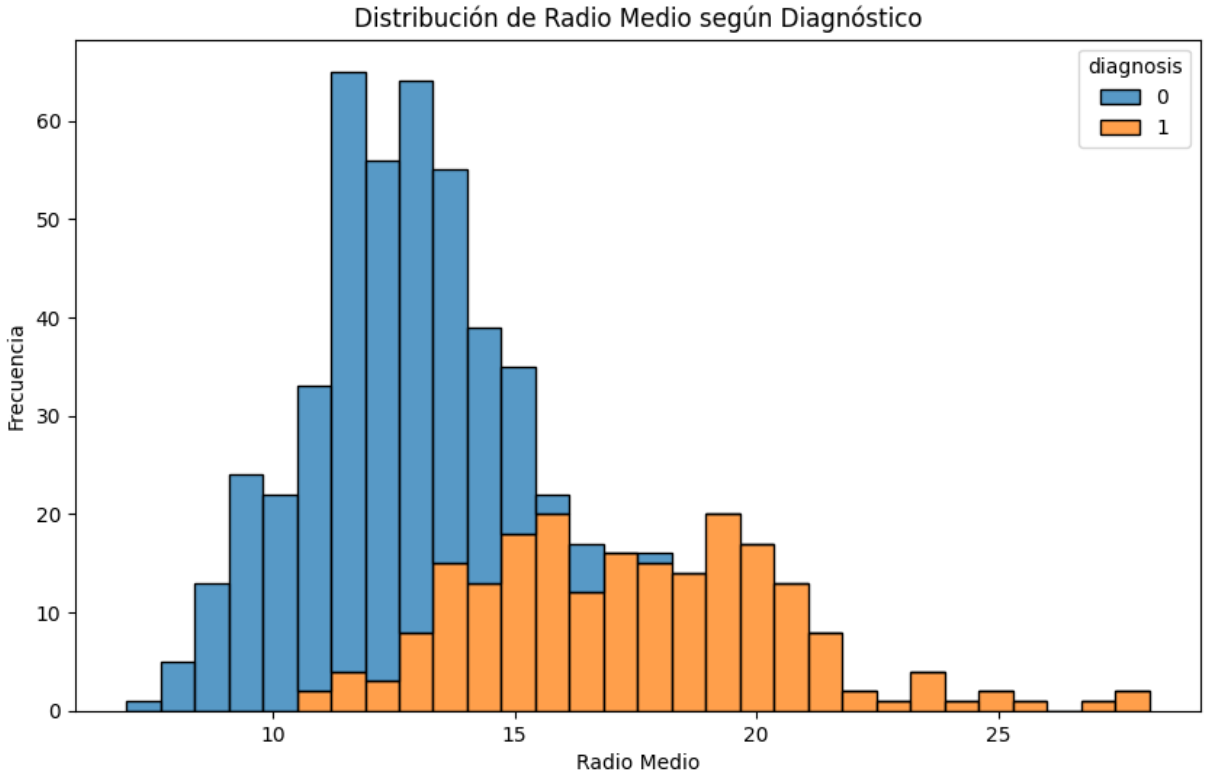


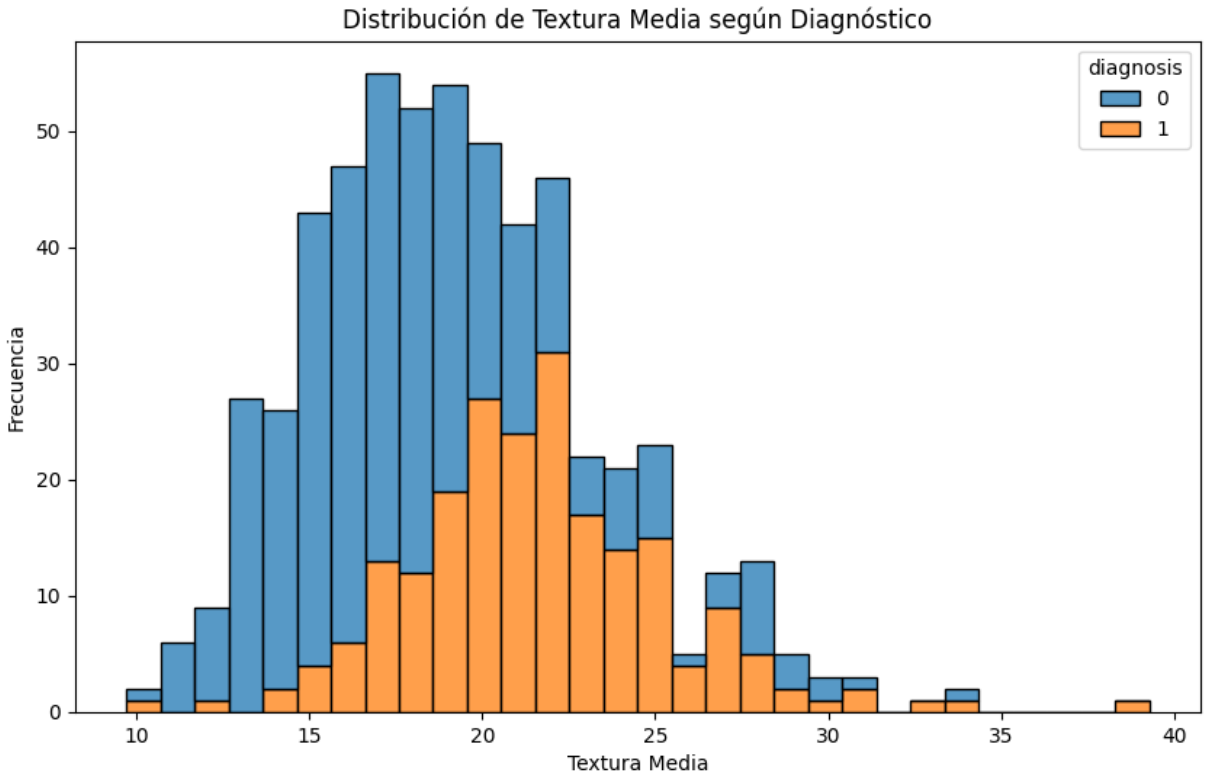
6. Distribuciones de Características Específicas por Diagnóstico

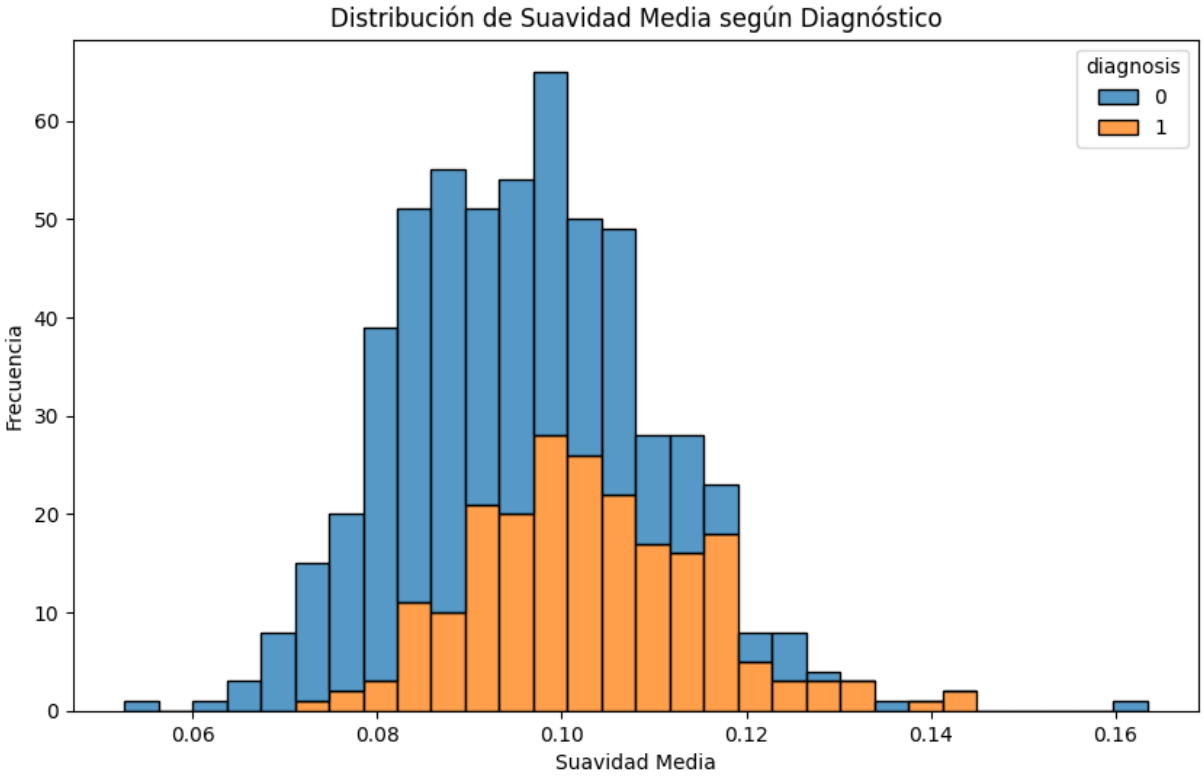
Propósito: Visualizar la distribución de características específicas (radius_mean, texture_mean, smoothness_mean) según el diagnóstico.

Interpretación: Permite observar cómo estas características varían entre los tumores benignos y malignos, proporcionando pistas sobre su relevancia para la predicción del diagnóstico.

In [ ]:
```
# Visualización de la Distribución de Radio Medio según Diagnóstico
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='radius_mean', hue='diagnosis', multiple='stack', bins=30
plt.title('Distribución de Radio Medio según Diagnóstico')
plt.xlabel('Radio Medio')
plt.ylabel('Frecuencia')
plt.show()

# Visualización de la Distribución de Textura Media según Diagnóstico
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='texture_mean', hue='diagnosis', multiple='stack', bins=3
plt.title('Distribución de Textura Media según Diagnóstico')
plt.xlabel('Textura Media')
plt.ylabel('Frecuencia')
plt.show()

# Visualización de la Distribución de Suavidad Media según Diagnóstico
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='smoothness_mean', hue='diagnosis', multiple='stack', bin
plt.title('Distribución de Suavidad Media según Diagnóstico')
plt.xlabel('Suavidad Media')
plt.ylabel('Frecuencia')
plt.show()
```


Distribución de Radio Medio según Diagnóstico
diagnosis
0
1
Frecuencia
Radio Medio
0
10
20
30
40
50
60
10
15
20
25


Distribución de Textura Media según Diagnóstico
diagnosis
0
1
Frecuencia
Textura Media
0
10
20
30
40
50
10
15
20
25
30
35
40



