CASO FINAL

Integrantes: CARLOS PADILLA & XAVIER ASMAL

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 titanic.csv 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
data = pd.read_csv('titanic.csv')


########## Entendimiento de la data ##########

#Verifica la cantidad de datos que hay en los dataset
print('Cantidad de datos:')
print(data.shape)

#Verifica el tipo de datos contenida en ambos dataset
print('Tipos de datos:')
print(data.info())
```

```
Cantidad de datos:
(891, 12)
Tipos de datos:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```

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

#Verifica las estadísticas básicas del dataset
print('Estadísticas del dataset:')
print(data.describe())
```

```
Datos faltantes:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
Estadísticas del dataset:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

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 [ ]:
```
# Transforma los datos de la variable sexo (categórico) en números
data['Sex'].replace(['female','male'],[0,1],inplace=True)

#Transforma los datos de embarque (categórico) en números
data['Embarked'].replace(['Q','S', 'C'],[0,1,2],inplace=True)

#Reemplazo los datos faltantes en la edad por la media de esta variable
print(data["Age"].mean())
promedio = 30
```

```
data['Age'] = data['Age'].replace(np.nan, promedio)

#Crea varios grupos/rangos de edades
#Rangos: 0-8, 9-15, 16-18, 19-25, 26-40, 41-60, 61-100
bins = [0, 8, 15, 18, 25, 40, 60, 100]
names = ['1', '2', '3', '4', '5', '6', '7']
data['Age'] = pd.cut(data['Age'], bins, labels = names)


#Se elimina la columna de "Cabin" ya que tiene muchos datos perdidos
# El parámetro axis=1 indica que se deben eliminar columnas en lugar de filas (axis
# El parámetro inplace indica si la operación se realiza directamente en el
# DataFrame original o devolvuelve una nueva copia con las filas o columnas elimina
data.drop(['Cabin'], axis = 1, inplace=True)

#Elimina las columnas que se considera que no son necesarias para el analisis
data = data.drop(['PassengerId','Name','Ticket'], axis=1)

#Se elimina las filas con datos perdidos
data.dropna(axis=0, how='any', inplace=True)

#Verifica los datos
print(pd.isnull(data).sum())

print(data.shape)

print(data.head())

# Guardar el DataFrame en un archivo CSV
# El parámetro index=False evita que los índices del DataFrame
# se guarden como una columna en el archivo CSV
data.to_csv('train_procesado.csv', index=False, sep=',', encoding='utf-8')
```

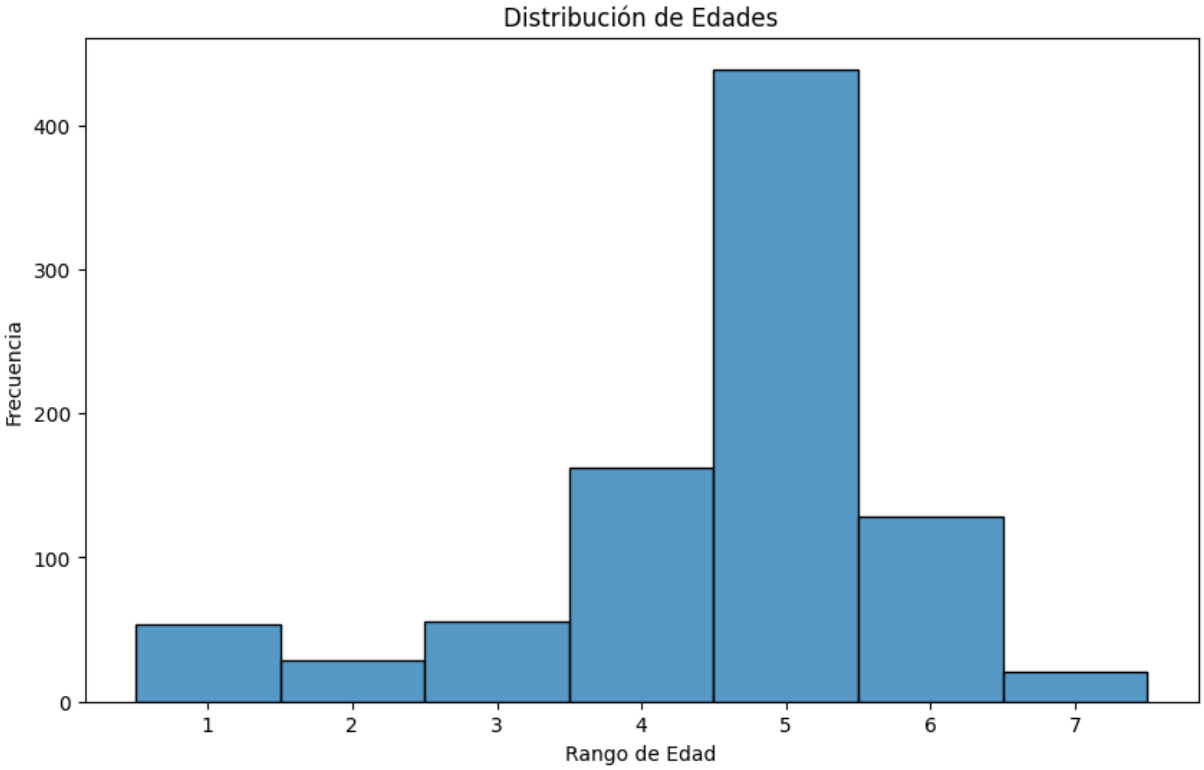
```
29.69911764705882
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
(889, 8)
   Survived  Pclass  Sex Age  SibSp  Parch     Fare  Embarked
0         0       3    1   4      1      0   7.2500       1.0
1         1       1    0   5      1      0  71.2833       2.0
2         1       3    0   5      0      0   7.9250       1.0
3         1       1    0   5      1      0  53.1000       1.0
4         0       3    1   5      0      0   8.0500       1.0
```

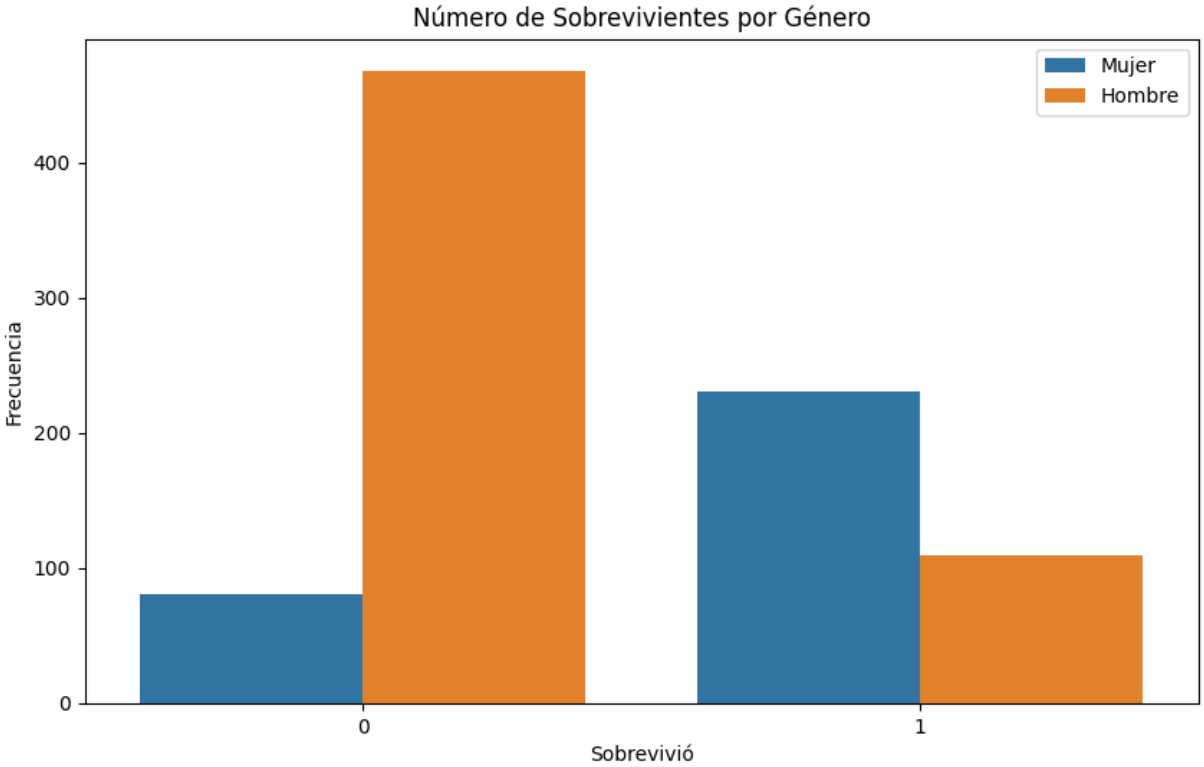
4. Vizualización y Análisis Exploratorio

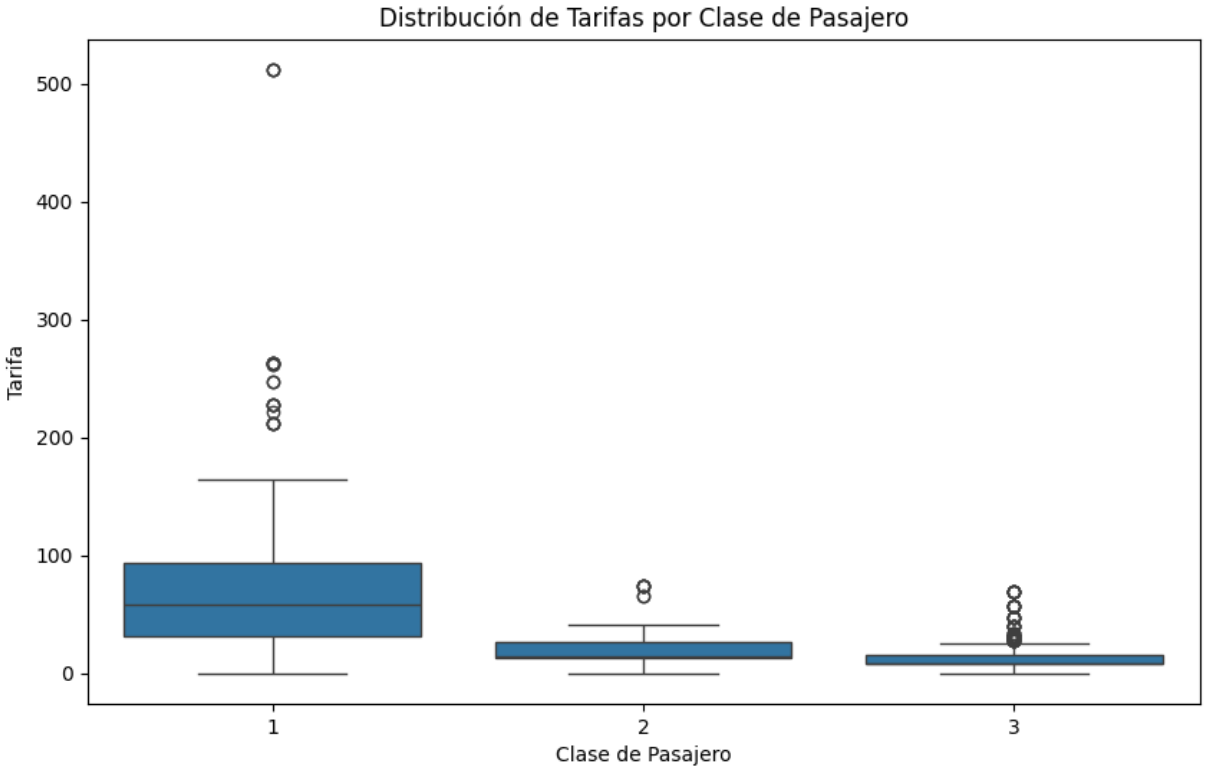
Estos bloques incluyen varias visualizaciones que exploran diferentes aspectos del dataset:

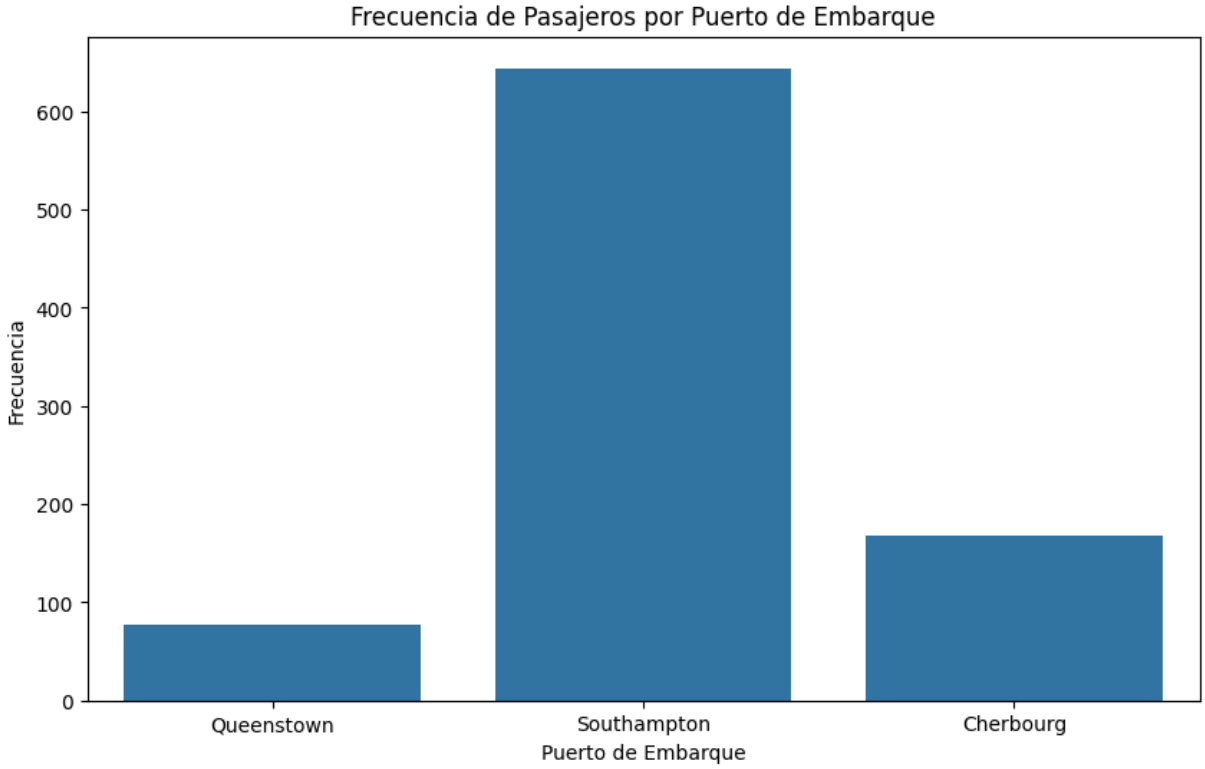
- Histograma de Distribución de Edades
- Barplot del Número de Sobrevivientes por Género
- Boxplot de Tarifas por Clase de Pasajero
- Countplot del Puerto de Embarque
- Matriz de Correlación entre Variables

In [ ]:
```
# 1. Histograma de la distribución de edades
plt.figure(figsize=(10, 6))
sns.histplot(data['Age'], bins=7, kde=False)
plt.title('Distribución de Edades')
plt.xlabel('Rango de Edad')
plt.ylabel('Frecuencia')
plt.show()

# 2. Barplot del número de sobrevivientes por género
plt.figure(figsize=(10, 6))
sns.countplot(x='Survived', hue='Sex', data=data)
plt.title('Número de Sobrevivientes por Género')
plt.xlabel('Sobrevivió')
plt.ylabel('Frecuencia')
plt.legend(['Mujer', 'Hombre'])
plt.show()

# 3. Boxplot de tarifas por clase de pasajero
plt.figure(figsize=(10, 6))
sns.boxplot(x='Pclass', y='Fare', data=data)
plt.title('Distribución de Tarifas por Clase de Pasajero')
plt.xlabel('Clase de Pasajero')
plt.ylabel('Tarifa')
plt.show()

# 4. Countplot del puerto de embarque
plt.figure(figsize=(10, 6))
sns.countplot(x='Embarked', data=data)
plt.title('Frecuencia de Pasajeros por Puerto de Embarque')
plt.xlabel('Puerto de Embarque')
plt.ylabel('Frecuencia')
plt.xticks(ticks=[0, 1, 2], labels=['Queenstown', 'Southampton', 'Cherbourg'])
plt.show()
```


Distribución de Edades
Frecuencia
0
100
200
300
400
1
2
3
4
5
6
7
Rango de Edad


Número de Sobrevivientes por Género
Mujer
Hombre
Frecuencia
0
100
200
300
400
0
1
Sobrevivió





Propósito:

Cada visualización tiene como objetivo explorar y entender diferentes aspectos de los datos, como la distribución de edades, la relación entre género y supervivencia, las diferencias en las tarifas según la clase de pasajero, la distribución de pasajeros por puerto de embarque y las correlaciones entre variables.

Interpretación:

Estas visualizaciones proporcionan insights visuales sobre patrones, tendencias y posibles relaciones dentro del dataset. Por ejemplo, el barplot sugiere que las mujeres tuvieron mayores tasas de supervivencia, mientras que el boxplot indica diferencias en las tarifas pagadas según la clase de pasajero.

5. Matrices de Correlación Detalladas

Propósito:

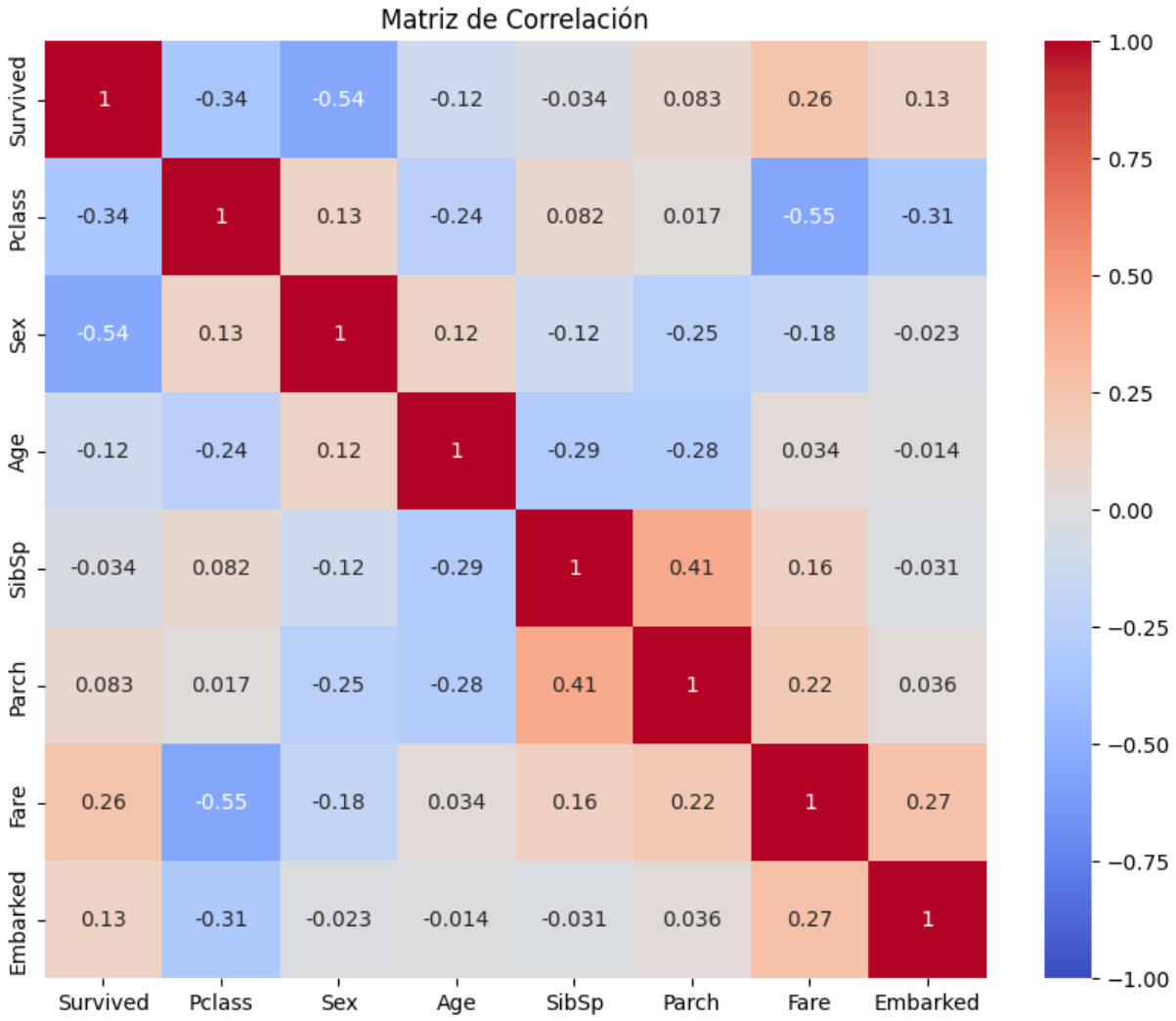
Estos bloques calculan y visualizan matrices de correlación entre variables, enfocándose en aquellas que tienen mayores correlaciones con la variable objetivo (en este caso, 'Survived').

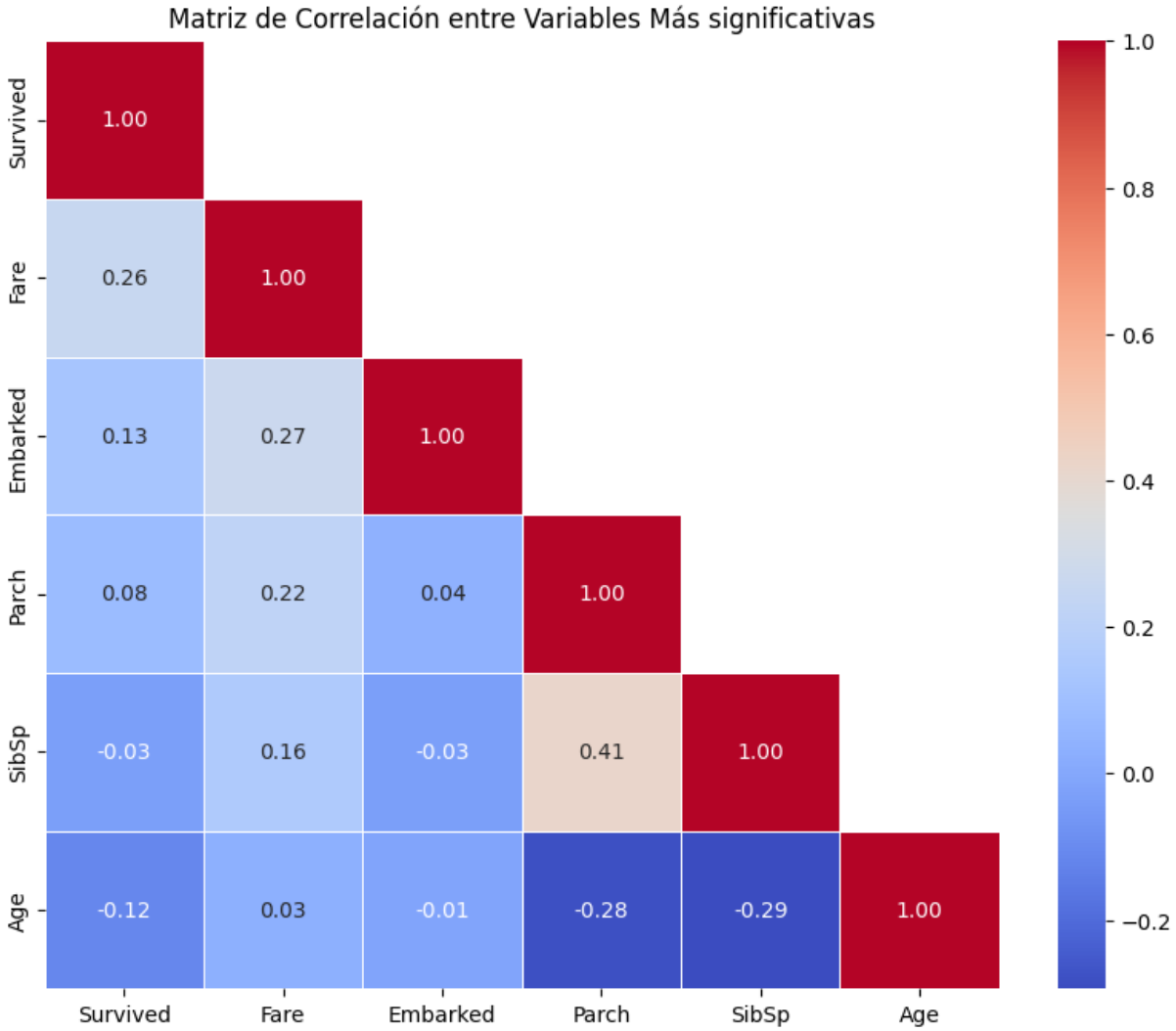
Interpretación:

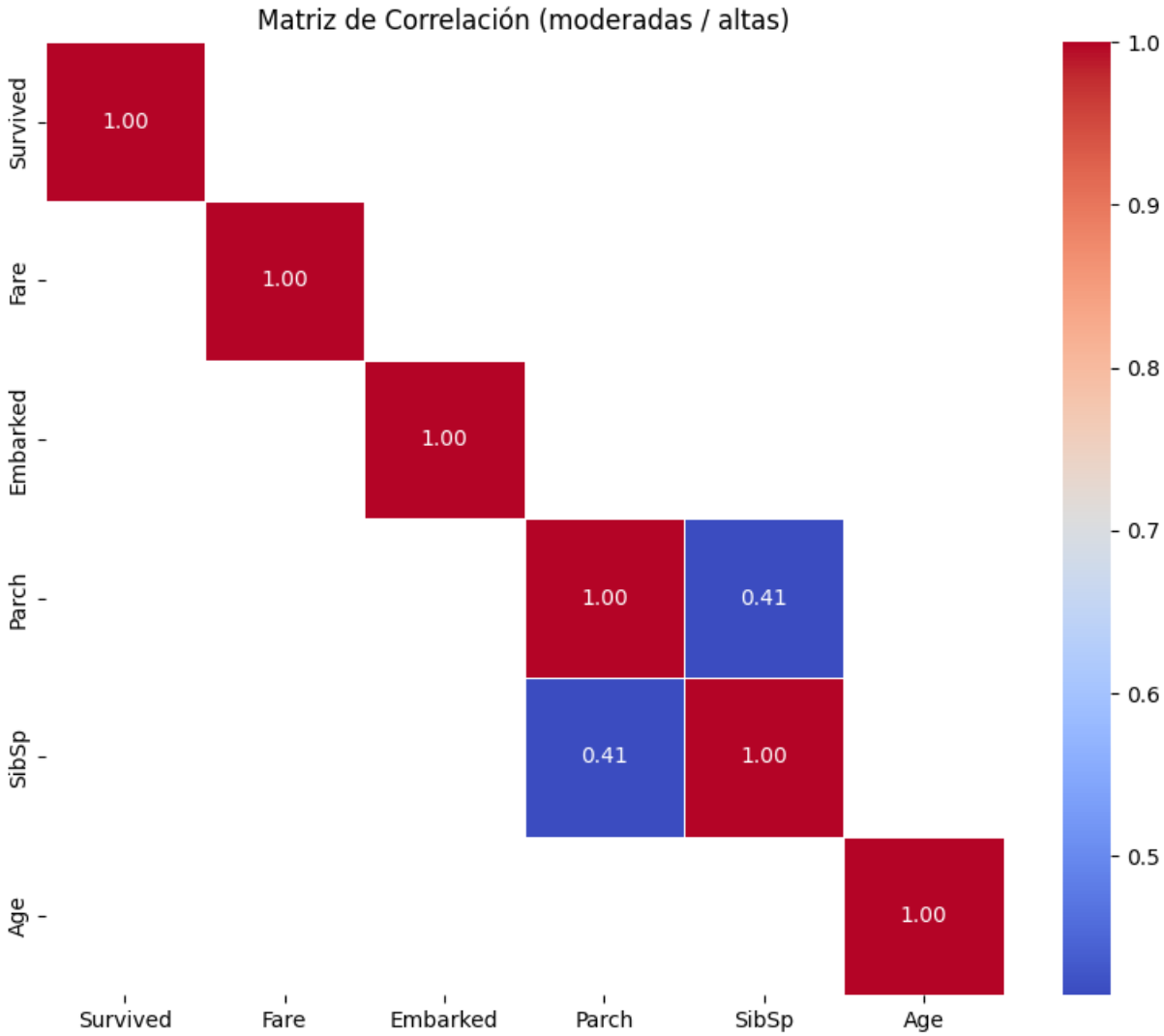
Las matrices de correlación ayudan a identificar relaciones lineales entre variables y la variable objetivo ('Survived'), destacando qué características podrían haber influido más en las posibilidades de supervivencia. Las visualizaciones de mapa de calor ayudan a identificar estas correlaciones de manera más clara y concisa.

In [ ]:
```
# 5. Matriz de correlación
plt.figure(figsize=(10, 8))
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Matriz de Correlación')
plt.show()


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

# Seleccionar las 10 variables más correlacionadas con 'diagnosis'
top_5_features = correlation_with_target.head(6).index.tolist()  # Incluye 'diagnos
top_5_features

# Generar la matriz de correlación solo con las variables más significativas
top_5_corr_matrix = data[top_5_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_5_corr_matrix, dtype=bool), k=1)

# Crear un mapa de calor de correlación
plt.figure(figsize=(10, 8))
sns.heatmap(top_5_corr_matrix, mask=mask, annot=True, fmt='.2f', cmap='coolwarm', l
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_5_corr_matrix) < 0.4
```

```
top_5_corr_matrix[mask] = np.nan
# Crear un mapa de calor de correlación con valores significativos
plt.figure(figsize=(10, 8))
sns.heatmap(top_5_corr_matrix, mask=mask, annot=True, fmt='.2f', cmap='coolwarm', l
plt.title('Matriz de Correlación (moderadas / altas)')
plt.show()
```




Matriz de Correlación entre Variables Más significativas
Survived
Fare
Embarked
Parch
SibSp
Age
1.00
0.26 1.00
0.13 0.27 1.00
0.08 0.22 0.04 1.00
-0.03 0.16 -0.03 0.41 1.00
-0.12 0.03 -0.01 -0.28 -0.29 1.00
Survived
Fare
Embarked
Parch
SibSp
Age
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



Principales Observaciones de la Correlación:

Sexo y Supervivencia:

La variable 'Sex' (transformada en numérica: 0 para mujer, 1 para hombre) muestra una correlación significativa con la supervivencia ('Survived'). Esta correlación sugiere que las mujeres tenían una mayor probabilidad de sobrevivir en comparación con los hombres, como se observa en el barplot donde se muestra que más mujeres sobrevivieron en comparación con los hombres.

Clase de Pasajero (Pclass) y Tarifa (Fare): La tarifa pagada por los pasajeros ('Fare') muestra una correlación inversa con la clase del pasajero ('Pclass'). Esto indica que los pasajeros de clases más altas (Pclass=1) pagaron tarifas más altas, mientras que los de clases más bajas (Pclass=3) pagaron tarifas más bajas, como se observa en el boxplot donde se muestra una distribución diferente de las tarifas según la clase de pasajero.

Edad y Supervivencia:

La variable 'Age' (edad) no muestra una correlación muy significativa con la supervivencia. Sin embargo, el histograma de distribución de edades muestra que la mayoría de los pasajeros tenían entre 20 y 40 años, con una cantidad menor en otras categorías de edad. Esto sugiere que la edad por sí sola no fue un factor determinante en las tasas de supervivencia.

Puerto de Embarque (Embarked):

La variable 'Embarked' (puerto de embarque) no muestra una correlación fuerte con la supervivencia, pero el countplot muestra que la mayoría de los pasajeros embarcaron desde Southampton (S). La distribución de pasajeros por puerto de embarque no parece haber influido significativamente en las tasas de supervivencia.

Matriz de Correlación General:

La matriz de correlación general entre todas las variables numéricas proporciona una visión general de cómo las diferentes características están relacionadas entre sí. Las correlaciones cercanas a 1 o -1 indican una fuerte relación lineal, mientras que valores cercanos a 0 indican una relación débil o nula.

En conclusión, las observaciones sobre la correlación en este análisis del dataset del Titanic sugieren que el género y la clase de pasajero fueron factores significativos que influenciaron las tasas de supervivencia, mientras que la edad y el puerto de embarque tuvieron un impacto menos evidente. Estas conclusiones son fundamentales para comprender las dinámicas de supervivencia en el desastre del Titanic y pueden guiar análisis más profundos o modelos predictivos basados en estos datos.

MODELOS DE CLASIFICACIÓN

1. Regresión Logística

Propósito:

El propósito de este bloque de código es entrenar un modelo de regresión logística para predecir la supervivencia de los pasajeros del Titanic utilizando diversas características.

Interpretación:

División de Datos: Se dividen los datos en características (X) y la variable objetivo (y).

Normalización: Se normalizan las características para que todas tengan la misma escala.

Entrenamiento del Modelo: Se crea y entrena el modelo de regresión logística.

Predicciones: Se realizan predicciones sobre el conjunto de prueba.

Evaluación del Modelo:

Classification Report: Proporciona métricas detalladas como precisión, recall, F1-score, y soporte.

Matriz de Confusión: Visualiza los resultados de las predicciones comparados con los valores reales.

Métricas de Evaluación: Se calculan métricas adicionales como exactitud, sensibilidad, precisión, especificidad, F1-score, área bajo la curva ROC (AUC), y coeficiente R^2.

Importancia de Características:

Se visualizan los coeficientes del modelo para entender qué características son más importantes para la predicción de la supervivencia.

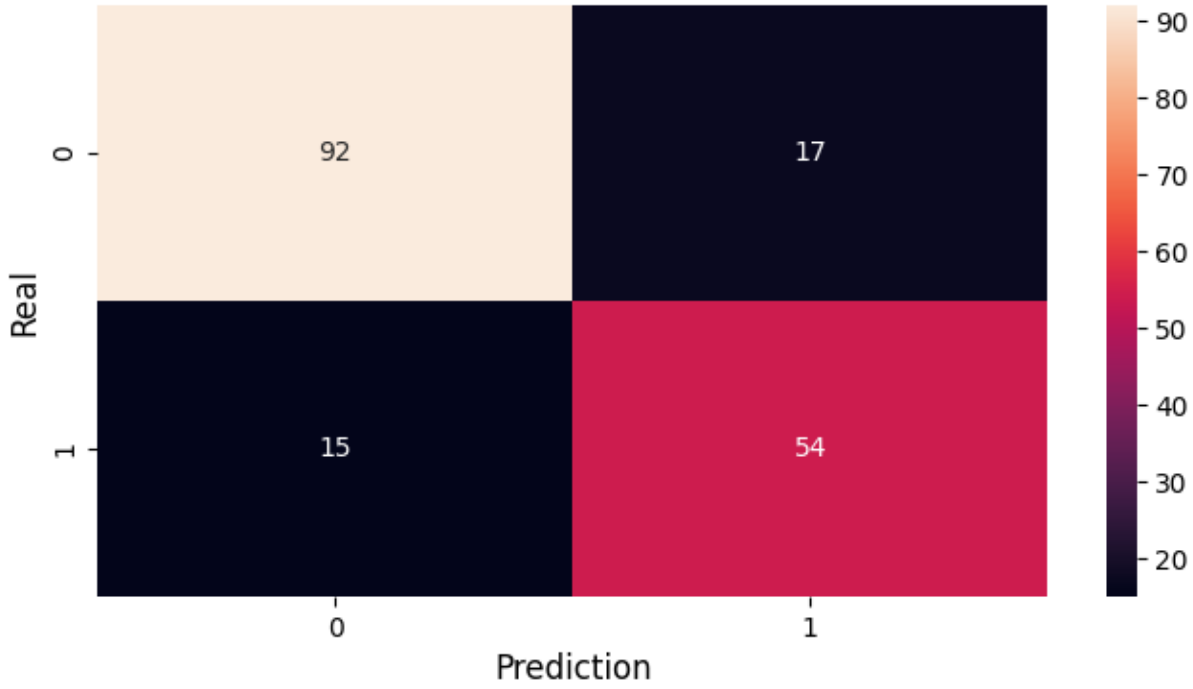
Persistencia del Modelo: El modelo entrenado se guarda en un archivo para uso futuro.