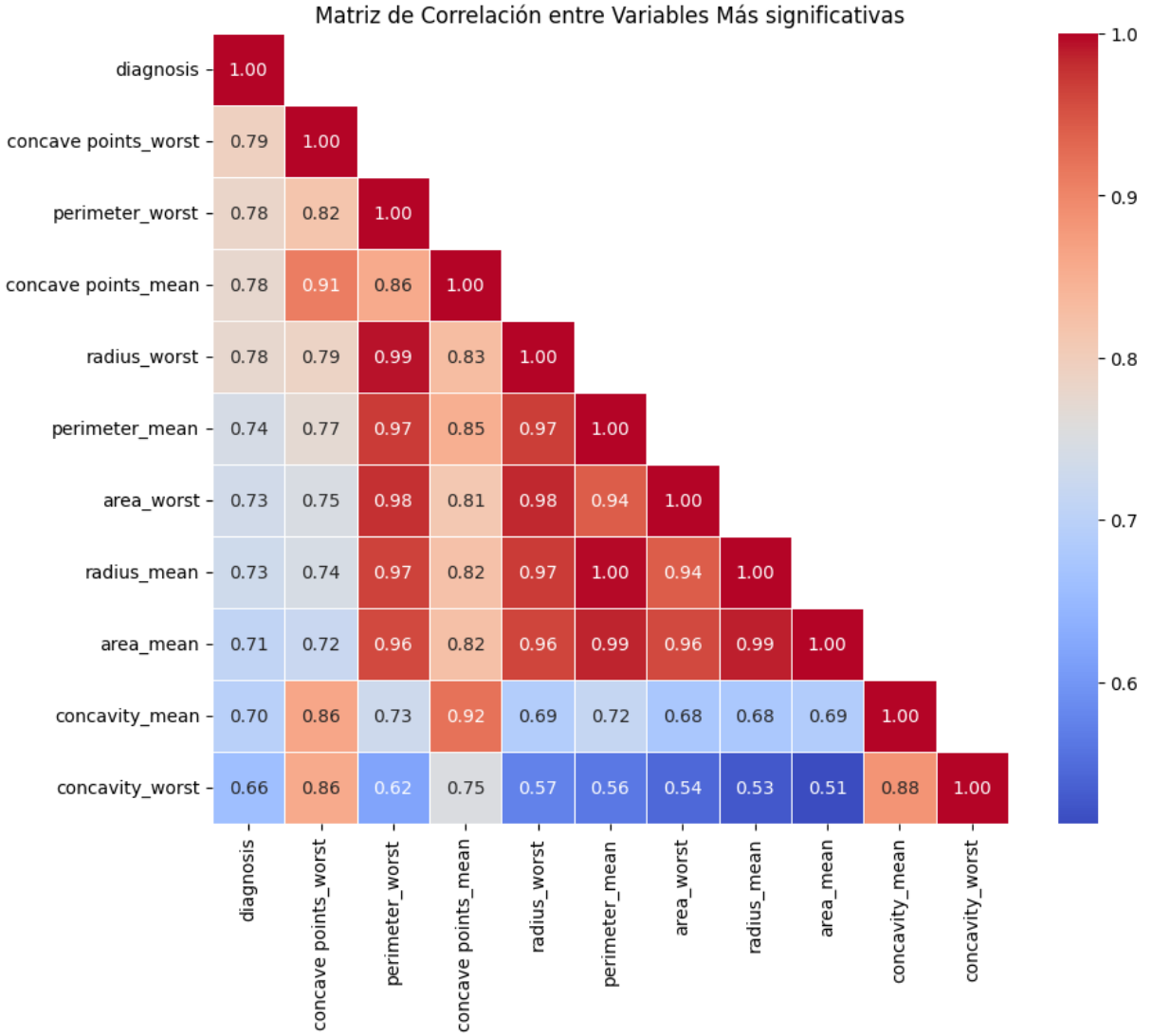
7. Análisis de Correlación

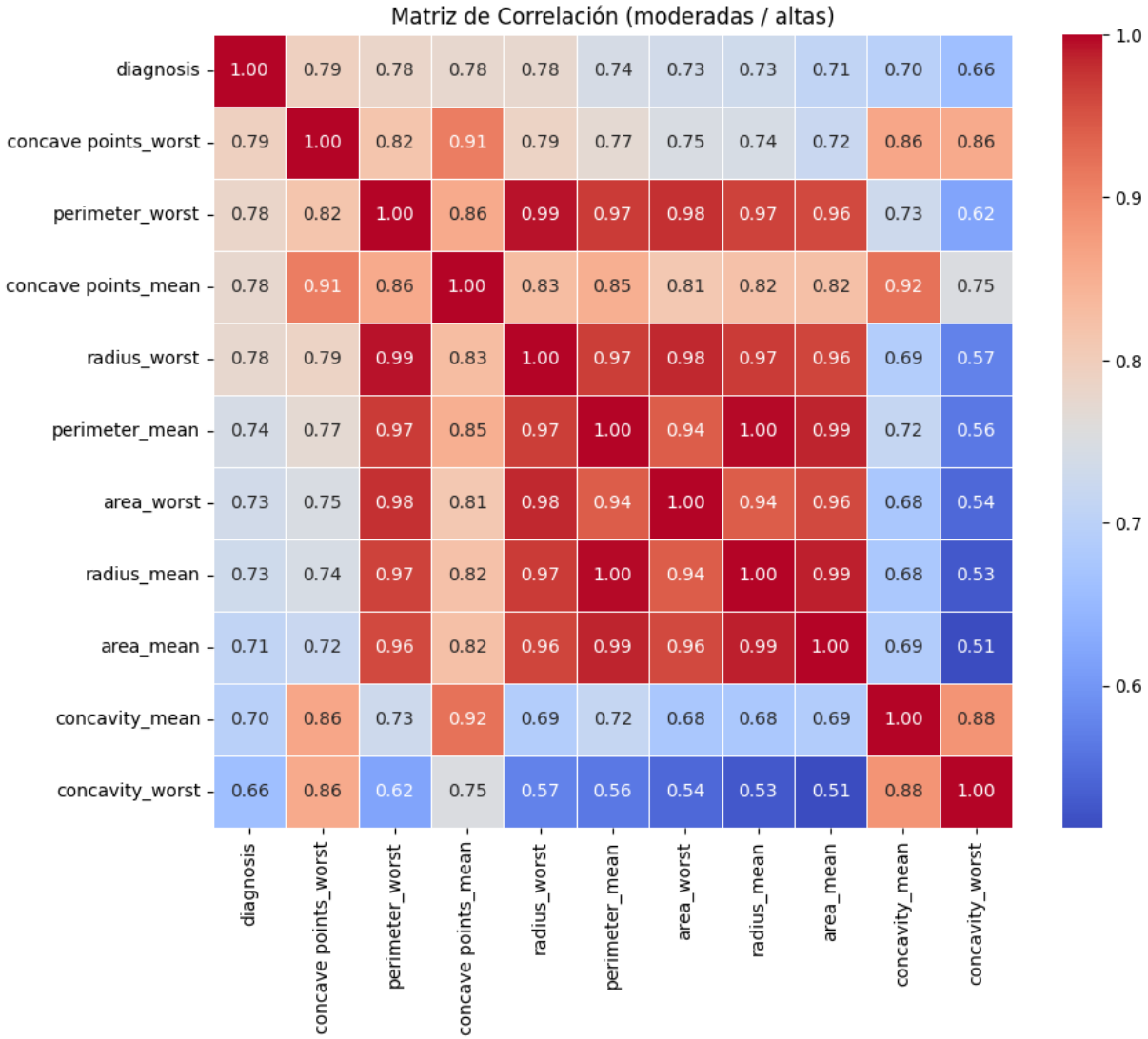
Propósito: Calcular y visualizar las correlaciones entre las características más significativas y el diagnóstico.

Interpretación: Las matrices de correlación ayudan a identificar cuáles características están más fuertemente relacionadas con el diagnóstico de cáncer, proporcionando información valiosa para la selección de características y el desarrollo de modelos predictivos.

In [ ]:
```
# Calcular la correlación de todas las variables con 'diagnosis'
correlation_with_target = data.corr()['diagnosis'].sort_values(ascending=False)

# Seleccionar las 10 variables más correlacionadas con 'diagnosis'
top_10_features = correlation_with_target.head(11).index.tolist()  # Incluye 'diagn
top_10_features

# Generar la matriz de correlación solo con las variables más significativas
top_10_corr_matrix = data[top_10_features].corr()
# crea una máscara para ocultar la parte superior de la matriz de correlación
# con k=0 no incluye la diagonal principal y con k=1 si
mask = np.triu(np.ones_like(top_10_corr_matrix, dtype=bool), k=1)

# Crear un mapa de calor de correlación
plt.figure(figsize=(10, 8))
sns.heatmap(top_10_corr_matrix, mask=mask, annot=True, fmt='.2f', cmap='coolwarm',
plt.title('Matriz de Correlación entre Variables Más significativas')
plt.show()
```

```
# Aplicar una máscara para mostrar solo correlaciones moderadas/altas mayores a 0.4
mask = np.abs(top_10_corr_matrix) < 0.4
top_10_corr_matrix[mask] = np.nan
# Crear un mapa de calor de correlación con valores significativos
plt.figure(figsize=(10, 8))
sns.heatmap(top_10_corr_matrix, mask=mask, annot=True, fmt='.2f', cmap='coolwarm',
plt.title('Matriz de Correlación (moderadas / altas)')
plt.show()
```





Principales Observaciones de la Correlación:

Variables Altamente Correlacionadas con diagnosis:

Radio Medio (radius_mean): Una alta correlación positiva con el diagnóstico indica que, a medida que aumenta el radio medio del tumor, es más probable que el tumor sea maligno.

** Perímetro Medio (perimeter_mean):** Similar al radio medio, el perímetro medio tiene una alta correlación positiva, sugiriendo que los tumores malignos tienden a tener perímetros más grandes.

Área Media (area_mean): Los tumores malignos tienden a tener áreas más grandes, como lo indica la alta correlación positiva.

Suavidad Media (smoothness_mean): Aunque con una menor correlación positiva, también sugiere que los tumores más suaves son más probablemente malignos.

Concavidad Media (concavity_mean) y Puntos Cóncavos Medios (concave

points_mean): Ambas características muestran alta correlación positiva con el diagnóstico maligno, lo que indica que los tumores malignos tienden a tener más concavidades y puntos cóncavos.

Variables con Correlación Negativa o Baja:

Fractal Dimension Mean (fractal_dimension_mean): Tiene una baja correlación con el diagnóstico, lo que sugiere que no es un buen predictor del tipo de tumor.

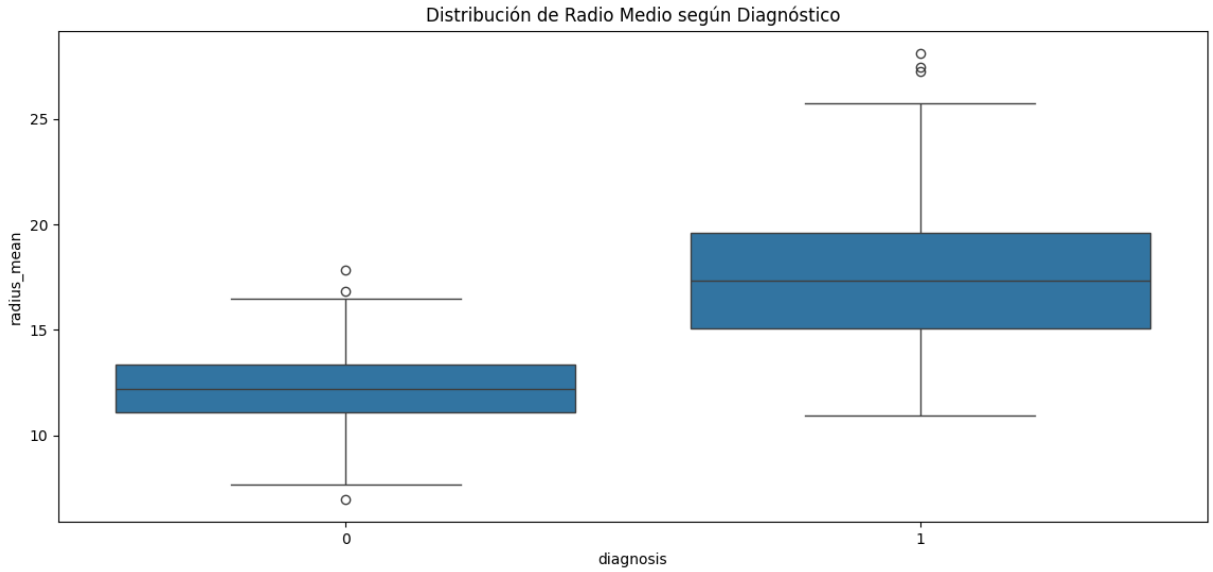
Symmetry Mean (symmetry_mean): También tiene una correlación baja con el diagnóstico, indicando que la simetría no varía significativamente entre tumores benignos y malignos.

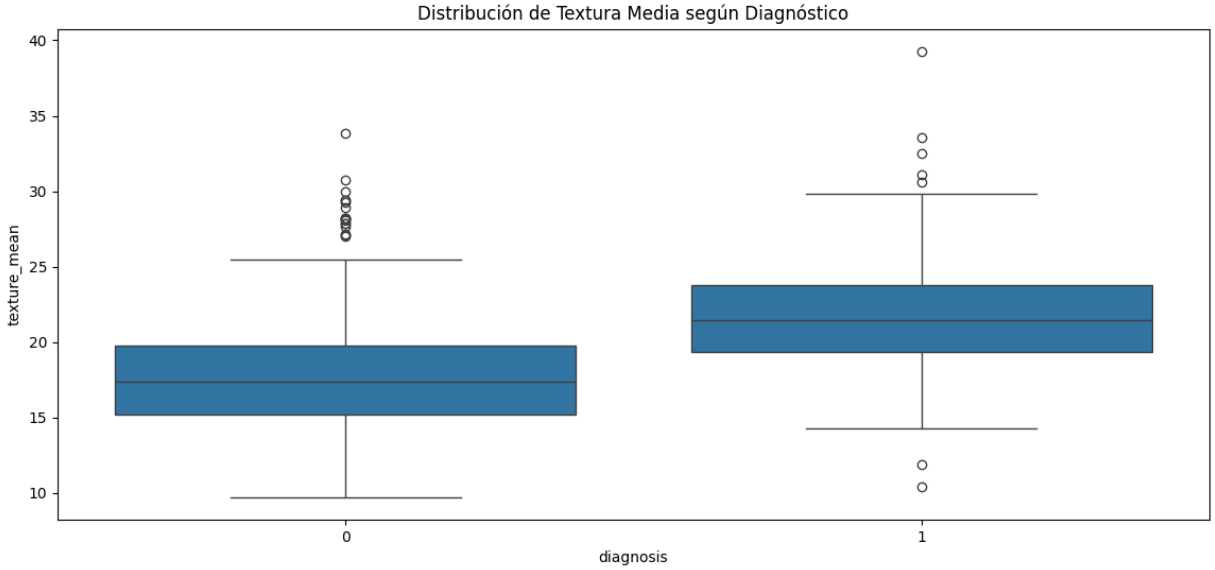
8. Visualización de Características Clave mediante Boxplots

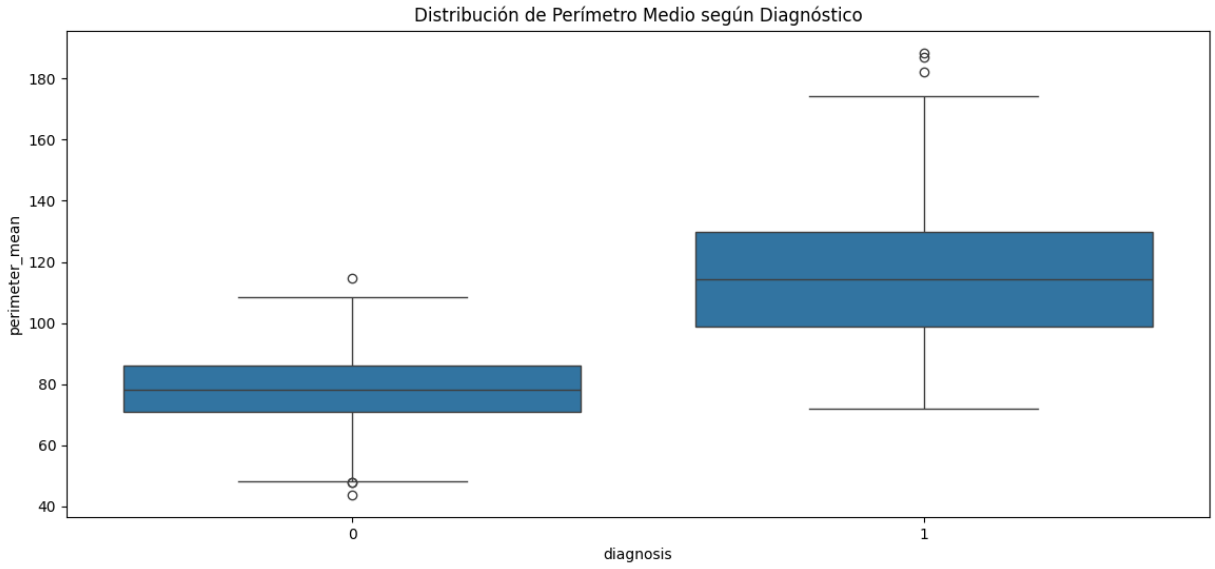
Propósito: Visualizar la distribución de características clave (radius_mean, texture_mean, perimeter_mean, area_mean) según el diagnóstico usando boxplots.

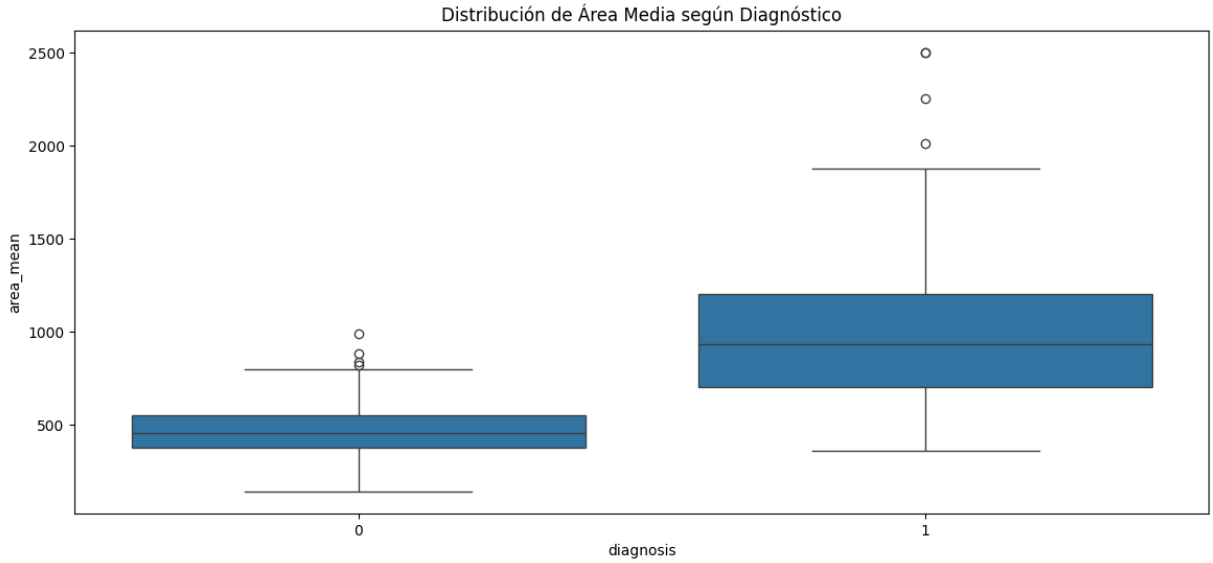
Interpretación: Los boxplots permiten comparar la distribución de estas características entre los diagnósticos benignos y malignos, mostrando si hay diferencias significativas que podrían ser útiles para la predicción del diagnóstico.

In [ ]:
```
# Visualización de algunas variables importantes

plt.figure(figsize=(14, 6))
sns.boxplot(x='diagnosis', y='radius_mean', data=data)
plt.title('Distribución de Radio Medio según Diagnóstico')
plt.show()


# Distribución
plt.figure(figsize=(14, 6))
sns.boxplot(x='diagnosis', y='texture_mean', data=data)
plt.title('Distribución de Textura Media según Diagnóstico')
plt.show()

plt.figure(figsize=(14, 6))
sns.boxplot(x='diagnosis', y='perimeter_mean', data=data)
plt.title('Distribución de Perímetro Medio según Diagnóstico')
plt.show()

plt.figure(figsize=(14, 6))
sns.boxplot(x='diagnosis', y='area_mean', data=data)
plt.title('Distribución de Área Media según Diagnóstico')
plt.show()
```


Distribución de Radio Medio según Diagnóstico
radius_mean
25
20
15
10
0
1
diagnosis


Distribución de Textura Media según Diagnóstico
texture_mean
40
35
30
25
20
15
10
0
1
diagnosis


Distribución de Perímetro Medio según Diagnóstico
perimeter_mean
180
160
140
120
100
80
60
40
0
1
diagnosis



Conclusión de EDA

Este análisis proporciona una comprensión detallada del dataset de cáncer de mama. A través de diversas técnicas de visualización y análisis estadístico, hemos identificado características clave que pueden ayudar en la predicción del diagnóstico de cáncer de mama. La correlación y los gráficos de distribución muestran que ciertas características tienen una relación significativa con el diagnóstico, lo cual es esencial para construir modelos predictivos eficaces.

MODELOS DE CLASIFICACIÓN

1. Regresión Logística

Librerías

In [ ]:
```
# Logistic regression for breast cancer

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
```

Carga de datos

In [ ]:
```
# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
```

```
y = dataset.target # 569x1
```

División datos en train y test

In [ ]:
```
# Divide el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Normalización Entrenamiento y Predicción

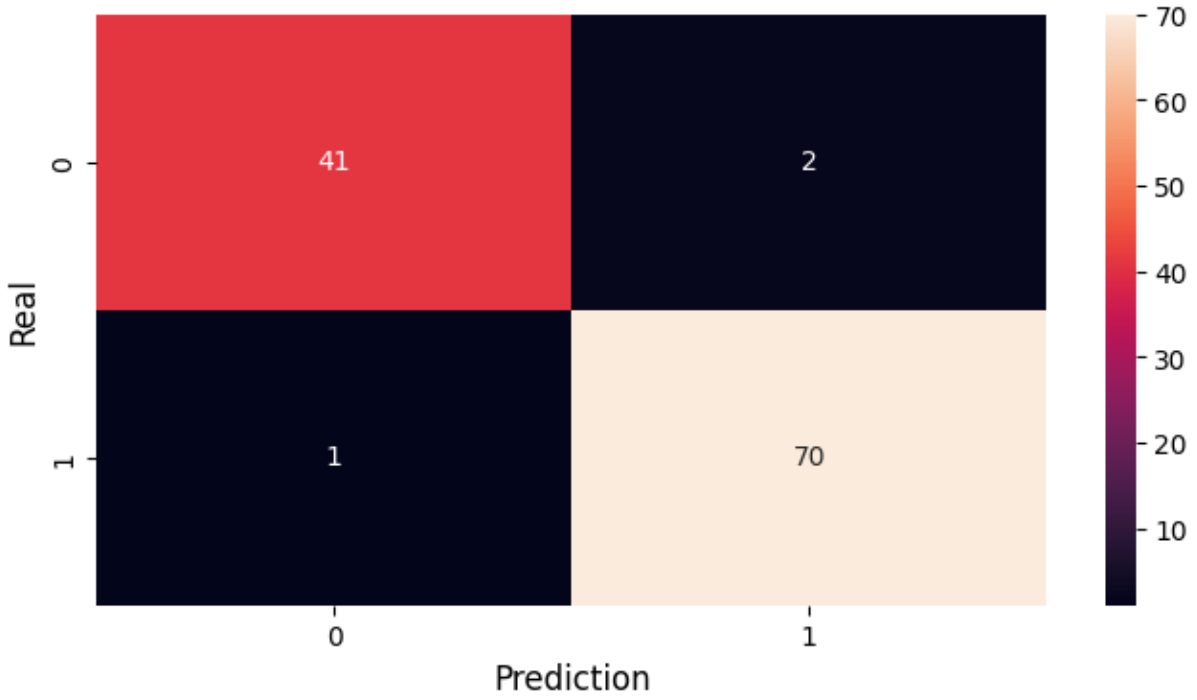
In [ ]:
```
# Normaliza los datos para que todas las características tengan una escala similar
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Crea y entrena el modelo de regresión logistica
model = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=100)
model.fit(X_train, y_train)

# Imprime los coeficientes y el intercepto del modelo entrenado
print("\nCoeficientes del modelo:")
print(model.coef_)
print("\nIntercepto del modelo:")
print(model.intercept_)


# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)


# Convierte las probabilidades en etiquetas binarias (0 o 1)
y_pred = (y_pred > 0.5)

# Muestra el informe de evaluación del modelo entrenado
print(classification_report(y_test, y_pred))
```

```
Coeficientes del modelo:
[[-0.42789615 -0.39391343 -0.38955025 -0.46431618 -0.06675416  0.54210625
  -0.79677127 -1.1170207   0.23571257  0.07670117 -1.27114722  0.18863977
  -0.60936581 -0.90979979 -0.31246106  0.68597229  0.18081531 -0.31769168
   0.49997976  0.61340541 -0.87861043 -1.3421883  -0.58755707 -0.84655924
  -0.54994459  0.00520705 -0.94571375 -0.77343621 -1.20853126 -0.1541604 ]]

Intercepto del modelo:
[0.44359695]
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

Matriz de confusión y Métricas de Evaluación

In [ ]:
```
# Matriz de confusión:
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
print("confusion matrix: \n", cm)
# gráfica cm
plt.figure(figsize = (8,4))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Prediction', fontsize = 12)
plt.ylabel('Real', fontsize = 12)
plt.show()

# Exactitud:
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("accuracy: ", acc)

# Sensibilidad:
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("recall: ", recall)

# Precisión:
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("precision: ", precision)

# Especificidad
# 'specificity' is just a special case of 'recall'.
# specificity is the recall of the negative class
specificity = recall_score(y_test, y_pred, pos_label=0)
print("specificity: ", specificity)

# Puntuación F1:
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("f1 score: ", f1)

# Área bajo la curva:
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, y_pred)
print("auc: ", auc)
```

```
confusion matrix:
 [[41  2]
 [ 1 70]]
```

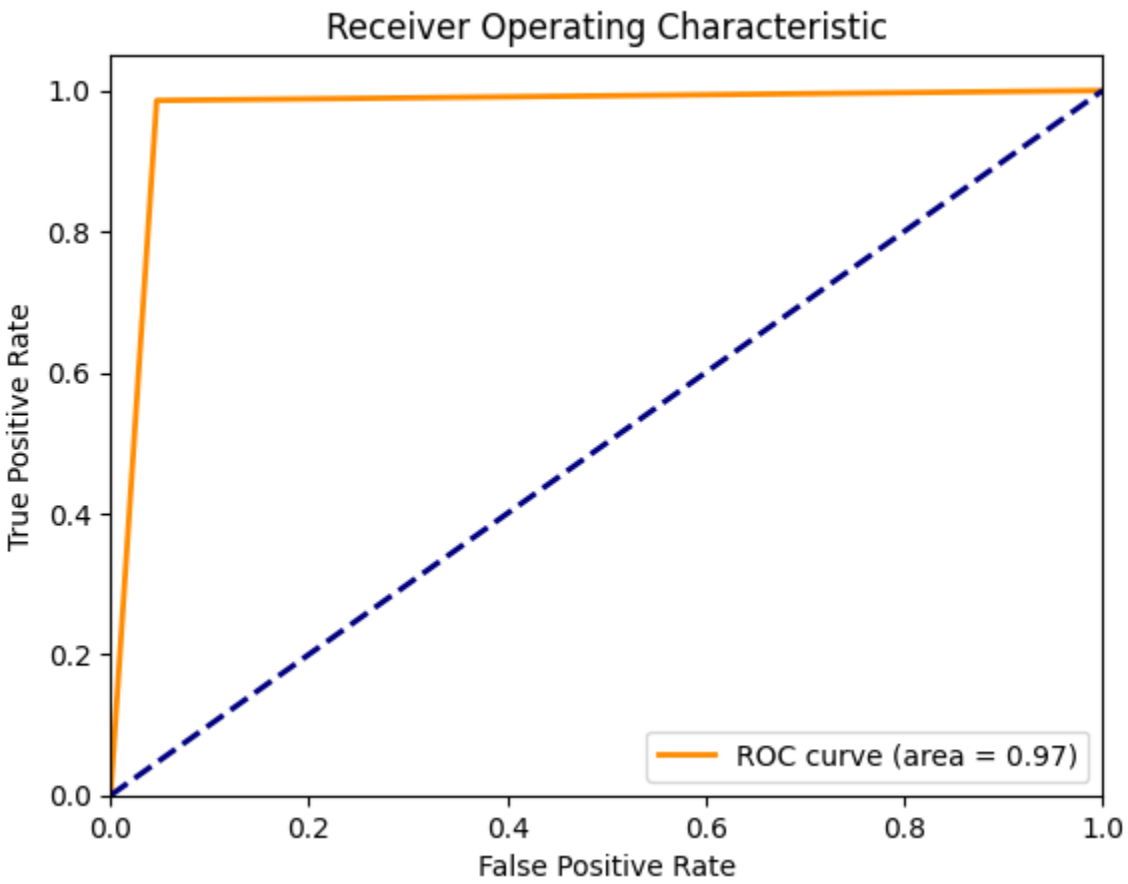



```
accuracy:  0.9736842105263158
recall:  0.9859154929577465
precision:  0.9722222222222222
specificity:  0.9534883720930233
f1 score:  0.979020979020979
auc:  0.969701932525385
```

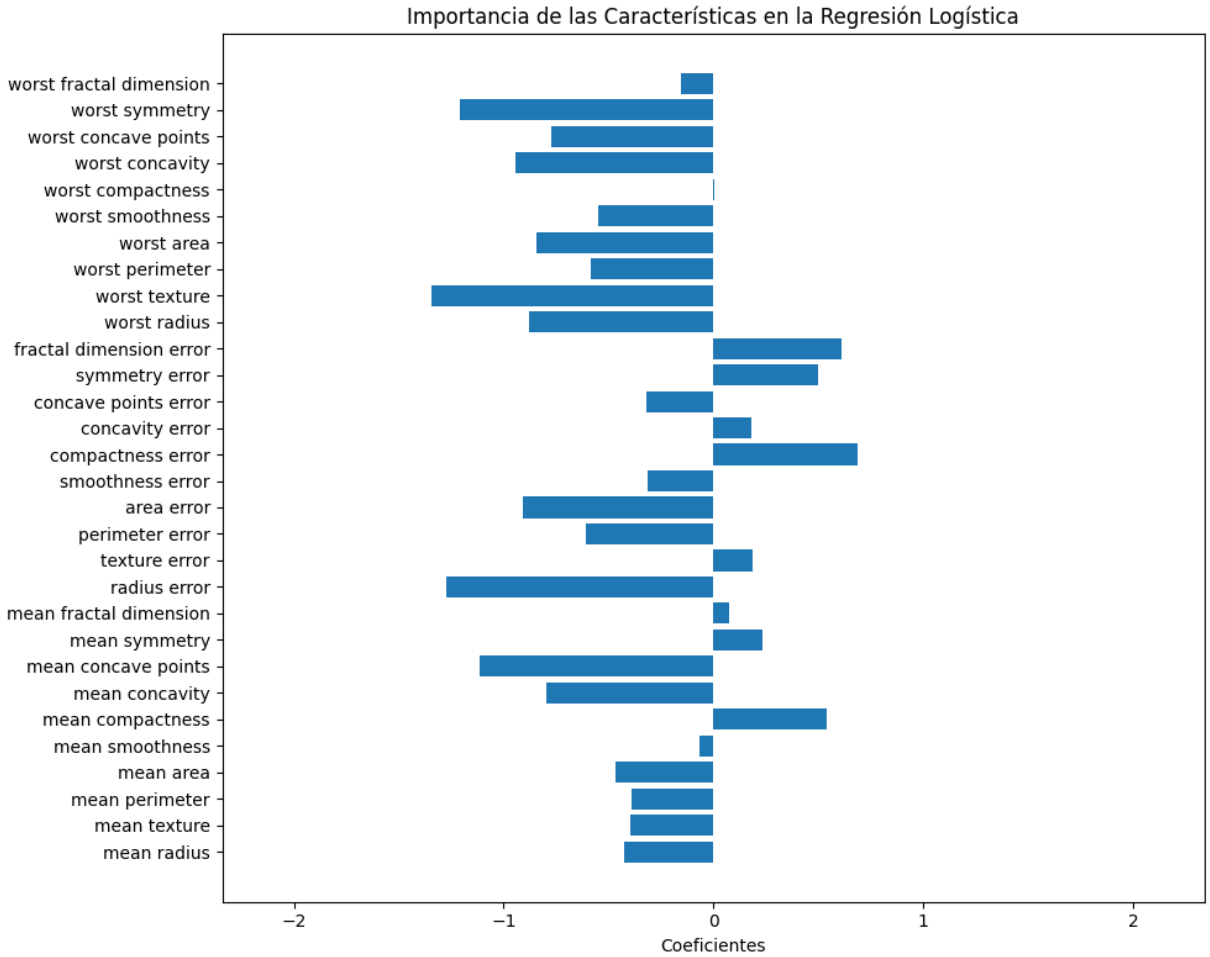
Curva ROC, R cuadrado, Visualización de la importancia de las características.