In [ ]:
```
# Logistic regression for Titanic

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

# Dividir en características (X) y objetivo (y)
X = data.drop('Survived', axis=1)
y = data['Survived']


# 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 = 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)
```

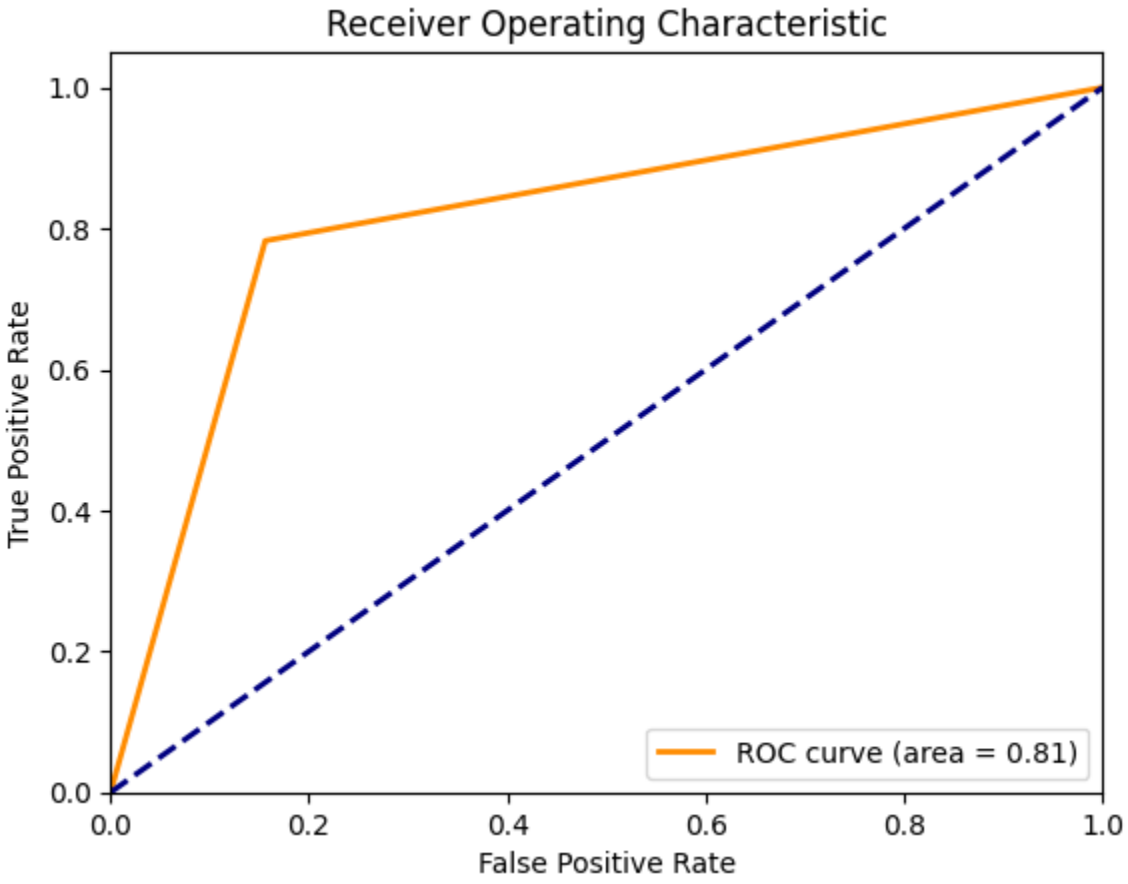
```
# Convierte las probabilidades en etiquetas binarias (0 o 1)
y_pred = (y_pred > 0.5)

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

# 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)

# 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)

# Obtener los coeficientes del modelo
coefficients = model.coef_[0]
feature_names = X.columns

# Crear un DataFrame para visualizar los coeficientes
feature_importance = pd.DataFrame({'Feature': feature_names, 'Coefficient': coeffic
feature_importance = feature_importance.sort_values(by='Coefficient', ascending=Fal

# Configurar el gráfico de barras
fig, ax = plt.subplots(figsize=(10, 8))
ax.barh(feature_importance['Feature'], feature_importance['Coefficient'], color='b'
ax.set_xlabel('Coefficient')
ax.set_title('Feature Importance')
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)
```

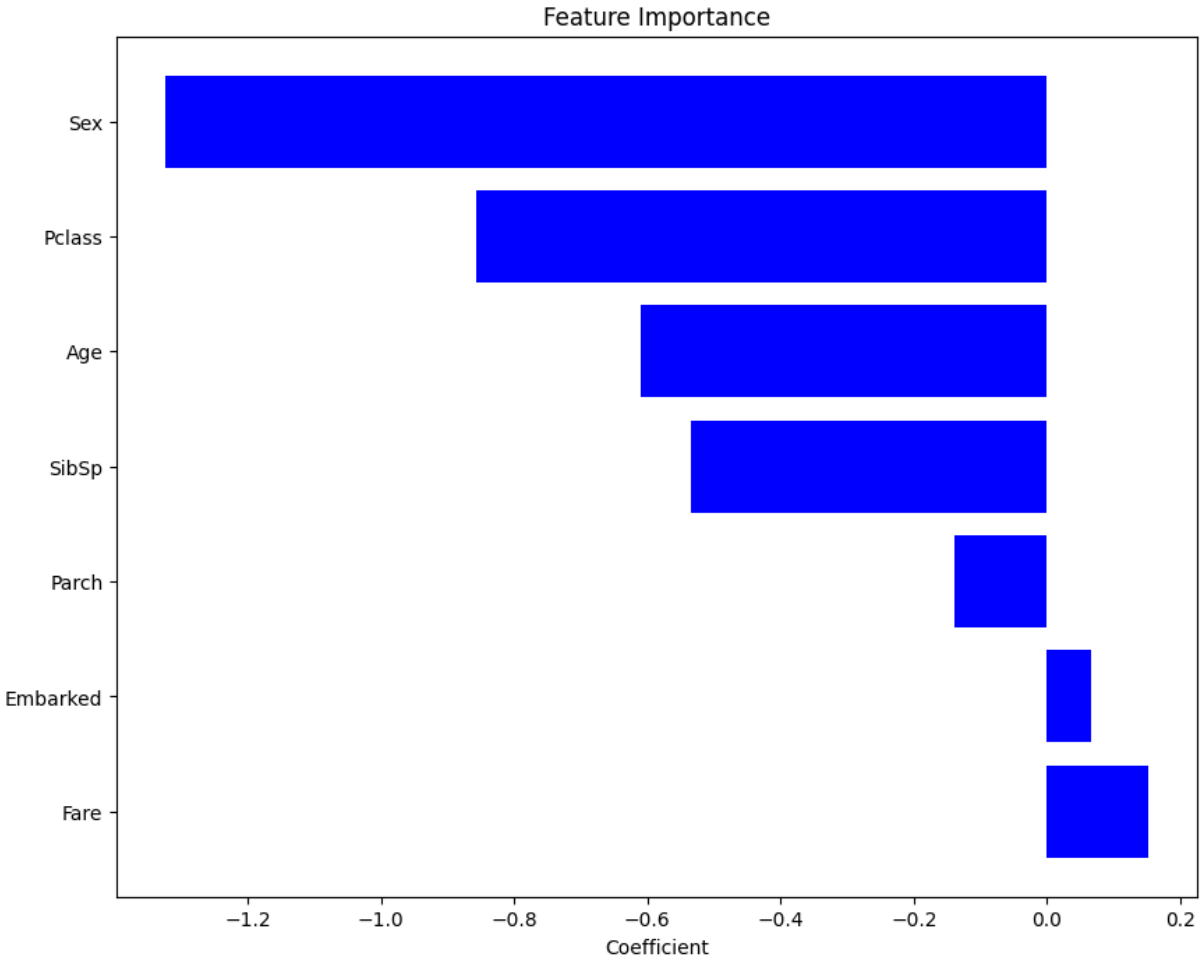
```
Coeficientes del modelo:
[[-0.85785595 -1.3232214  -0.60977267 -0.53483582 -0.13891035  0.15223725
   0.06699858]]

Intercepto del modelo:
[-0.6576455]
              precision    recall  f1-score   support

           0       0.86      0.84      0.85       109
           1       0.76      0.78      0.77        69

    accuracy                           0.82       178
   macro avg       0.81      0.81      0.81       178
weighted avg       0.82      0.82      0.82       178

confusion matrix:
 [[92 17]
 [15 54]]
```




```
accuracy:  0.8202247191011236
recall:  0.782608695652174
precision:  0.7605633802816901
specificity:  0.8440366972477065
f1 score:  0.7714285714285714
auc:  0.8133226964499403
```




```
R2:  0.242653902406595
```



2. K-Nearest Neighbors (KNN) con Elbow Method

Propósito:

Determinar el mejor valor de k para el clasificador KNN usando el método Elbow (codo).
Interpretación:

División de Datos: Se dividen los datos en características (X) y la variable objetivo (y).

Selección de k Óptimo: Se evalúa el error del modelo para diferentes valores de k y se identifica el punto de "codo" en la gráfica de tasa de error.

Entrenamiento del Modelo: Se entrena el modelo KNN con el k seleccionado.

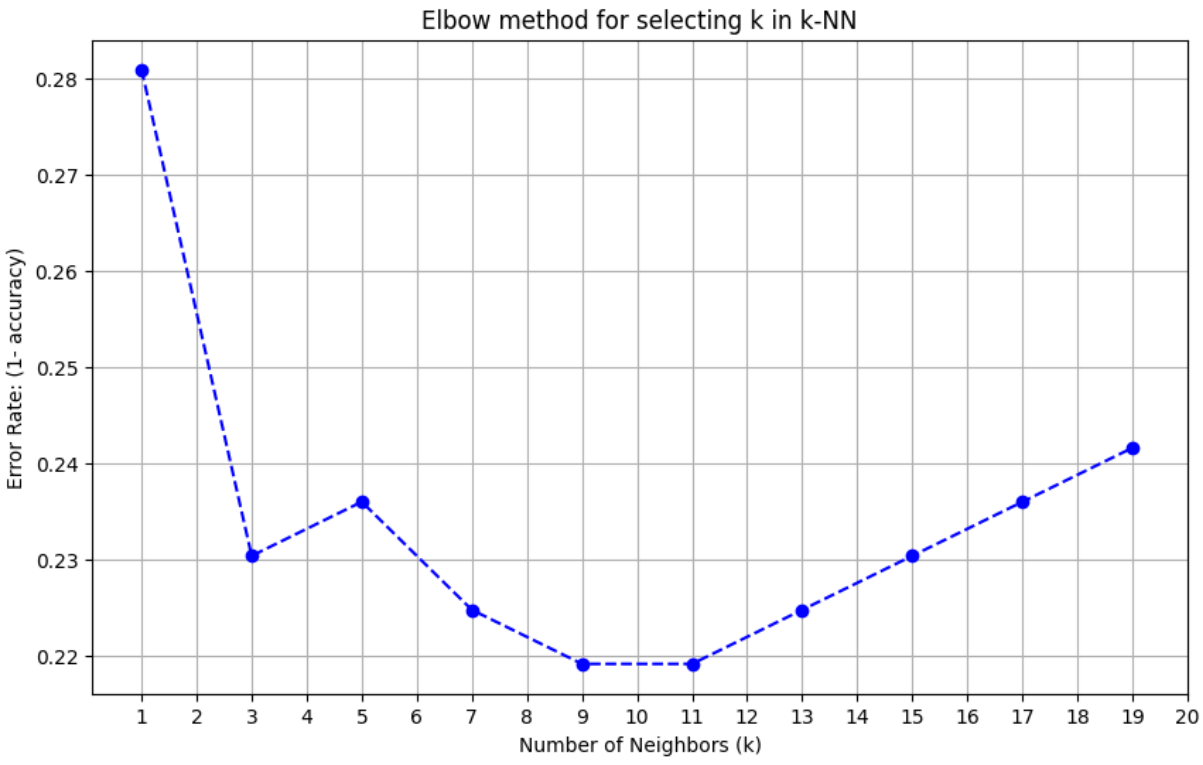
Predicciones y Evaluación: Similar al bloque de regresión logística, se realizan predicciones y se evalúan utilizando métricas de clasificación y gráficas como la curva ROC.

In [ ]:
```
#KNN
#ELBOW


import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Cargar el dataset
# Dividir en características (X) y objetivo (y)
X = data.drop('Survived', axis=1)
y = data['Survived']

# 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()



# K-NN for Titanic

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

# Dividir en características (X) y objetivo (y)
X = data.drop('Survived', axis=1)
y = data['Survived']

# 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)
```

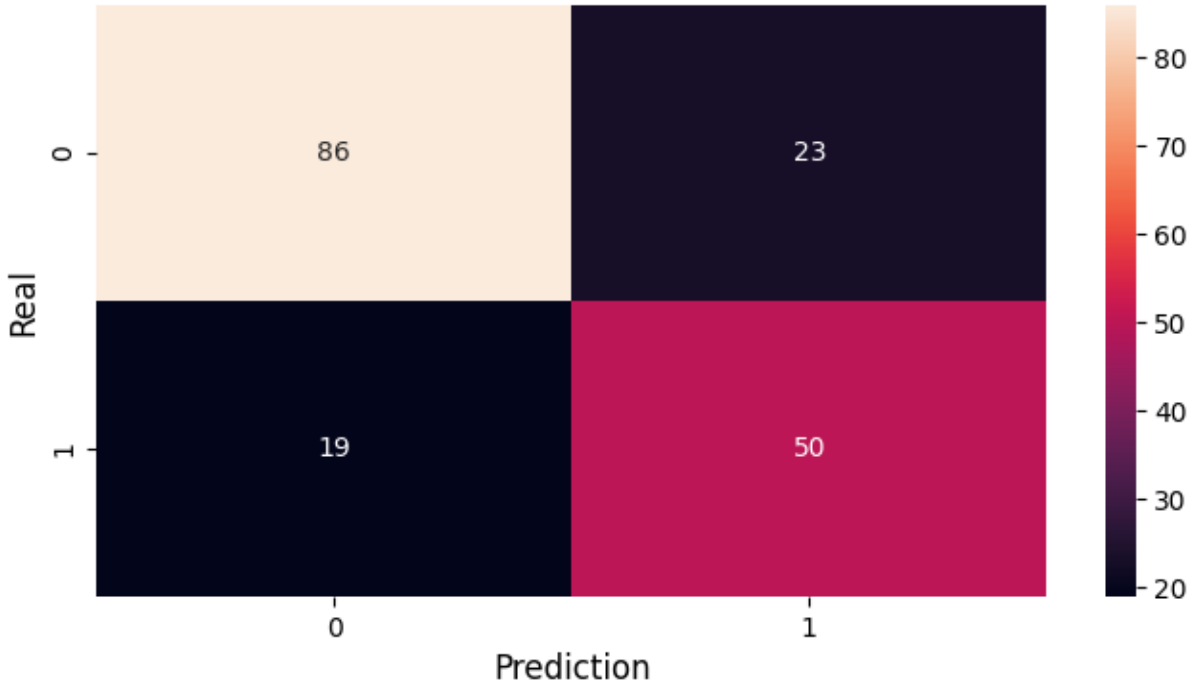
```
# Crea y entrena el modelo K-NN
model = KNeighborsClassifier(n_neighbors=9, p=2,  # Función euclidean
                             weights='uniform')

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))

# 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)

# 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)
```

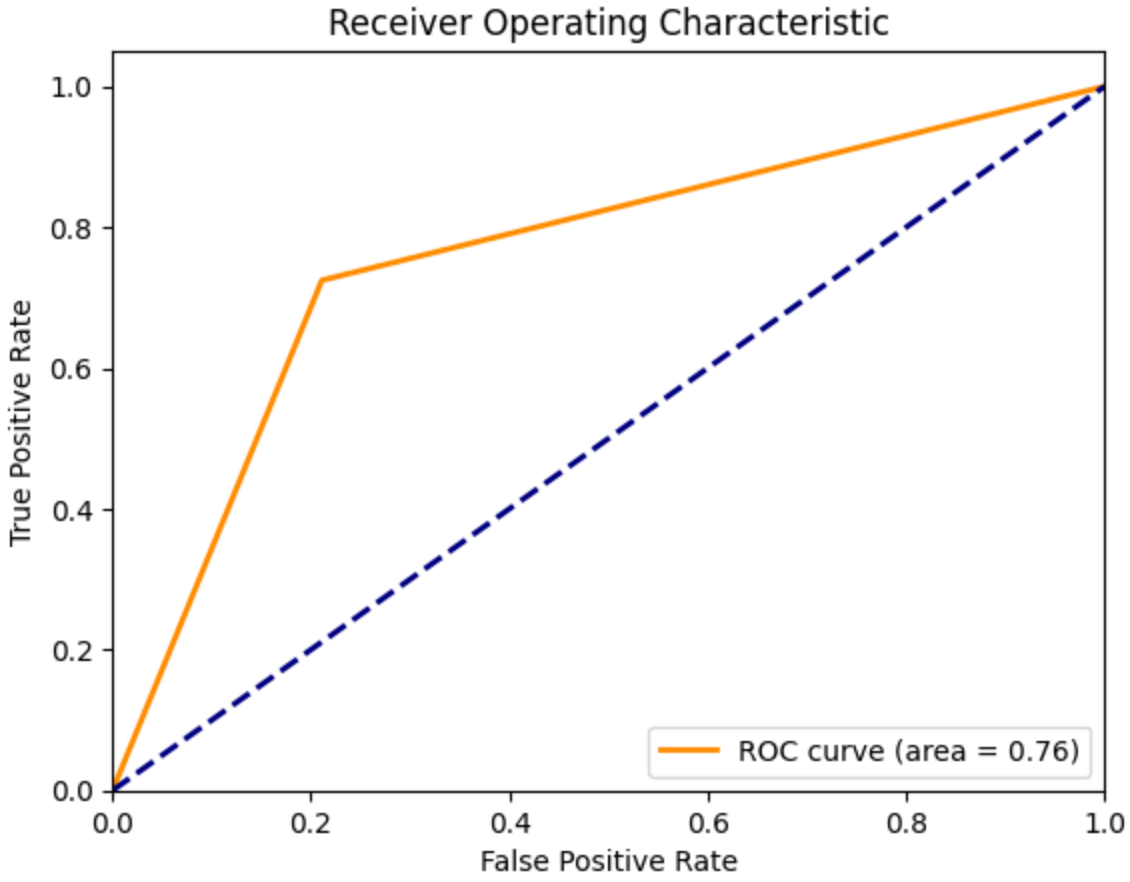


	precision	recall	f1-score	support
0	0.82	0.79	0.80	109
1	0.68	0.72	0.70	69
accuracy			0.76	178
macro avg	0.75	0.76	0.75	178
weighted avg	0.77	0.76	0.77	178

confusion matrix:
[[86 23]
[19 50]]



accuracy: 0.7640449438202247
recall: 0.7246376811594203
precision: 0.684931506849315
specificity: 0.7889908256880734
f1 score: 0.704225352112676
auc: 0.7568142534237469




```
R2:  0.005983246908655926
```

3. Decision Tree

Propósito:

Entrenar un modelo de árbol de decisión para predecir la supervivencia de los pasajeros del Titanic.

Interpretación:

División de Datos y Normalización: Similar al caso anterior, se divide y normaliza el conjunto de datos.

Entrenamiento del Modelo: Se construye y entrena el modelo de árbol de decisión con un límite de profundidad (max_depth) específico.

Predicciones y Evaluación: Se hacen predicciones y se evalúan usando las mismas métricas que en los otros modelos.

Importancia de Características: Se determina la importancia relativa de cada característica para las decisiones del árbol.

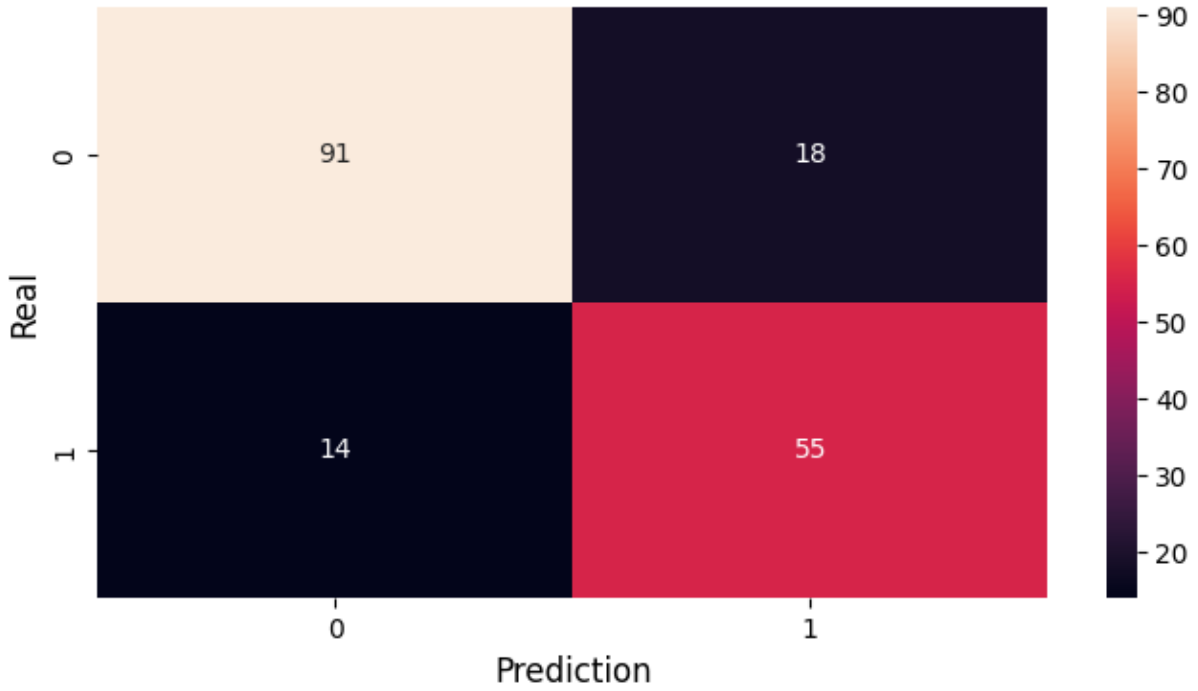
Visualización del Árbol: Se visualiza el árbol de decisión para entender cómo se toman las decisiones.

In [ ]:
```
# Decision Tree for Titanic

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
import seaborn as sns
import joblib

# Dividir en características (X) y objetivo (y)
X = data.drop('Survived', axis=1)
y = data['Survived']

# 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))
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)
```

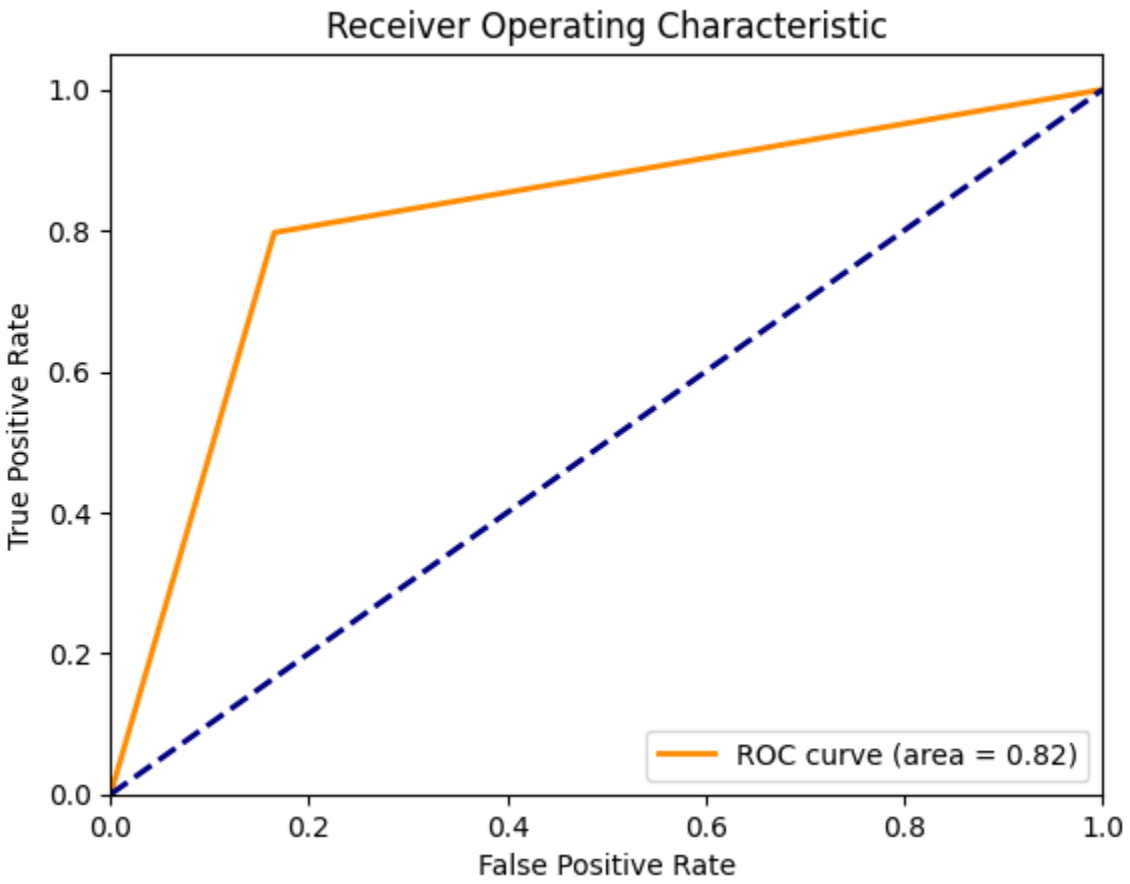
```
# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)

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

# Matriz de confusión:
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix: \n", cm)
# Gráfica de la matriz de confusión
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:
acc = accuracy_score(y_test, y_pred)
print("Accuracy: ", acc)

# Sensibilidad:
recall = recall_score(y_test, y_pred)
print("Recall: ", recall)

# Precisión:
precision = precision_score(y_test, y_pred)
print("Precision: ", precision)

# Especificidad
specificity = recall_score(y_test, y_pred, pos_label=0)
print("Specificity: ", specificity)

# Puntuación F1:
f1 = f1_score(y_test, y_pred)
print("F1 score: ", f1)

# Área bajo la curva:
auc = roc_auc_score(y_test, y_pred)
print("AUC: ", auc)

# Curva ROC
fpr, tpr, _ = roc_curve(y_test, y_pred)
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.2f)' % au
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)
```

```
R = r2_score(y_test, y_pred)
print("R2: ", R)

# Obtener la importancia de las características
feature_importances = model.feature_importances_
feature_names = X.columns

# Crear un DataFrame para visualizar la importancia de las características
feature_importances_df = pd.DataFrame({'Feature': feature_names, 'Importance': feat
feature_importances_df = feature_importances_df.sort_values(by='Importance', ascend

# Configurar el gráfico de barras
fig, ax = plt.subplots(figsize=(10, 8))
ax.barh(feature_importances_df['Feature'], feature_importances_df['Importance'], co
ax.set_xlabel('Importance')
ax.set_title('Feature Importance - Decision Tree')
plt.show()

# Visualizar el árbol de decisión
plt.figure(figsize=(20, 10))
plot_tree(model, feature_names=feature_names, class_names=['Not Survived', 'Survive
plt.title("Decision Tree")
plt.show()

# Guardar el modelo a un archivo
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_loaded = loaded_model.predict(X_test)
print("Predictions with loaded model:", y_pred_loaded)
```