In [ ]:
```
# Curva ROC
from sklearn.metrics import roc_curve
plt.figure()
lw = 2
plt.plot(roc_curve(y_test, y_pred)[0], roc_curve(y_test, y_pred)[1], color='darkora
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# R Score (R^2 coefficient of determination)
from sklearn.metrics import r2_score
R = r2_score(y_test, y_pred)
print("R2: ", R)

# Visualizar la importancia de las características
feature_names = dataset.feature_names
coefficients = model.coef_
# Configurar el gráfico de barras
```

```
fig, ax = plt.subplots(figsize=(10, 8))

# Crear un gráfico de barras horizontal
ax.barh(feature_names, coefficients[0])
ax.set_title('Importancia de las Características en la Regresión Logística')
ax.set_xlabel('Coeficientes')
ax.set_xlim(-max(abs(coefficients[0]))-1, max(abs(coefficients[0]))+1)

plt.tight_layout()
plt.show()


# Guardar el modelo a un archivo
import joblib
joblib.dump(model, 'logistic_regression_model.pkl')
# Cargar el modelo desde el archivo
loaded_model = joblib.load('logistic_regression_model.pkl')
# Hacer predicciones con el modelo cargado
y_pred = model.predict(X_test)
```




```
R2:  0.8879790370127744
```

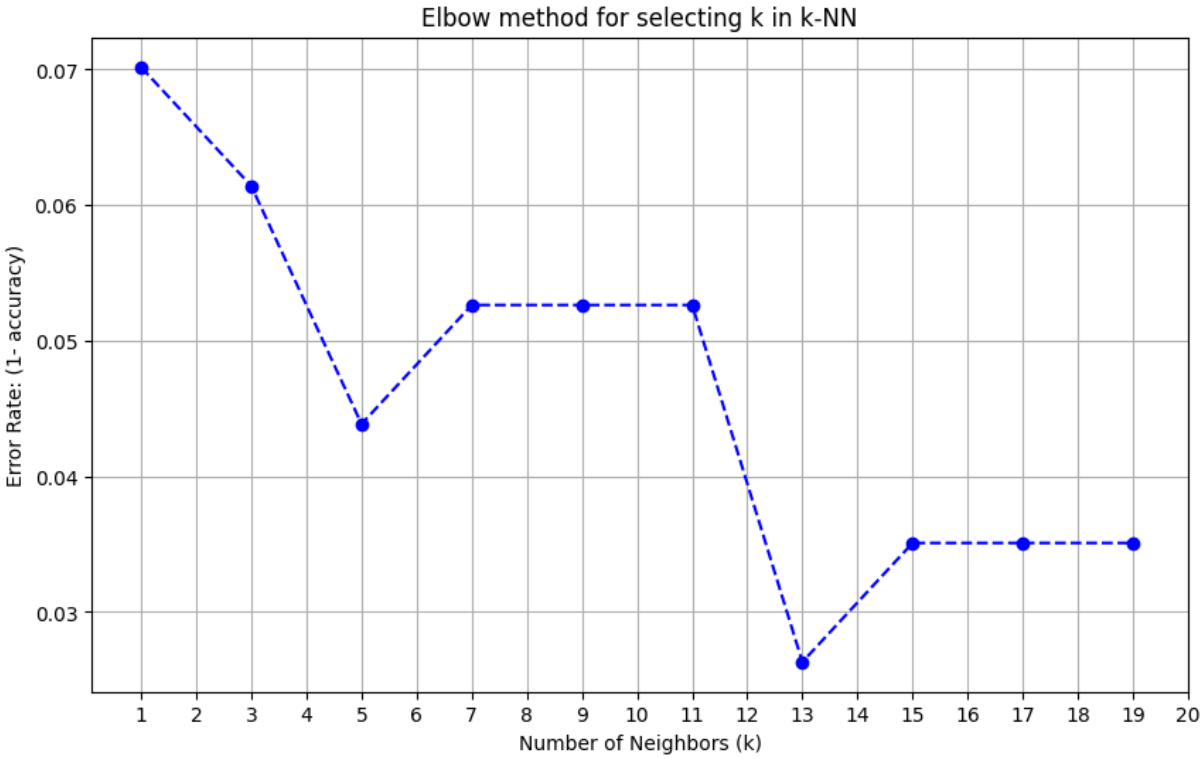


2. K-Nearest Neighbors (K-NN)

Selección de k usando el método del codo.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Cargar el dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Definir el rango de valores de k a evaluar
n = 21
k_range = range(1, n, 2) # en saltos de 2 (solo impares)
error_rates = []
```

```
# Evaluar el modelo para cada valor de k
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k, p=2, weights='distance')
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    error = 1 - accuracy_score(y_test, y_pred)
    error_rates.append(error)


# Graficar la tasa de error para cada valor de k
plt.figure(figsize=(10, 6))
plt.plot(k_range, error_rates, marker='o', linestyle='--', color='b')
plt.title('Elbow method for selecting k in k-NN')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Error Rate: (1- accuracy)')
plt.xticks(np.arange(1, n, 1))
plt.grid()
plt.show()
```



Entrenamiento y evaluación.

In [ ]:
```
# K-NN for breast cancer

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier

# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
```

```
X = dataset.data # 569x30
y = dataset.target # 569x1

# Divide el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Normaliza los datos para que todas las características tengan una escala similar
scaler = MinMaxScaler(feature_range=(0,1)) # [0, 1]
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Crea y entrena el modelo K-NN
model = KNeighborsClassifier(n_neighbors=13, p=2,  # Función euclidean
                             weights='uniform')

model.fit(X_train, y_train)

# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)

# Convierte las probabilidades en etiquetas binarias (0 o 1)
# y_pred = (y_pred > 0.5)

# Muestra el informe de evaluación del modelo entrenado
print(classification_report(y_test, y_pred))
```

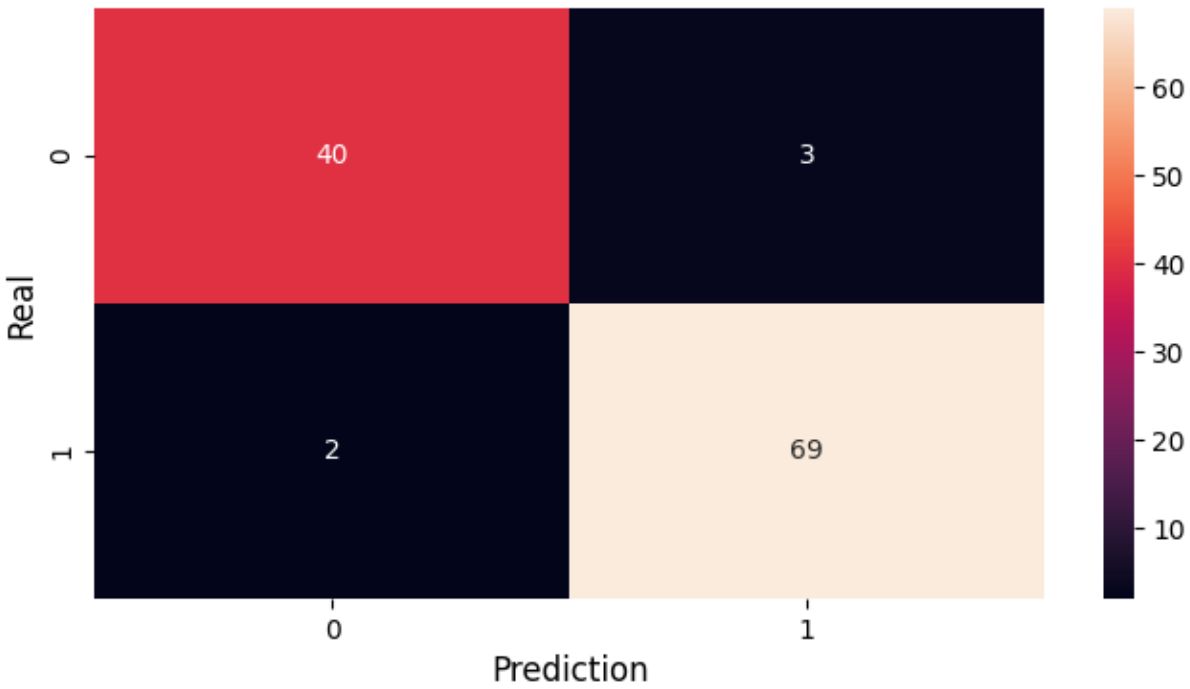
```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        43
           1       0.96      0.97      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Matriz de confusión y Métricas de Evaluación

In [ ]:
```
# Matriz de confusión:
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
print("confusion matrix: \n", cm)
# gráfica cm
plt.figure(figsize = (8,4))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Prediction', fontsize = 12)
plt.ylabel('Real', fontsize = 12)
plt.show()

# Exactitud:
from sklearn.metrics import accuracy_score
```

```
acc = accuracy_score(y_test, y_pred)
print("accuracy: ", acc)

# Sensibilidad:
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("recall: ", recall)

# Precisión:
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("precision: ", precision)

# Especificidad
# 'specificity' is just a special case of 'recall'.
# specificity is the recall of the negative class
specificity = recall_score(y_test, y_pred, pos_label=0)
print("specificity: ", specificity)

# Puntuación F1:
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("f1 score: ", f1)

# Área bajo la curva:
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, y_pred)
print("auc: ", auc)
```

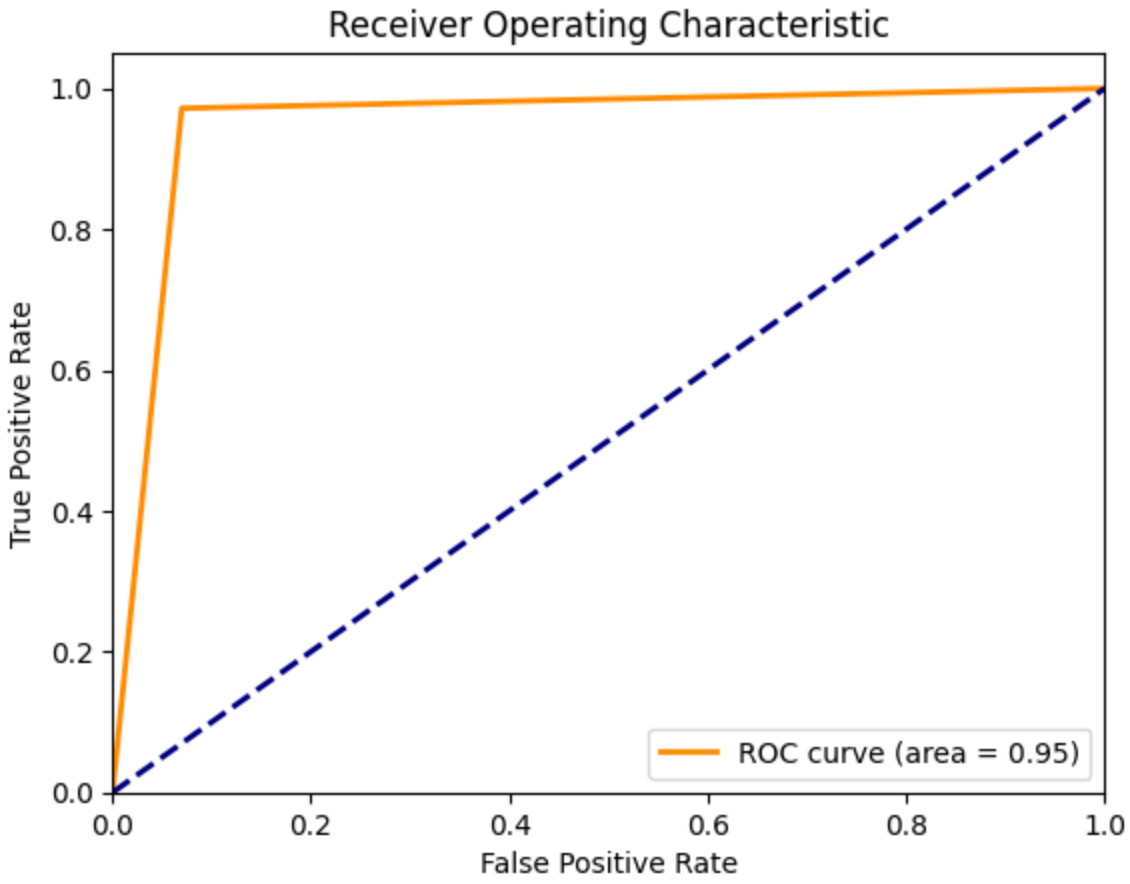
```
confusion matrix:
 [[40  3]
 [ 2 69]]
```



```
accuracy:  0.956140350877193
recall:  0.971830985915493
precision:  0.9583333333333334
specificity:  0.9302325581395349
f1 score:  0.965034965034965
auc:  0.9510317720275139
```

Curva ROC, R cuadrado.