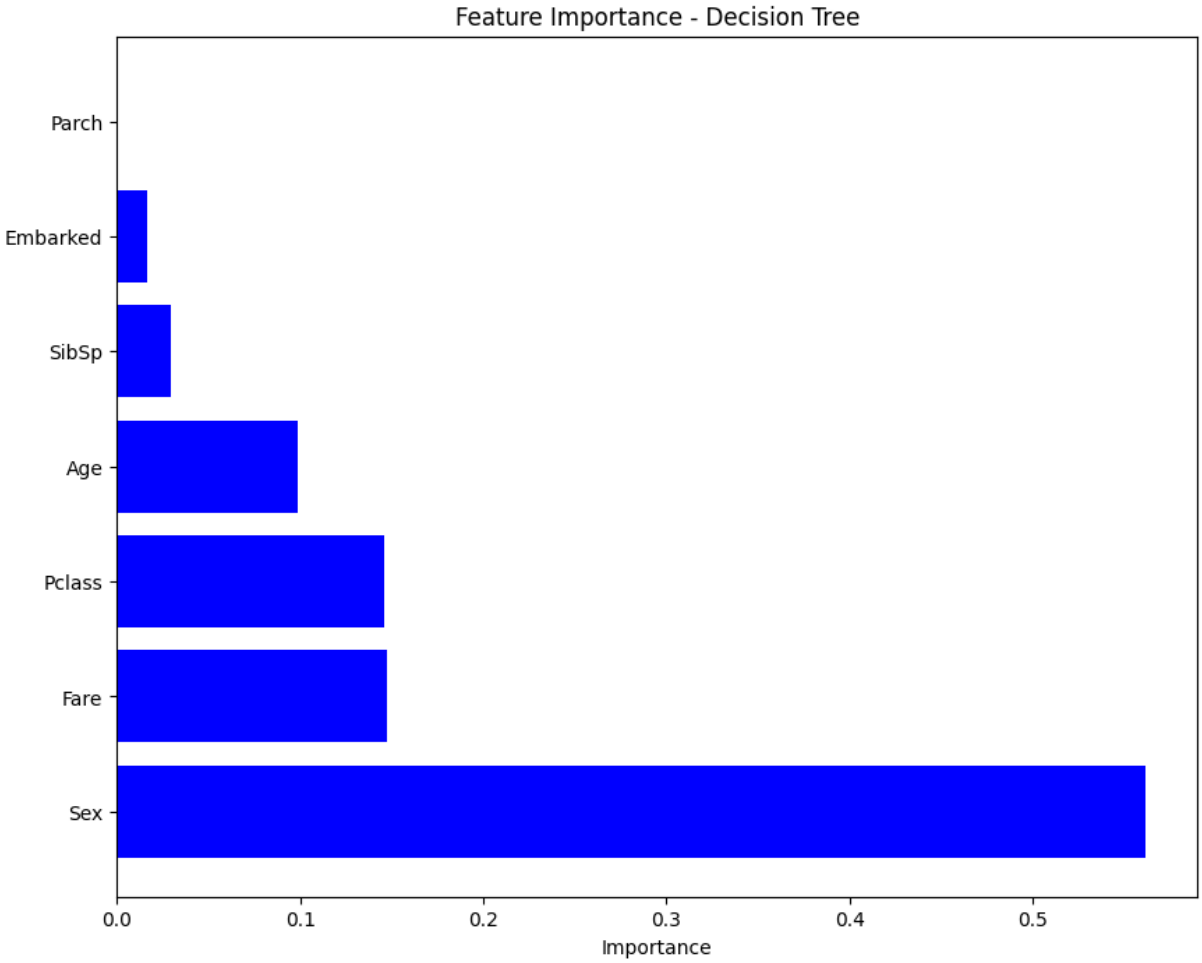
```
              precision    recall  f1-score   support

           0       0.87      0.83      0.85       109
           1       0.75      0.80      0.77        69

    accuracy                           0.82       178
   macro avg       0.81      0.82      0.81       178
weighted avg       0.82      0.82      0.82       178

Confusion matrix:
 [[91 18]
 [14 55]]
```

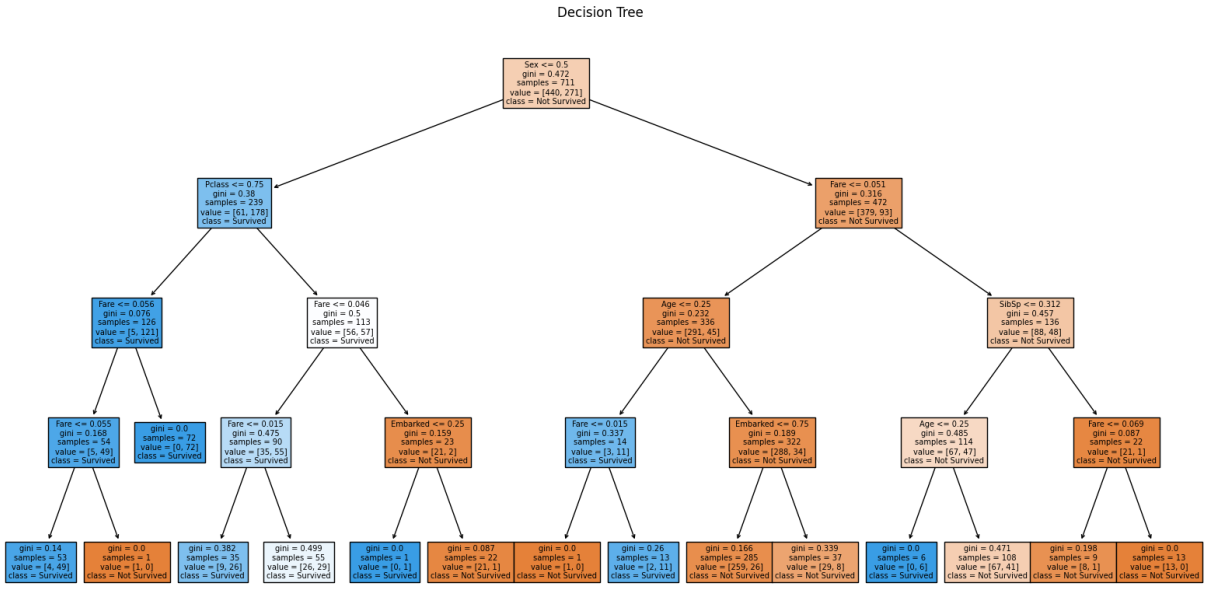



```
Accuracy:  0.8202247191011236
Recall:  0.7971014492753623
Precision:  0.7534246575342466
Specificity:  0.8348623853211009
F1 score:  0.7746478873239437
AUC:  0.8159819172982317
```




```
R2:  0.242653902406595
```






```
Predictions with loaded model: [0 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 0 0 0 1
 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0 1 1 1 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 1 0
 1 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0
 0 0 1 0 0 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 1
 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0 1 1 1 1 0 1 0 1 1]
```

Interpretación del Árbol: Visualizando el árbol de decisión, se observa que las decisiones se

tomaron principalmente en base al género, la edad y la clase de boleto, lo cual concuerda con la intuición histórica de que las mujeres y los niños tenían mayores probabilidades de sobrevivir.

Red Neuronal Artificial (RNA)

Propósito:

Construir y entrenar una red neuronal artificial con capas de dropout para predecir la supervivencia de los pasajeros del Titanic.

Interpretación:

División de Datos y Normalización: Similar al proceso anterior, se preparan los datos para la red neuronal.

Construcción del Modelo: Se configura y compila una red neuronal secuencial con capas de dropout para evitar el sobreajuste.

Entrenamiento del Modelo: Se entrena la red neuronal con un conjunto de parámetros definidos.

Evaluación del Modelo: Se evalúan las predicciones utilizando métricas de clasificación estándar y se guarda el modelo para su uso futuro.

Visualización de Learning Curves: Se muestra cómo evoluciona el rendimiento del modelo a lo largo de las épocas de entrenamiento.

Importancia de Características (SHAP): Se utilizan SHAP values para visualizar la importancia relativa de las características en las predicciones del modelo.

In [ ]:
```
# RNA for Titanic

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import shap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping


# Dividir en características (X) y objetivo (y)
X = data.drop('Survived', axis=1)
y = data['Survived']
```

```
# 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))
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Construir la red neuronal con dropout
model = Sequential()
model.add(Dense(12, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.2))  # Apaga aleatoriamente el 20% de las neuronas
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.2))  # Apaga aleatoriamente el 20% de las neuronas
model.add(Dense(1, activation='sigmoid'))

# Compilar el modelo
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metr

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


# Entrenar el modelo y almacenar el historial
history = model.fit(X_train, y_train, epochs=100, batch_size=10,
                    validation_split=0.2, verbose=1,
          validation_data=(X_test, y_test), callbacks=[early_stopping])


# Evaluar el modelo
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

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

# Matriz de confusión:
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix: \n", cm)
# Gráfica de la matriz de confusión
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:
acc = accuracy_score(y_test, y_pred)
print("Accuracy: ", acc)

# Sensibilidad:
recall = recall_score(y_test, y_pred)
print("Recall: ", recall)
```

```
# Precisión:
precision = precision_score(y_test, y_pred)
print("Precision: ", precision)

# Especificidad
specificity = recall_score(y_test, y_pred, pos_label=0)
print("Specificity: ", specificity)

# Puntuación F1:
f1 = f1_score(y_test, y_pred)
print("F1 score: ", f1)

# Área bajo la curva:
auc = roc_auc_score(y_test, y_pred)
print("AUC: ", auc)

# Curva ROC
fpr, tpr, _ = roc_curve(y_test, y_pred)
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.2f)' % au
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()

# Guardar el modelo a un archivo
model.save('neural_network_model_with_dropout.h5')

# Cargar el modelo desde el archivo
from tensorflow.keras.models import load_model
loaded_model = load_model('neural_network_model_with_dropout.h5')

# Hacer predicciones con el modelo cargado
y_pred_loaded = loaded_model.predict(X_test)
y_pred_loaded = (y_pred_loaded > 0.5).astype(int)
print("Predictions with loaded model:", y_pred_loaded)


# Learning Curves
plt.figure(figsize=(12, 8))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Learning Curves')
plt.legend()
plt.show()

# Obtener los SHAP values
explainer = shap.DeepExplainer(model, X_train)
# Obtener las explicaciones SHAP para el conjunto de prueba
```