In [ ]:
```python
# Curva ROC
from sklearn.metrics import roc_curve
plt.figure()
lw = 2
plt.plot(roc_curve(y_test, y_pred)[0], roc_curve(y_test, y_pred)[1], color='darkora
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# R Score (R^2 coefficient of determination)
from sklearn.metrics import r2_score
R = r2_score(y_test, y_pred)
print("R2: ", R)

# Guardar el modelo a un archivo
import joblib
joblib.dump(model, 'knn_model.pkl')
# Cargar el modelo desde el archivo
loaded_model = joblib.load('knn_model.pkl')
# Hacer predicciones con el modelo cargado
y_pred = model.predict(X_test)
```




```
R2:  0.8132983950212905
```

3. Árbol de Decisión

Entrenamiento y evaluación.

In [ ]:
```
# Decision Tree for breast cancer

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier

# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
y = dataset.target # 569x1

# Divide el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Normaliza los datos para que todas las características tengan una escala similar
scaler = MinMaxScaler(feature_range=(0,1)) # [0, 1]
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

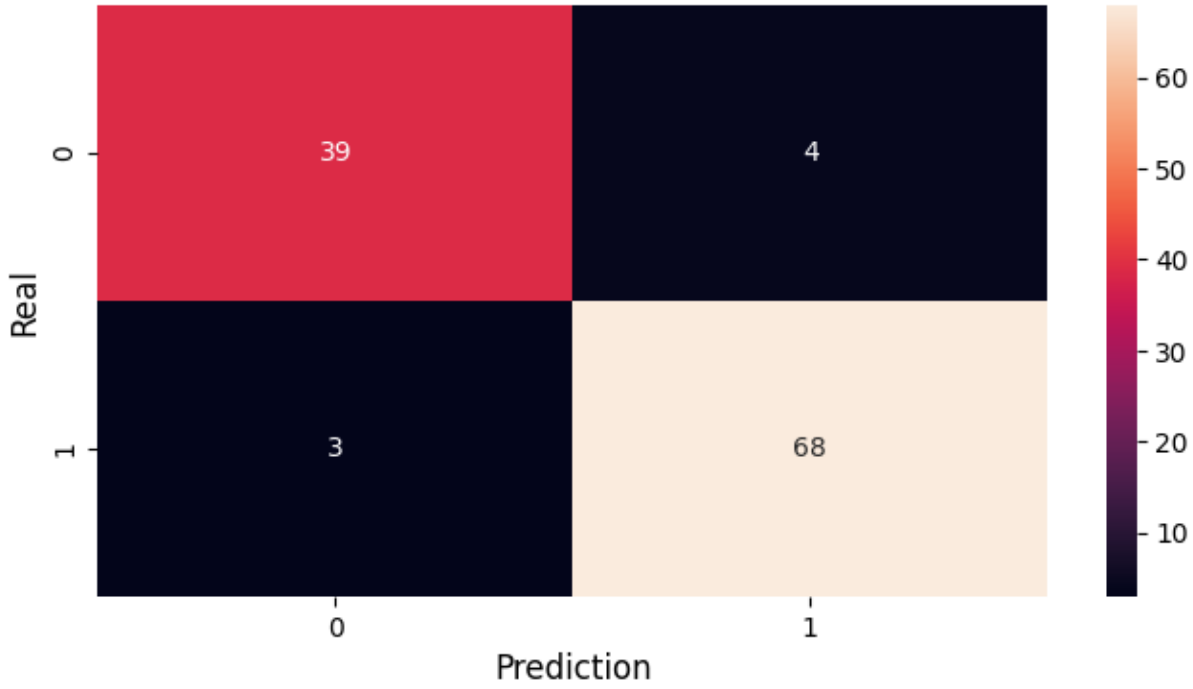
```
# Crea y entrena el modelo de árbol de decisión
model = DecisionTreeClassifier(max_depth=4, criterion = 'gini')
model.fit(X_train, y_train)

# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)


# Convierte las probabilidades en etiquetas binarias (0 o 1)
y_pred = (y_pred > 0.5)

# Muestra el informe de evaluación del modelo entrenado
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92        43
           1       0.94      0.96      0.95        71

    accuracy                           0.94       114
   macro avg       0.94      0.93      0.93       114
weighted avg       0.94      0.94      0.94       114
```

Matriz de confusión y Métricas de Evaluación

In [ ]:
```
# Matriz de confusión:
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
print("confusion matrix: \n", cm)
# gráfica cm
plt.figure(figsize = (8,4))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Prediction', fontsize = 12)
plt.ylabel('Real', fontsize = 12)
plt.show()

# Exactitud:
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("accuracy: ", acc)

# Sensibilidad:
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("recall: ", recall)

# Precisión:
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("precision: ", precision)

# Especificidad
# 'specificity' is just a special case of 'recall'.
# specificity is the recall of the negative class
specificity = recall_score(y_test, y_pred, pos_label=0)
print("specificity: ", specificity)

# Puntuación F1:
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("f1 score: ", f1)

# Área bajo la curva:
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, y_pred)
print("auc: ", auc)
```

```
confusion matrix:
 [[39  4]
 [ 3 68]]
```

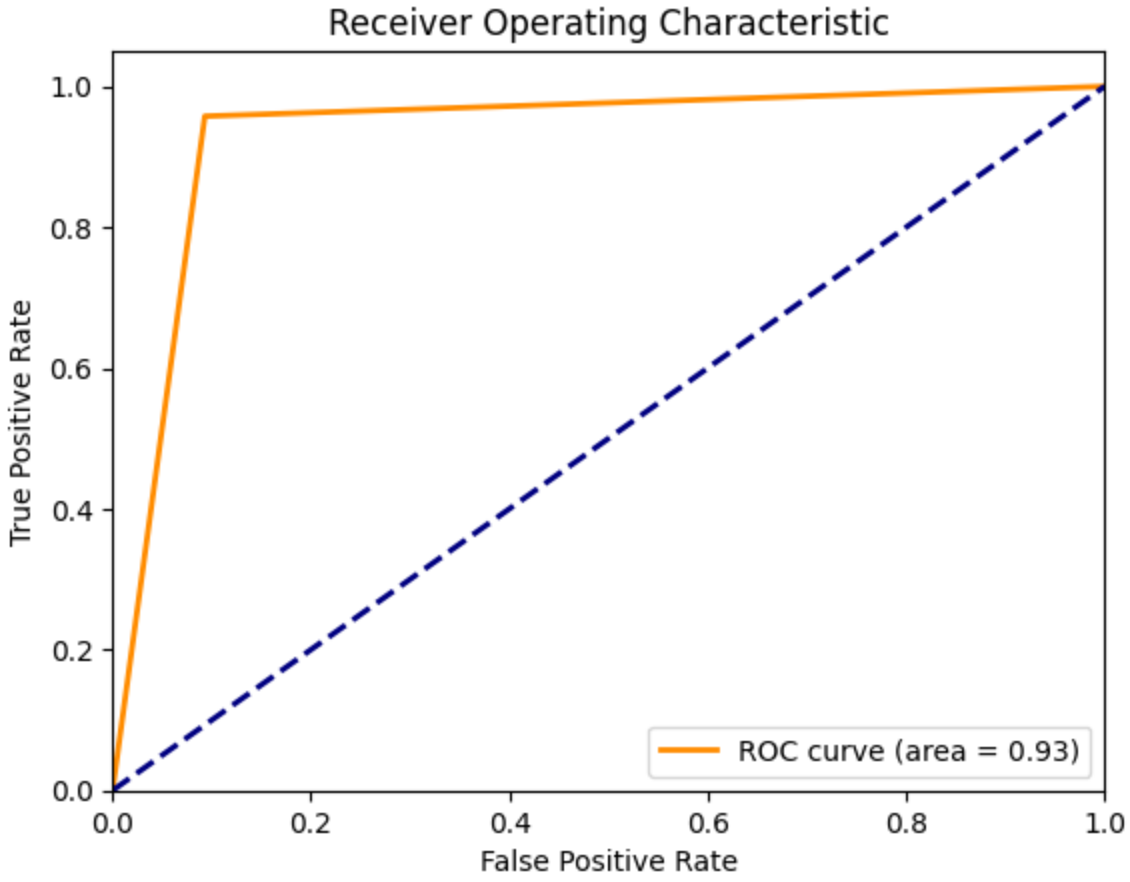



```
accuracy:  0.9385964912280702
recall:  0.9577464788732394
precision:  0.9444444444444444
specificity:  0.9069767441860465
f1 score:  0.951048951048951
auc:  0.932361611529643
```

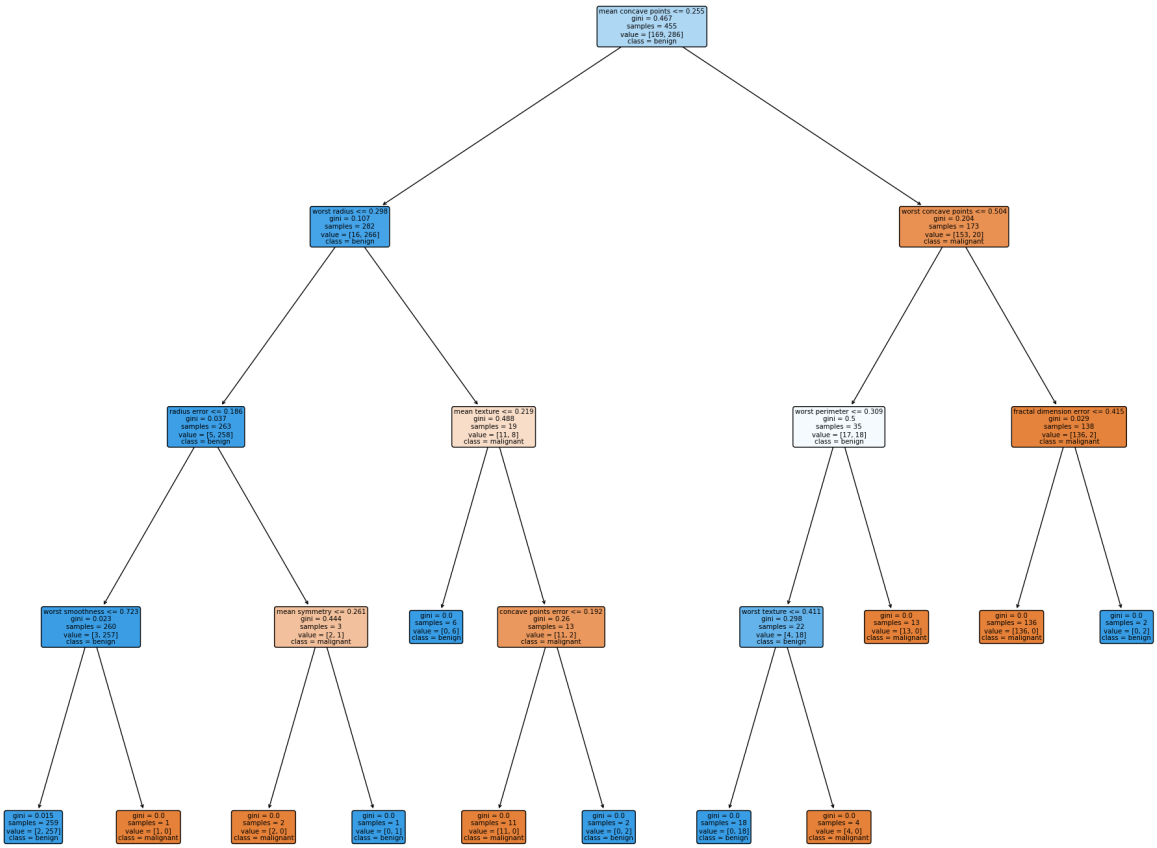
Curva ROC, R cuadrado, Visualización del árbol y la importancia de las características.

In [ ]:
```
# Curva ROC
from sklearn.metrics import roc_curve
plt.figure()
lw = 2
plt.plot(roc_curve(y_test, y_pred)[0], roc_curve(y_test, y_pred)[1], color='darkora
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# R Score (R^2 coefficient of determination)
from sklearn.metrics import r2_score
R = r2_score(y_test, y_pred)
print("R2: ", R)


# Visualizar un árbol de decisión usando matplotlib
from sklearn.tree import plot_tree
# Crear la figura y el eje
fig, ax = plt.subplots(figsize=(24, 20))
```

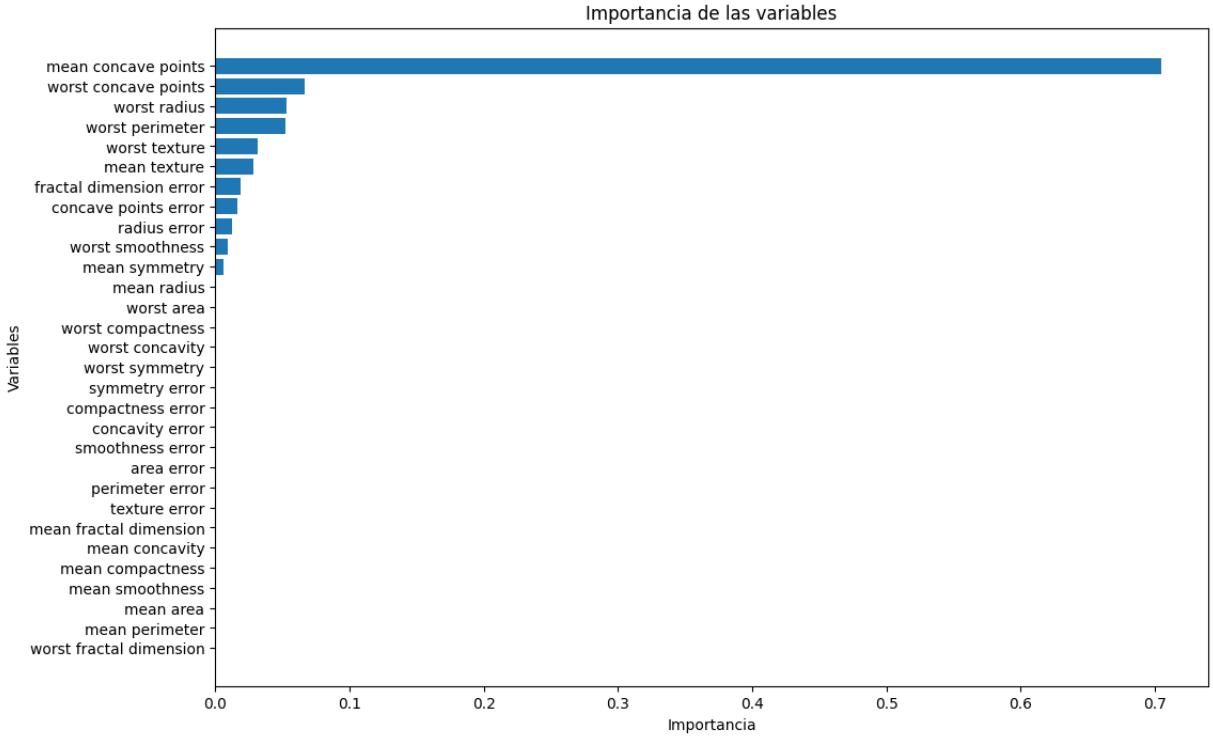
```
# Dibujar el árbol de decisión
plot_tree(model,
                feature_names = dataset.feature_names,
                class_names = dataset.target_names,
                filled=True,
                rounded=True,
                ax=ax)
# Mostrar la gráfica
plt.show()