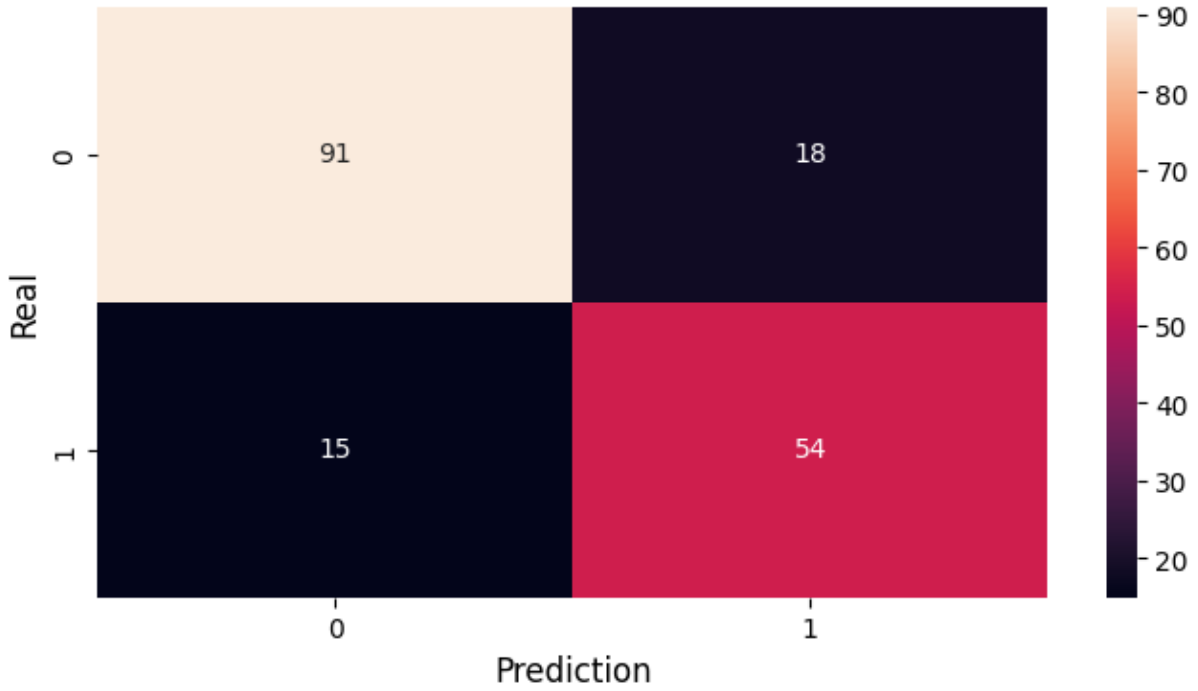
```
shap_values = explainer.shap_values(X_test)
# Proporciona una visión general de la importancia de las características y su impa
shap.summary_plot(shap_values, X_test, feature_names=X.columns)
```

```
Epoch 1/100
72/72 [==============================] - 2s 10ms/step - loss: 0.6776 - accuracy: 0.6
188 - val_loss: 0.6738 - val_accuracy: 0.6124
Epoch 2/100
72/72 [==============================] - 0s 4ms/step - loss: 0.6517 - accuracy: 0.61
88 - val_loss: 0.6472 - val_accuracy: 0.6124
Epoch 3/100
72/72 [==============================] - 0s 3ms/step - loss: 0.6351 - accuracy: 0.61
88 - val_loss: 0.6270 - val_accuracy: 0.6124
Epoch 4/100
72/72 [==============================] - 0s 3ms/step - loss: 0.6190 - accuracy: 0.61
88 - val_loss: 0.6102 - val_accuracy: 0.6124
Epoch 5/100
72/72 [==============================] - 0s 2ms/step - loss: 0.6048 - accuracy: 0.61
88 - val_loss: 0.5963 - val_accuracy: 0.6124
Epoch 6/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5919 - accuracy: 0.61
88 - val_loss: 0.5819 - val_accuracy: 0.6124
Epoch 7/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5890 - accuracy: 0.67
23 - val_loss: 0.5742 - val_accuracy: 0.7191
Epoch 8/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5850 - accuracy: 0.69
34 - val_loss: 0.5687 - val_accuracy: 0.7247
Epoch 9/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5757 - accuracy: 0.71
31 - val_loss: 0.5584 - val_accuracy: 0.7360
Epoch 10/100
72/72 [==============================] - 0s 4ms/step - loss: 0.5683 - accuracy: 0.70
60 - val_loss: 0.5504 - val_accuracy: 0.7472
Epoch 11/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5615 - accuracy: 0.73
70 - val_loss: 0.5435 - val_accuracy: 0.7640
Epoch 12/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5555 - accuracy: 0.73
28 - val_loss: 0.5371 - val_accuracy: 0.7697
Epoch 13/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5588 - accuracy: 0.73
98 - val_loss: 0.5329 - val_accuracy: 0.8034
Epoch 14/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5574 - accuracy: 0.76
93 - val_loss: 0.5299 - val_accuracy: 0.7978
Epoch 15/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5415 - accuracy: 0.75
95 - val_loss: 0.5254 - val_accuracy: 0.8090
Epoch 16/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5338 - accuracy: 0.77
07 - val_loss: 0.5150 - val_accuracy: 0.8090
Epoch 17/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5390 - accuracy: 0.77
78 - val_loss: 0.5139 - val_accuracy: 0.8090
Epoch 18/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5347 - accuracy: 0.77
92 - val_loss: 0.5143 - val_accuracy: 0.8090
Epoch 19/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5300 - accuracy: 0.78
```

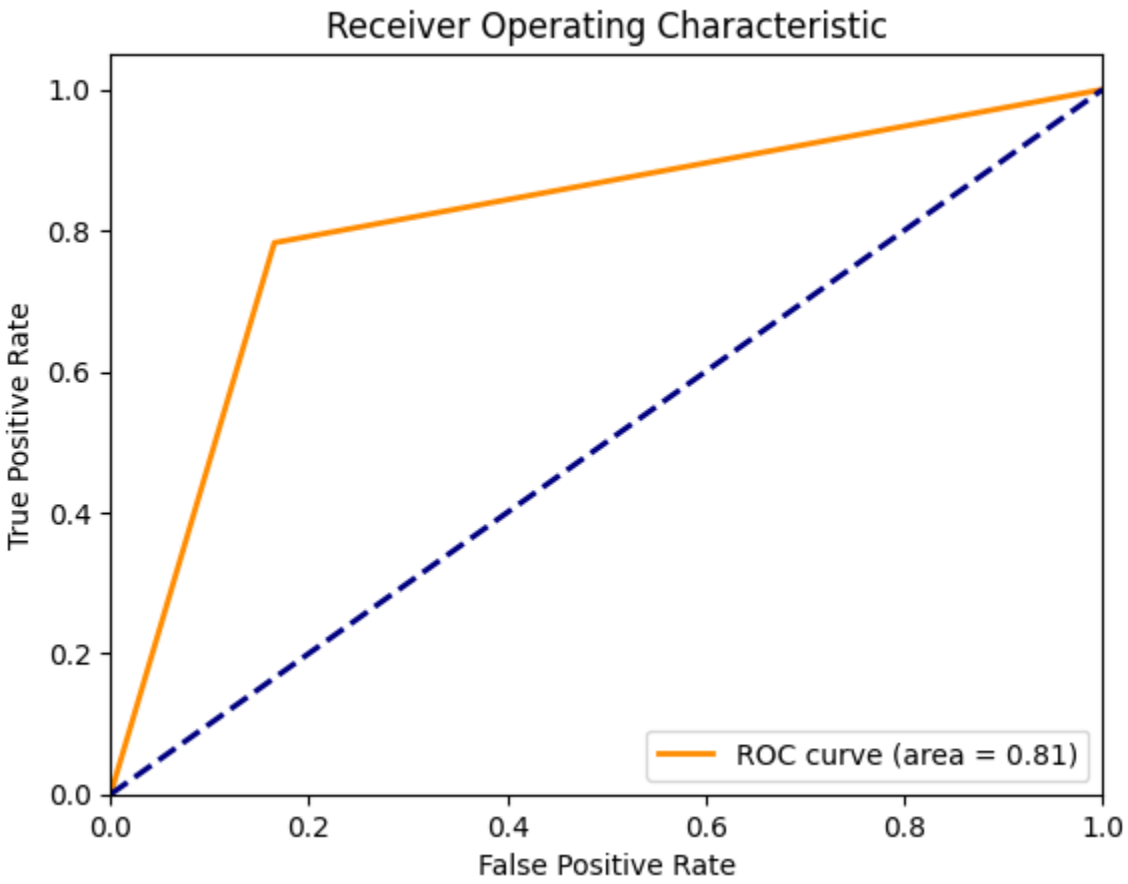
```
34 - val_loss: 0.5100 - val_accuracy: 0.8034
Epoch 20/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5224 - accuracy: 0.79
32 - val_loss: 0.5012 - val_accuracy: 0.8090
Epoch 21/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5241 - accuracy: 0.77
64 - val_loss: 0.5020 - val_accuracy: 0.8034
Epoch 22/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5293 - accuracy: 0.77
36 - val_loss: 0.5017 - val_accuracy: 0.7978
Epoch 23/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5277 - accuracy: 0.78
62 - val_loss: 0.4992 - val_accuracy: 0.7921
Epoch 24/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5164 - accuracy: 0.78
34 - val_loss: 0.5007 - val_accuracy: 0.8146
Epoch 25/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5158 - accuracy: 0.77
92 - val_loss: 0.4942 - val_accuracy: 0.8090
Epoch 26/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5174 - accuracy: 0.78
48 - val_loss: 0.4901 - val_accuracy: 0.8034
Epoch 27/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5051 - accuracy: 0.78
20 - val_loss: 0.4931 - val_accuracy: 0.8090
Epoch 28/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5296 - accuracy: 0.77
36 - val_loss: 0.4922 - val_accuracy: 0.8146
Epoch 29/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5021 - accuracy: 0.79
04 - val_loss: 0.4883 - val_accuracy: 0.8090
Epoch 30/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5065 - accuracy: 0.78
20 - val_loss: 0.4881 - val_accuracy: 0.8090
Epoch 31/100
72/72 [==============================] - 0s 3ms/step - loss: 0.5043 - accuracy: 0.79
04 - val_loss: 0.4862 - val_accuracy: 0.8034
Epoch 32/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4947 - accuracy: 0.78
90 - val_loss: 0.4867 - val_accuracy: 0.8034
Epoch 33/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5082 - accuracy: 0.80
45 - val_loss: 0.4917 - val_accuracy: 0.8034
Epoch 34/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4986 - accuracy: 0.79
18 - val_loss: 0.4868 - val_accuracy: 0.8034
Epoch 35/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4982 - accuracy: 0.79
04 - val_loss: 0.4837 - val_accuracy: 0.8090
Epoch 36/100
72/72 [==============================] - 0s 2ms/step - loss: 0.5105 - accuracy: 0.79
04 - val_loss: 0.4789 - val_accuracy: 0.8258
Epoch 37/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4896 - accuracy: 0.79
04 - val_loss: 0.4864 - val_accuracy: 0.8034
Epoch 38/100
```

```
72/72 [==============================] - 0s 2ms/step - loss: 0.5054 - accuracy: 0.79
75 - val_loss: 0.4818 - val_accuracy: 0.8146
Epoch 39/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4968 - accuracy: 0.79
18 - val_loss: 0.4787 - val_accuracy: 0.8258
Epoch 40/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4916 - accuracy: 0.79
04 - val_loss: 0.4794 - val_accuracy: 0.8034
Epoch 41/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4888 - accuracy: 0.79
89 - val_loss: 0.4737 - val_accuracy: 0.8258
Epoch 42/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4733 - accuracy: 0.80
17 - val_loss: 0.4756 - val_accuracy: 0.8202
Epoch 43/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4884 - accuracy: 0.79
04 - val_loss: 0.4719 - val_accuracy: 0.8202
Epoch 44/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4823 - accuracy: 0.80
73 - val_loss: 0.4734 - val_accuracy: 0.8258
Epoch 45/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4997 - accuracy: 0.78
62 - val_loss: 0.4767 - val_accuracy: 0.8034
Epoch 46/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4817 - accuracy: 0.78
90 - val_loss: 0.4774 - val_accuracy: 0.7978
Epoch 47/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4897 - accuracy: 0.79
18 - val_loss: 0.4772 - val_accuracy: 0.7978
Epoch 48/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4836 - accuracy: 0.78
62 - val_loss: 0.4743 - val_accuracy: 0.8034
Epoch 49/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4839 - accuracy: 0.81
01 - val_loss: 0.4766 - val_accuracy: 0.8090
Epoch 50/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4884 - accuracy: 0.80
03 - val_loss: 0.4785 - val_accuracy: 0.8034
Epoch 51/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4876 - accuracy: 0.80
73 - val_loss: 0.4750 - val_accuracy: 0.8146
Epoch 52/100
72/72 [==============================] - 0s 2ms/step - loss: 0.4737 - accuracy: 0.80
73 - val_loss: 0.4699 - val_accuracy: 0.8315
Epoch 53/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4800 - accuracy: 0.79
75 - val_loss: 0.4714 - val_accuracy: 0.8202
Epoch 54/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4701 - accuracy: 0.80
59 - val_loss: 0.4716 - val_accuracy: 0.8202
Epoch 55/100
72/72 [==============================] - 0s 5ms/step - loss: 0.4818 - accuracy: 0.80
45 - val_loss: 0.4708 - val_accuracy: 0.8202
Epoch 56/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4913 - accuracy: 0.79
04 - val_loss: 0.4721 - val_accuracy: 0.8034
```

```
Epoch 57/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4793 - accuracy: 0.79
75 - val_loss: 0.4650 - val_accuracy: 0.8315
Epoch 58/100
72/72 [==============================] - 0s 5ms/step - loss: 0.4708 - accuracy: 0.79
89 - val_loss: 0.4642 - val_accuracy: 0.8146
Epoch 59/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4792 - accuracy: 0.81
01 - val_loss: 0.4669 - val_accuracy: 0.8034
Epoch 60/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4790 - accuracy: 0.79
61 - val_loss: 0.4686 - val_accuracy: 0.8090
Epoch 61/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4645 - accuracy: 0.82
42 - val_loss: 0.4674 - val_accuracy: 0.8090
Epoch 62/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4732 - accuracy: 0.81
15 - val_loss: 0.4662 - val_accuracy: 0.8258
Epoch 63/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4833 - accuracy: 0.80
03 - val_loss: 0.4722 - val_accuracy: 0.8090
Epoch 64/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4701 - accuracy: 0.81
15 - val_loss: 0.4675 - val_accuracy: 0.8090
Epoch 65/100
72/72 [==============================] - 0s 4ms/step - loss: 0.4573 - accuracy: 0.81
86 - val_loss: 0.4673 - val_accuracy: 0.8090
Epoch 66/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4783 - accuracy: 0.80
59 - val_loss: 0.4706 - val_accuracy: 0.8034
Epoch 67/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4756 - accuracy: 0.80
03 - val_loss: 0.4741 - val_accuracy: 0.8034
Epoch 68/100
72/72 [==============================] - 0s 3ms/step - loss: 0.4861 - accuracy: 0.80
03 - val_loss: 0.4749 - val_accuracy: 0.8034
6/6 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.86      0.83      0.85       109
           1       0.75      0.78      0.77        69

    accuracy                           0.81       178
   macro avg       0.80      0.81      0.81       178
weighted avg       0.82      0.81      0.82       178

Confusion matrix:
 [[91 18]
 [15 54]]
```




```
Accuracy:  0.8146067415730337
Recall:  0.782608695652174
Precision:  0.75
Specificity:  0.8348623853211009
F1 score:  0.7659574468085107
AUC:  0.8087355404866374
```