# Calcular y visualizar la importancia de las variables en la predicción del modelo
importances = model.feature_importances_

# Crear un DataFrame para visualizar las importancias
import pandas as pd
feature_importances = pd.DataFrame({
    'Variable': dataset.feature_names,
    'Importancia': importances
}).sort_values(by='Importancia', ascending=False)

print(feature_importances)

# Visualizar las importancias de las variables
plt.figure(figsize=(12, 8))
plt.barh(feature_importances['Variable'], feature_importances['Importancia'])
plt.xlabel('Importancia')
plt.ylabel('Variables')
plt.title('Importancia de las variables')
plt.gca().invert_yaxis()
plt.show()

# Guardar el modelo a un archivo
import joblib
joblib.dump(model, 'decision_tree_model.pkl')
# Cargar el modelo desde el archivo
loaded_model = joblib.load('decision_tree_model.pkl')
# Hacer predicciones con el modelo cargado
y_pred = model.predict(X_test)
```




```
R2:  0.7386177530298068
```


mean concave points <= 0.255
gini = 0.467
samples = 455
value = [169, 286]
class = benign
worst radius <= 0.298
gini = 0.107
samples = 282
value = [16, 266]
class = benign
worst concave points <= 0.504
gini = 0.204
samples = 173
value = [153, 20]
class = malignant
radius error <= 0.186
gini = 0.037
samples = 263
value = [5, 258]
class = benign
mean texture <= 0.219
gini = 0.488
samples = 19
value = [11, 8]
class = malignant
worst perimeter <= 0.309
gini = 0.5
samples = 35
value = [17, 18]
class = benign
fractal dimension error <= 0.415
gini = 0.029
samples = 138
value = [136, 2]
class = malignant
worst smoothness <= 0.723
gini = 0.023
samples = 260
value = [3, 257]
class = benign
mean symmetry <= 0.261
gini = 0.444
samples = 3
value = [2, 1]
class = malignant
gini = 0.0
samples = 6
value = [0, 6]
class = benign
concave points error <= 0.192
gini = 0.26
samples = 13
value = [11, 2]
class = malignant
worst texture <= 0.411
gini = 0.298
samples = 22
value = [4, 18]
class = benign
gini = 0.0
samples = 13
value = [13, 0]
class = malignant
gini = 0.0
samples = 136
value = [136, 0]
class = malignant
gini = 0.0
samples = 2
value = [0, 2]
class = benign
gini = 0.015
samples = 259
value = [2, 257]
class = benign
gini = 0.0
samples = 1
value = [1, 0]
class = malignant
gini = 0.0
samples = 2
value = [2, 0]
class = malignant
gini = 0.0
samples = 1
value = [0, 1]
class = benign
gini = 0.0
samples = 11
value = [11, 0]
class = malignant
gini = 0.0
samples = 2
value = [0, 2]
class = benign
gini = 0.0
samples = 18
value = [0, 18]
class = benign
gini = 0.0
samples = 4
value = [4, 0]
class = malignant

	Variable	Importancia
7	mean concave points	0.704583
27	worst concave points	0.066901
20	worst radius	0.053295
22	worst perimeter	0.052474
21	worst texture	0.031395
1	mean texture	0.028196
19	fractal dimension error	0.018908
17	concave points error	0.016234
10	radius error	0.012211
24	worst smoothness	0.009409
8	mean symmetry	0.006395
0	mean radius	0.000000
23	worst area	0.000000
25	worst compactness	0.000000
26	worst concavity	0.000000
28	worst symmetry	0.000000
18	symmetry error	0.000000
15	compactness error	0.000000
16	concavity error	0.000000
14	smoothness error	0.000000
13	area error	0.000000
12	perimeter error	0.000000
11	texture error	0.000000
9	mean fractal dimension	0.000000
6	mean concavity	0.000000
5	mean compactness	0.000000
4	mean smoothness	0.000000
3	mean area	0.000000
2	mean perimeter	0.000000
29	worst fractal dimension	0.000000



3. Redes Neuronales Artificiales (RNA)

Creación, Entrenamiento y evaluación.

In [ ]:
```
# RNA for breast cancer


from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping


# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
y = dataset.target # 569x1

# Divide el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Normaliza los datos para que todas las características tengan una escala similar
scaler = MinMaxScaler(feature_range=(0,1)) # [0, 1]
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Crea y entrena el modelo RNA
model = Sequential()
model.add(Dense(10, activation='relu', input_dim=30))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.summary()

opt = Adam(learning_rate = 1e-2) # by default lr=1e-3
model.compile(loss='binary_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

# Configurar early stopping para evitar overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
                               restore_best_weights=True)


history = model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1,
          validation_data=(X_test, y_test), callbacks=[early_stopping])

# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)
```

```
# Convierte las salidas en etiquetas binarias (0 o 1)
y_pred = (y_pred > 0.5)

# Muestra el informe de evaluación del modelo entrenado
print(classification_report(y_test, y_pred))
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 10)                310

 dropout_1 (Dropout)         (None, 10)                0

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 321 (1.25 KB)
Trainable params: 321 (1.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/100
15/15 [==============================] - 2s 44ms/step - loss: 0.6302 - accuracy: 0.7
121 - val_loss: 0.5408 - val_accuracy: 0.9123
Epoch 2/100
15/15 [==============================] - 0s 10ms/step - loss: 0.5230 - accuracy: 0.8
154 - val_loss: 0.4173 - val_accuracy: 0.9474
Epoch 3/100
15/15 [==============================] - 0s 8ms/step - loss: 0.4003 - accuracy: 0.88
13 - val_loss: 0.3091 - val_accuracy: 0.9474
Epoch 4/100
15/15 [==============================] - 0s 12ms/step - loss: 0.3337 - accuracy: 0.8
747 - val_loss: 0.2477 - val_accuracy: 0.9386
Epoch 5/100
15/15 [==============================] - 0s 24ms/step - loss: 0.3001 - accuracy: 0.8
967 - val_loss: 0.2099 - val_accuracy: 0.9386
Epoch 6/100
15/15 [==============================] - 0s 10ms/step - loss: 0.2698 - accuracy: 0.8
901 - val_loss: 0.1827 - val_accuracy: 0.9561
Epoch 7/100
15/15 [==============================] - 0s 14ms/step - loss: 0.2404 - accuracy: 0.8
989 - val_loss: 0.1634 - val_accuracy: 0.9474
Epoch 8/100
15/15 [==============================] - 0s 18ms/step - loss: 0.2321 - accuracy: 0.9
011 - val_loss: 0.1586 - val_accuracy: 0.9474
Epoch 9/100
15/15 [==============================] - 0s 19ms/step - loss: 0.2186 - accuracy: 0.9
055 - val_loss: 0.1528 - val_accuracy: 0.9649
Epoch 10/100
15/15 [==============================] - 0s 11ms/step - loss: 0.2107 - accuracy: 0.9
077 - val_loss: 0.1265 - val_accuracy: 0.9474
Epoch 11/100
15/15 [==============================] - 0s 15ms/step - loss: 0.1820 - accuracy: 0.9
429 - val_loss: 0.1186 - val_accuracy: 0.9737
Epoch 12/100
15/15 [==============================] - 0s 19ms/step - loss: 0.1827 - accuracy: 0.9
231 - val_loss: 0.1126 - val_accuracy: 0.9649
Epoch 13/100
15/15 [==============================] - 0s 20ms/step - loss: 0.1891 - accuracy: 0.9
121 - val_loss: 0.1064 - val_accuracy: 0.9737
Epoch 14/100
15/15 [==============================] - 0s 18ms/step - loss: 0.1607 - accuracy: 0.9
```

```
495 - val_loss: 0.1113 - val_accuracy: 0.9737
Epoch 15/100
15/15 [==============================] - 0s 8ms/step - loss: 0.1622 - accuracy: 0.92
75 - val_loss: 0.0939 - val_accuracy: 0.9649
Epoch 16/100
15/15 [==============================] - 0s 11ms/step - loss: 0.1501 - accuracy: 0.9
385 - val_loss: 0.0872 - val_accuracy: 0.9825
Epoch 17/100
15/15 [==============================] - 0s 17ms/step - loss: 0.1525 - accuracy: 0.9
363 - val_loss: 0.0885 - val_accuracy: 0.9737
Epoch 18/100
15/15 [==============================] - 0s 15ms/step - loss: 0.1364 - accuracy: 0.9
473 - val_loss: 0.0849 - val_accuracy: 0.9737
Epoch 19/100
15/15 [==============================] - 0s 10ms/step - loss: 0.1479 - accuracy: 0.9
407 - val_loss: 0.0806 - val_accuracy: 0.9825
Epoch 20/100
15/15 [==============================] - 0s 13ms/step - loss: 0.1185 - accuracy: 0.9
560 - val_loss: 0.0773 - val_accuracy: 0.9825
Epoch 21/100
15/15 [==============================] - 0s 9ms/step - loss: 0.1300 - accuracy: 0.94
95 - val_loss: 0.0851 - val_accuracy: 0.9737
Epoch 22/100
15/15 [==============================] - 0s 8ms/step - loss: 0.1313 - accuracy: 0.94
51 - val_loss: 0.0735 - val_accuracy: 0.9737
Epoch 23/100
15/15 [==============================] - 0s 12ms/step - loss: 0.1249 - accuracy: 0.9
604 - val_loss: 0.0734 - val_accuracy: 0.9737
Epoch 24/100
15/15 [==============================] - 0s 13ms/step - loss: 0.1208 - accuracy: 0.9
495 - val_loss: 0.0725 - val_accuracy: 0.9737
Epoch 25/100
15/15 [==============================] - 0s 13ms/step - loss: 0.1221 - accuracy: 0.9
473 - val_loss: 0.0728 - val_accuracy: 0.9737
Epoch 26/100
15/15 [==============================] - 0s 14ms/step - loss: 0.1246 - accuracy: 0.9
451 - val_loss: 0.0729 - val_accuracy: 0.9825
Epoch 27/100
15/15 [==============================] - 0s 16ms/step - loss: 0.1127 - accuracy: 0.9
670 - val_loss: 0.0726 - val_accuracy: 0.9737
Epoch 28/100
15/15 [==============================] - 0s 20ms/step - loss: 0.1242 - accuracy: 0.9
385 - val_loss: 0.0688 - val_accuracy: 0.9825
Epoch 29/100
15/15 [==============================] - 0s 20ms/step - loss: 0.1133 - accuracy: 0.9
560 - val_loss: 0.0651 - val_accuracy: 0.9825
Epoch 30/100
15/15 [==============================] - 0s 14ms/step - loss: 0.1081 - accuracy: 0.9
604 - val_loss: 0.0734 - val_accuracy: 0.9737
Epoch 31/100
15/15 [==============================] - 0s 16ms/step - loss: 0.1091 - accuracy: 0.9
560 - val_loss: 0.0629 - val_accuracy: 0.9737
Epoch 32/100
15/15 [==============================] - 0s 16ms/step - loss: 0.1062 - accuracy: 0.9
582 - val_loss: 0.0632 - val_accuracy: 0.9737
Epoch 33/100
```

```
15/15 [==============================] - 0s 19ms/step - loss: 0.0964 - accuracy: 0.9
604 - val_loss: 0.0732 - val_accuracy: 0.9737
Epoch 34/100
15/15 [==============================] - 0s 19ms/step - loss: 0.0915 - accuracy: 0.9
626 - val_loss: 0.0614 - val_accuracy: 0.9737
Epoch 35/100
15/15 [==============================] - 0s 19ms/step - loss: 0.0929 - accuracy: 0.9
648 - val_loss: 0.0653 - val_accuracy: 0.9737
Epoch 36/100
15/15 [==============================] - 0s 26ms/step - loss: 0.0960 - accuracy: 0.9
648 - val_loss: 0.0650 - val_accuracy: 0.9737
Epoch 37/100
15/15 [==============================] - 0s 20ms/step - loss: 0.1055 - accuracy: 0.9
626 - val_loss: 0.0611 - val_accuracy: 0.9825
Epoch 38/100
15/15 [==============================] - 0s 14ms/step - loss: 0.0791 - accuracy: 0.9
758 - val_loss: 0.0640 - val_accuracy: 0.9737
Epoch 39/100
15/15 [==============================] - 0s 14ms/step - loss: 0.0872 - accuracy: 0.9
692 - val_loss: 0.0585 - val_accuracy: 0.9825
Epoch 40/100
15/15 [==============================] - 0s 16ms/step - loss: 0.0965 - accuracy: 0.9
604 - val_loss: 0.0622 - val_accuracy: 0.9737
Epoch 41/100
15/15 [==============================] - 0s 18ms/step - loss: 0.0935 - accuracy: 0.9
648 - val_loss: 0.0612 - val_accuracy: 0.9825
Epoch 42/100
15/15 [==============================] - 0s 17ms/step - loss: 0.0936 - accuracy: 0.9
648 - val_loss: 0.0588 - val_accuracy: 0.9825
Epoch 43/100
15/15 [==============================] - 0s 20ms/step - loss: 0.0881 - accuracy: 0.9
670 - val_loss: 0.0641 - val_accuracy: 0.9737
Epoch 44/100
15/15 [==============================] - 0s 18ms/step - loss: 0.0938 - accuracy: 0.9
604 - val_loss: 0.0625 - val_accuracy: 0.9737
Epoch 45/100
15/15 [==============================] - 0s 20ms/step - loss: 0.0873 - accuracy: 0.9
626 - val_loss: 0.0601 - val_accuracy: 0.9825
Epoch 46/100
15/15 [==============================] - 0s 20ms/step - loss: 0.0812 - accuracy: 0.9
714 - val_loss: 0.0581 - val_accuracy: 0.9825
Epoch 47/100
15/15 [==============================] - 0s 24ms/step - loss: 0.0850 - accuracy: 0.9
692 - val_loss: 0.0607 - val_accuracy: 0.9825
Epoch 48/100
15/15 [==============================] - 0s 16ms/step - loss: 0.0953 - accuracy: 0.9
604 - val_loss: 0.0605 - val_accuracy: 0.9825
Epoch 49/100
15/15 [==============================] - 0s 11ms/step - loss: 0.0927 - accuracy: 0.9
582 - val_loss: 0.0620 - val_accuracy: 0.9737
Epoch 50/100
15/15 [==============================] - 0s 8ms/step - loss: 0.0971 - accuracy: 0.95
60 - val_loss: 0.0575 - val_accuracy: 0.9825
Epoch 51/100
15/15 [==============================] - 0s 10ms/step - loss: 0.0851 - accuracy: 0.9
604 - val_loss: 0.0602 - val_accuracy: 0.9737
```