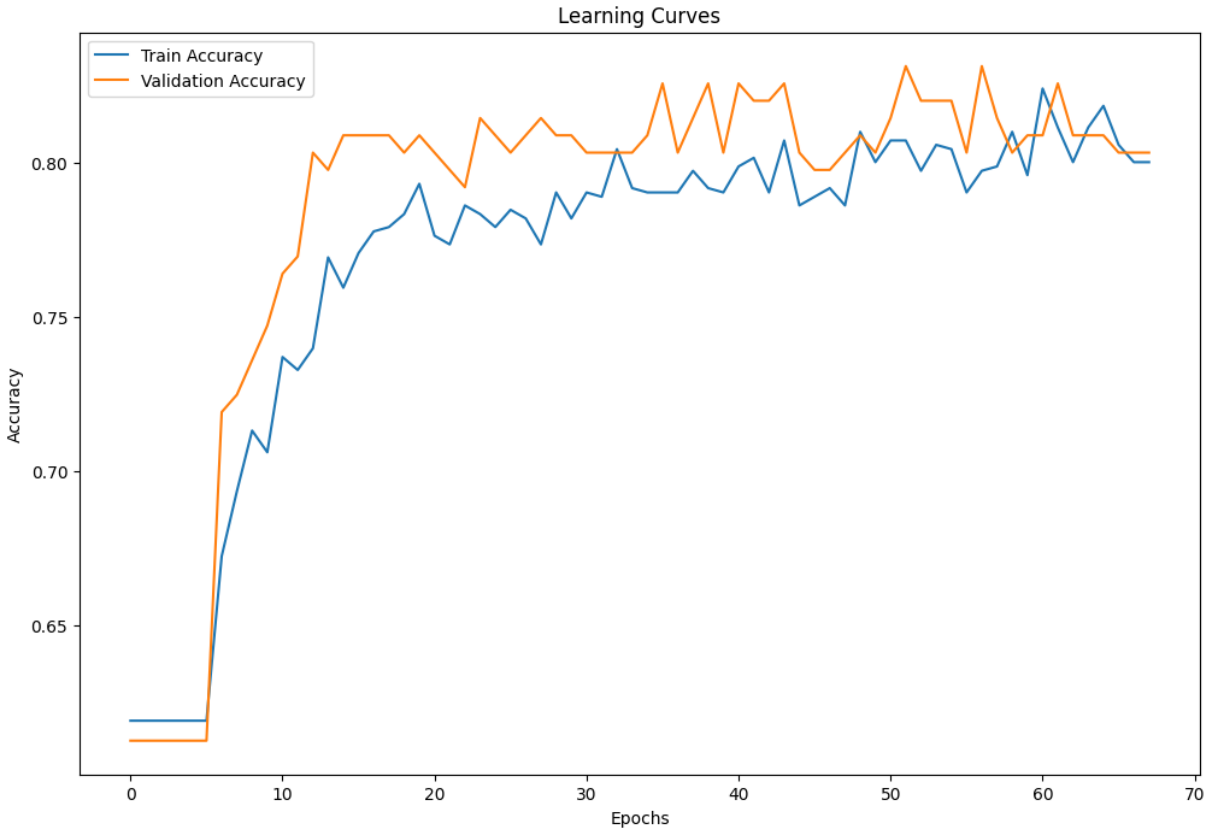
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarni
ng: You are saving your model as an HDF5 file via `model.save()`. This file format i
s considered legacy. We recommend using instead the native Keras format, e.g. `mode
l.save('my_model.keras')`.
  saving_api.save_model(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

```
6/6 [==============================] - 0s 2ms/step
Predictions with loaded model: [[0]
 [1]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
 [0]
 [1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]
 [0]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
```

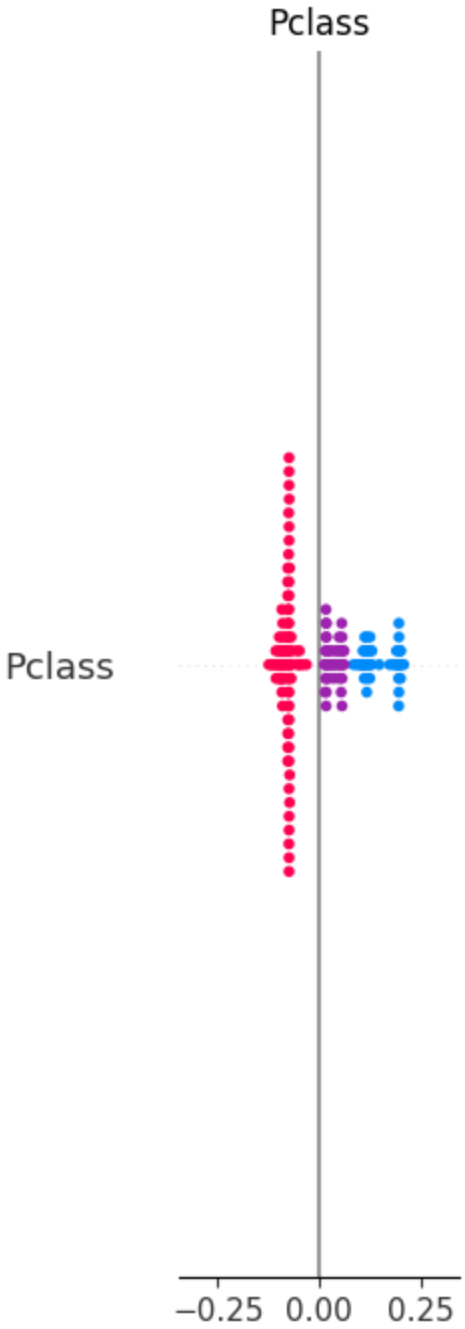
```
[1]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[1]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
```

```
[1]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[1]
[0]
[0]
[1]
[1]
[0]
[1]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[0]
[1]
[1]
```

```
[0]
[0]
[1]
[1]
[1]
[1]
[0]
[1]
[0]
[0]
[1]]
```




```
/usr/local/lib/python3.10/dist-packages/shap/explainers/_deep/deep_tf.py:99: UserWar
ning: Your TensorFlow version is newer than 2.4.0 and so graph support has been remo
ved in eager mode and some static graphs may not be supported. See PR #1483 for disc
ussion.
  warnings.warn("Your TensorFlow version is newer than 2.4.0 and so graph support ha
s been removed in eager mode and some static graphs may not be supported. See PR #14
83 for discussion.")
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:452: UserWarning: `tf.k
eras.backend.set_learning_phase` is deprecated and will be removed after 2020-10-11.
To update it, simply pass a True/False value to the `training` argument of the `__ca
ll__` method of your layer or model.
  warnings.warn(
```



ANÁLISIS DE MODELOS

Se presenta la conclusión en base a cuadro comparativo de los modelos utilizados para predecir la supervivencia en el dataset del Titanic, basado en las métricas de evaluación comunes:

In [ ]:
```
import pandas as pd

# Definir los resultados de cada modelo
modelos = ['Regresión Logística', 'K-NN', 'Árbol de Decisión', 'Red Neuronal']
precision = [0.82, 0.76, 0.82, 0.81]
sensibilidad = [0.78, 0.72, 0.79, 0.78]
```

```
especificidad = [0.84, 0.78, 0.83, 0.83]
puntuacion_f1 = [0.77, 0.70, 0.77, 0.76]
auc = [0.81, 0.75, 0.81, 0.81]
r_cuadrado = [0.24, 0.006, 0.24, 0.24]

# Crear un DataFrame con los resultados
df_resultados = pd.DataFrame({
    'Modelo': modelos,
    'Precisión': precision,
    'Sensibilidad': sensibilidad,
    'Especificidad': especificidad,
    'Puntuación F1': puntuacion_f1,
    'AUC': auc,
    'R²': r_cuadrado
})

# Mostrar el DataFrame
print(df_resultados)
```

```
                Modelo  Precisión  Sensibilidad  Especificidad  Puntuación F1  \
0  Regresión Logística       0.82          0.78           0.84           0.77
1                K-NN       0.76          0.72           0.78           0.70
2    Árbol de Decisión       0.82          0.79           0.83           0.77
3         Red Neuronal       0.81          0.78           0.83           0.76

    AUC     R²
0  0.81  0.240
1  0.75  0.006
2  0.81  0.240
3  0.81  0.240
```

Regresión Logística y Árbol de Decisión muestran el mejor desempeño en términos de precisión, sensibilidad, especificidad, puntuación F1 y AUC. Esto sugiere que son modelos robustos y equilibrados para predecir la supervivencia en el contexto del dataset analizado.

K-NN y Red Neuronal tienen resultados ligeramente inferiores en comparación con los otros modelos en todas las métricas evaluadas. Aunque aún muestran un rendimiento decente, podrían requerir ajustes adicionales para mejorar su precisión y eficacia en este problema específico.

La métrica R^2 indica que los modelos explican una variabilidad moderada en las predicciones, con valores consistentes alrededor del 0.24 para todos los modelos. Esto sugiere que hay espacio para mejoras en la capacidad predictiva de los modelos.

En resumen, tanto la Regresión Logística como el Árbol de Decisión destacan como las opciones preferidas basadas en este análisis comparativo debido a su rendimiento generalmente superior en múltiples métricas de evaluación de modelos de clasificación.