```
Epoch 52/100
15/15 [==============================] - 0s 11ms/step - loss: 0.0885 - accuracy: 0.9
692 - val_loss: 0.0574 - val_accuracy: 0.9737
Epoch 53/100
15/15 [==============================] - 0s 10ms/step - loss: 0.0900 - accuracy: 0.9
604 - val_loss: 0.0608 - val_accuracy: 0.9737
Epoch 54/100
15/15 [==============================] - 0s 8ms/step - loss: 0.0809 - accuracy: 0.97
36 - val_loss: 0.0581 - val_accuracy: 0.9825
Epoch 55/100
15/15 [==============================] - 0s 8ms/step - loss: 0.0834 - accuracy: 0.96
92 - val_loss: 0.0620 - val_accuracy: 0.9737
Epoch 56/100
15/15 [==============================] - 0s 7ms/step - loss: 0.0859 - accuracy: 0.95
38 - val_loss: 0.0587 - val_accuracy: 0.9737
Epoch 57/100
15/15 [==============================] - 0s 11ms/step - loss: 0.0752 - accuracy: 0.9
582 - val_loss: 0.0612 - val_accuracy: 0.9737
Epoch 58/100
15/15 [==============================] - 0s 17ms/step - loss: 0.0793 - accuracy: 0.9
714 - val_loss: 0.0568 - val_accuracy: 0.9825
Epoch 59/100
15/15 [==============================] - 0s 10ms/step - loss: 0.0866 - accuracy: 0.9
560 - val_loss: 0.0572 - val_accuracy: 0.9825
Epoch 60/100
15/15 [==============================] - 0s 8ms/step - loss: 0.0794 - accuracy: 0.97
14 - val_loss: 0.0563 - val_accuracy: 0.9825
Epoch 61/100
15/15 [==============================] - 0s 15ms/step - loss: 0.0712 - accuracy: 0.9
670 - val_loss: 0.0566 - val_accuracy: 0.9825
Epoch 62/100
15/15 [==============================] - 0s 17ms/step - loss: 0.0657 - accuracy: 0.9
802 - val_loss: 0.0668 - val_accuracy: 0.9737
Epoch 63/100
15/15 [==============================] - 0s 11ms/step - loss: 0.0830 - accuracy: 0.9
670 - val_loss: 0.0560 - val_accuracy: 0.9825
Epoch 64/100
15/15 [==============================] - 0s 9ms/step - loss: 0.0790 - accuracy: 0.96
26 - val_loss: 0.0572 - val_accuracy: 0.9825
Epoch 65/100
15/15 [==============================] - 0s 20ms/step - loss: 0.0909 - accuracy: 0.9
560 - val_loss: 0.0619 - val_accuracy: 0.9737
Epoch 66/100
15/15 [==============================] - 0s 24ms/step - loss: 0.0663 - accuracy: 0.9
802 - val_loss: 0.0574 - val_accuracy: 0.9737
Epoch 67/100
15/15 [==============================] - 0s 20ms/step - loss: 0.0799 - accuracy: 0.9
692 - val_loss: 0.0631 - val_accuracy: 0.9737
Epoch 68/100
15/15 [==============================] - 0s 12ms/step - loss: 0.0849 - accuracy: 0.9
648 - val_loss: 0.0600 - val_accuracy: 0.9737
Epoch 69/100
15/15 [==============================] - 0s 12ms/step - loss: 0.0747 - accuracy: 0.9
648 - val_loss: 0.0601 - val_accuracy: 0.9737
Epoch 70/100
15/15 [==============================] - 0s 14ms/step - loss: 0.0856 - accuracy: 0.9
```

```
582 - val_loss: 0.0658 - val_accuracy: 0.9737
Epoch 71/100
15/15 [==============================] - 0s 15ms/step - loss: 0.0765 - accuracy: 0.9
692 - val_loss: 0.0591 - val_accuracy: 0.9825
Epoch 72/100
15/15 [==============================] - 0s 14ms/step - loss: 0.0662 - accuracy: 0.9
846 - val_loss: 0.0617 - val_accuracy: 0.9737
Epoch 73/100
15/15 [==============================] - 0s 12ms/step - loss: 0.0717 - accuracy: 0.9
846 - val_loss: 0.0605 - val_accuracy: 0.9825
4/4 [==============================] - 0s 4ms/step
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        43
           1       0.99      0.99      0.99        71

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

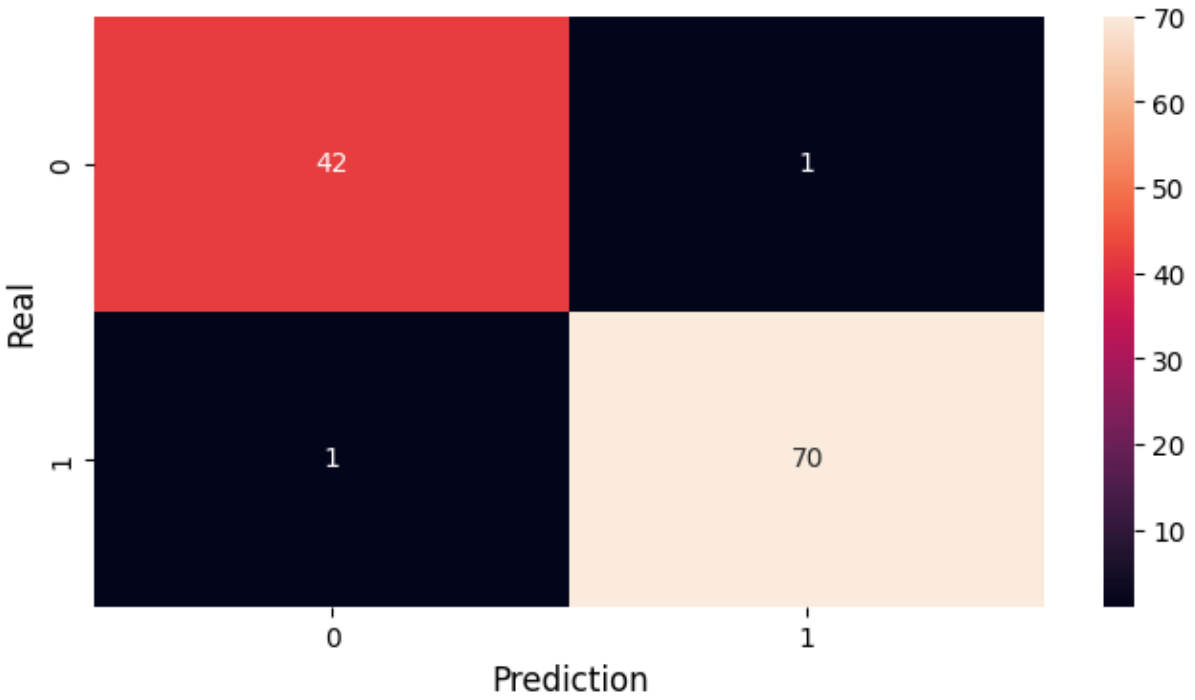
Matriz de confusión y Métricas de Evaluación

In [ ]:

```python
# Matriz de confusión:
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
print("confusion matrix: \n", cm)
# gráfica cm
plt.figure(figsize = (8,4))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Prediction', fontsize = 12)
plt.ylabel('Real', fontsize = 12)
plt.show()

# Exactitud:
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("accuracy: ", acc)

# Sensibilidad:
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("recall: ", recall)

# Precisión:
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("precision: ", precision)

# Especificidad
# 'specificity' is just a special case of 'recall'.
# specificity is the recall of the negative class
```

```
specificity = recall_score(y_test, y_pred, pos_label=0)
print("specificity: ", specificity)

# Puntuación F1:
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("f1 score: ", f1)

# Área bajo la curva:
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, y_pred)
print("auc: ", auc)
```

```
confusion matrix:
 [[42  1]
 [ 1 70]]
```

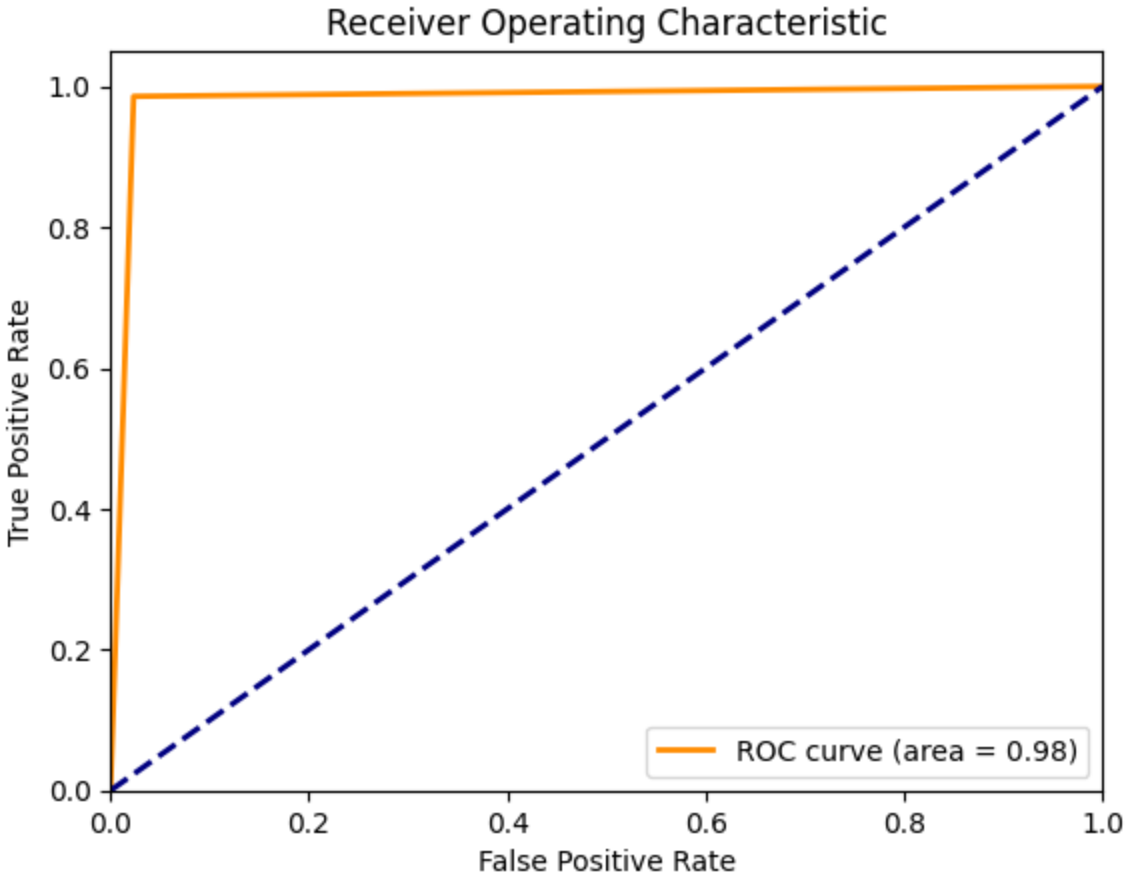



```
accuracy:  0.9824561403508771
recall:  0.9859154929577465
precision:  0.9859154929577465
specificity:  0.9767441860465116
f1 score:  0.9859154929577465
auc:  0.9813298395021289
```

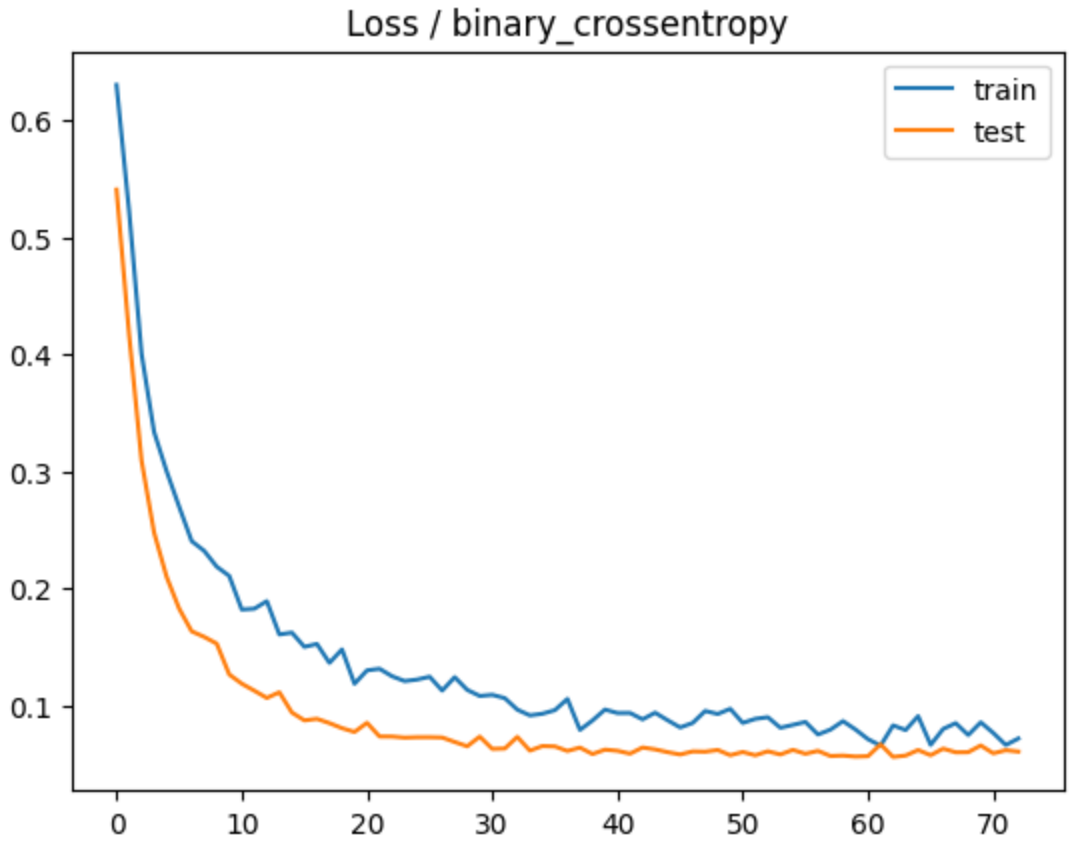
Curva ROC, R cuadrado y Visualización de curva de aprendizaje

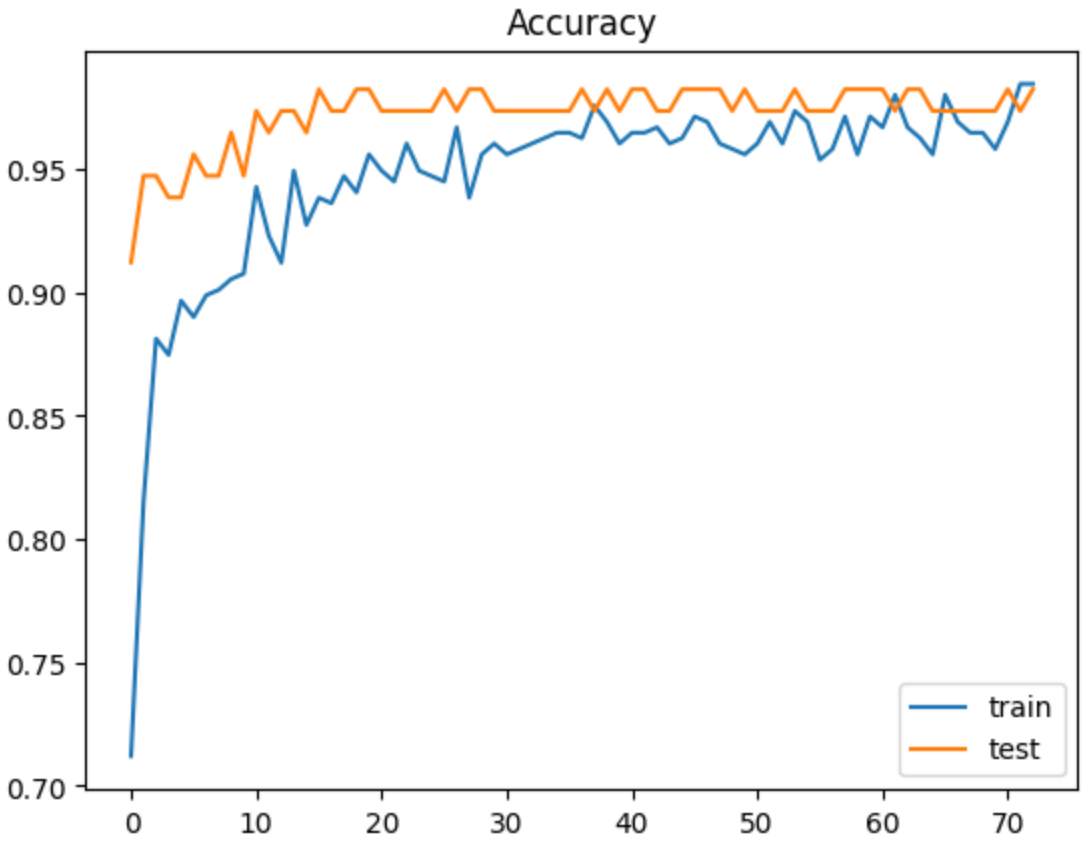
In [ ]:
```
# Curva ROC
from sklearn.metrics import roc_curve
plt.figure()
lw = 2
plt.plot(roc_curve(y_test, y_pred)[0], roc_curve(y_test, y_pred)[1], color='darkora
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# R Score (R^2 coefficient of determination)
from sklearn.metrics import r2_score
R = r2_score(y_test, y_pred)
print("R2: ", R)

# learning curves
# plot loss during training
plt.title('Loss / binary_crossentropy')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
# plot accuracy during training
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()

# Guardar el modelo completo (arquitectura, pesos y configuración) en un archivo HD
model.save('RNA_model.h5')
print("Modelo guardado.")
# Cargar el modelo desde el archivo HDF5
from tensorflow.keras.models import load_model
loaded_model = load_model('RNA_model.h5')
print("Modelo cargado.")


# SHAP (SHapley Additive exPlanations) para explicar las predicciones de un modelo
import shap # pip install shap
# Crear un explainer de SHAP usando en conjunto de entrenamiento
explainer = shap.Explainer(model, X_train)
# Obtener las explicaciones SHAP para el conjunto de prueba
shap_values = explainer.shap_values(X_test)
# Proporciona una visión general de la importancia de las características y su impa
shap.summary_plot(shap_values, X_test, feature_names=dataset.feature_names)
```



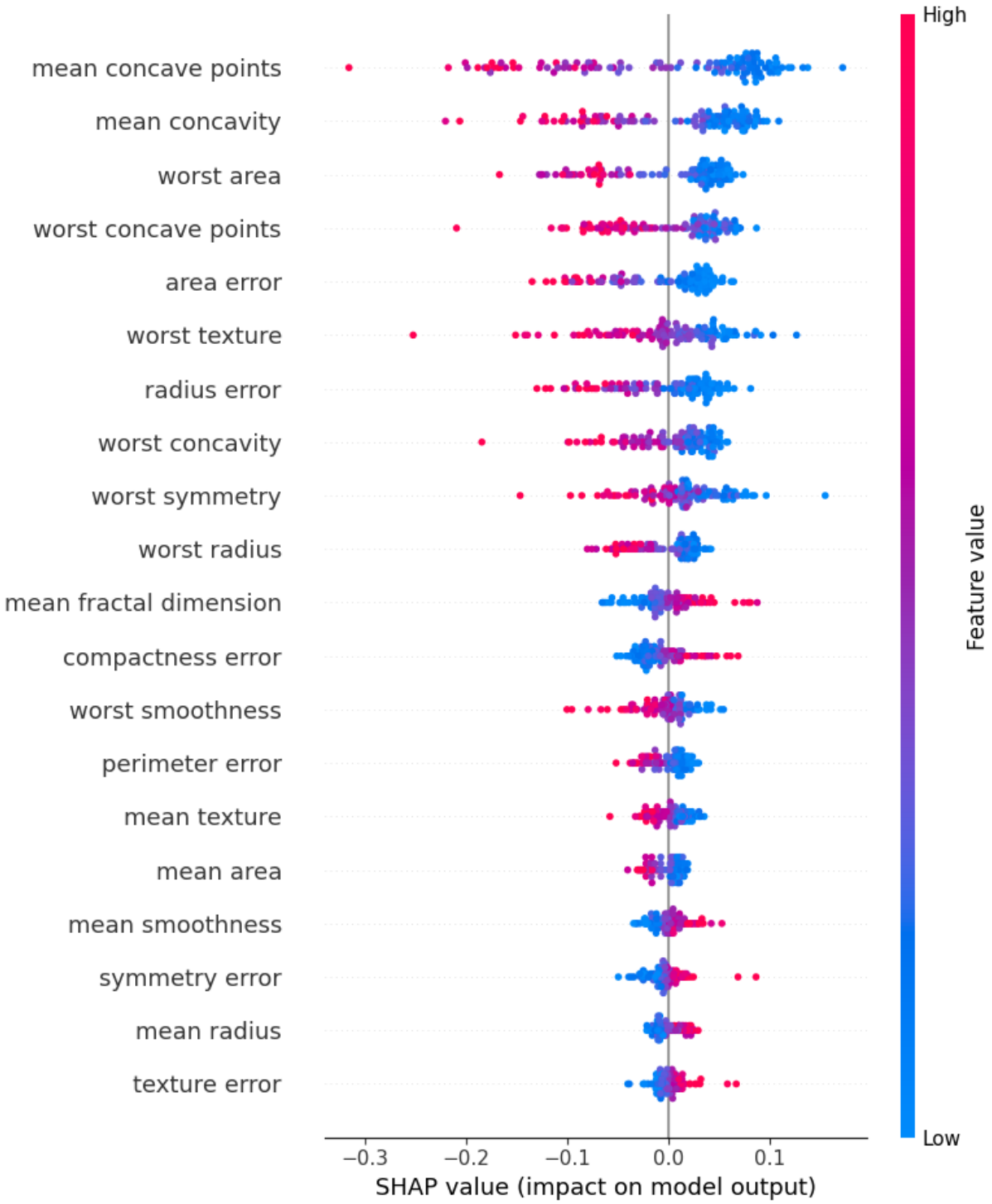

```
R2:  0.9253193580085162
```






```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarni
ng: You are saving your model as an HDF5 file via `model.save()`. This file format i
s considered legacy. We recommend using instead the native Keras format, e.g. `mode
l.save('my_model.keras')`.
  saving_api.save_model(
Modelo guardado.
Modelo cargado.
```

```
PermutationExplainer explainer: 115it [00:12,  7.17it/s]
```



Evaluación Comparativa:

Exactitud (Accuracy): La exactitud mide la proporción de predicciones correctas. Sin embargo, en problemas de clasificación desbalanceada, esta métrica puede ser engañosa.

Precisión y Recall: La precisión (Precision) mide la proporción de verdaderos positivos entre las predicciones positivas. El recall mide la proporción de verdaderos positivos que se identifican correctamente. En problemas médicos, el recall puede ser más importante para minimizar los falsos negativos (pacientes con cáncer no detectados).

F1-Score: El F1-score es la media armónica de la precisión y el recall, proporcionando un balance entre ambos.

AUC-ROC: El área bajo la curva ROC (AUC-ROC) mide la capacidad del modelo para distinguir entre las clases. Un AUC-ROC más alto indica un mejor rendimiento.

Selección del Mejor Modelo:

Para determinar el mejor modelo, se deben comparar estas métricas. Generalmente, se busca un modelo que tenga un alto recall y un buen F1-score, especialmente en contextos médicos.

En muchos casos, las redes neuronales o la regresión logística tienden a ofrecer buenos resultados debido a su capacidad para manejar relaciones complejas en los datos. Sin embargo, si se prioriza la interpretabilidad, un árbol de decisión puede ser más útil.

Si el modelo de regresión logística tiene un AUC-ROC más alto, junto con un buen F1-score, este puede ser el mejor modelo debido a su simplicidad y buen rendimiento general.

Si el modelo de redes neuronales supera significativamente a los demás en términos de AUC-ROC y F1-score, este sería el preferido a pesar de ser más complejo.

Si la interpretabilidad es clave y el árbol de decisión tiene métricas comparables, este modelo podría ser elegido.

Para nuestro caso de análisis vamos a realizar un cuadro comparativo.

Cuadro Comparativo de Modelos de Clasificación

Exactitud (Accuracy): Proporción de predicciones correctas entre todas las predicciones.

Precisión (Precision): Proporción de verdaderos positivos entre las predicciones positivas.

Recall: Proporción de verdaderos positivos entre todos los verdaderos positivos (esencial para minimizar falsos negativos).

F1-Score: Media armónica de la precisión y el recall, balancea ambos.

AUC-ROC: Área bajo la curva ROC, mide la capacidad del modelo para distinguir entre clases.

In [ ]:
```
import pandas as pd

# Resultados de los modelos
data = {
    'Modelo': ['Regresión Logística', 'K-Nearest Neighbors', 'Árbol de Decisión', '
    'Exactitud (Accuracy)': [0.97, 0.96, 0.94, 0.98],
    'Precisión (Precision)': [0.97, 0.95, 0.94, 0.99],
    'Recall': [0.99, 0.97, 0.96, 0.99],
    'F1-Score': [0.98, 0.97, 0.95, 0.99],
```

```
    'AUC-ROC': [0.97, 0.95, 0.93, 0.98]
}

# Crear un DataFrame
df = pd.DataFrame(data)

# Imprimir el DataFrame
print(df)

# # Exportar a un archivo CSV si es necesario
# df.to_csv('comparacion_modelos.csv', index=False)
```

```
                Modelo  Exactitud (Accuracy)  Precisión (Precision)  Recall  \
0  Regresión Logística                  0.97                   0.97    0.99
1  K-Nearest Neighbors                  0.96                   0.95    0.97
2    Árbol de Decisión                  0.94                   0.94    0.96
3     Redes Neuronales                  0.98                   0.99    0.99

   F1-Score  AUC-ROC
0      0.98     0.97
1      0.97     0.95
2      0.95     0.93
3      0.99     0.98
```

Interpretación:

Exactitud (Accuracy): La red neuronal tiene la mayor exactitud con un 98%, seguida por la regresión logística con un 97%. Esto indica que ambos modelos tienen un buen rendimiento en la clasificación de los datos.

Precisión (Precision): La red neuronal muestra la mayor precisión con un 99%, lo que sugiere que es el modelo que mejor identifica correctamente las muestras positivas entre todas las predicciones positivas que hace.

Recall: La red neuronal y la regresión logística tienen el recall más alto (99%), lo que significa que estos modelos capturan la mayoría de las muestras positivas en el conjunto de datos.

F1-Score: La red neuronal tiene el F1-score más alto (99%), lo cual indica un buen equilibrio entre precisión y recall. Esto sugiere que es el modelo que mejor combina ambas métricas.

AUC-ROC: La red neuronal también lidera con un 98% en AUC-ROC, indicando un mejor rendimiento en términos de la capacidad de distinguir entre clases.

Conclusión:

Basado en las métricas evaluadas, la Red Neuronal es el modelo que muestra el mejor rendimiento general en la clasificación del conjunto de datos de cáncer de mama. Esto se debe a su alta exactitud, precisión, recall, F1-score y AUC-ROC, superando a los otros modelos evaluados. La red neuronal es capaz de aprender patrones complejos en los datos

gracias a su estructura profunda, lo que le permite alcanzar estas métricas superiores en comparación con los otros modelos más simples como la regresión logística, KNN y el árbol de decisión.