In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle

from sklearn.metrics import accuracy_score, confusion_matrix, roc_
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_recall_curve

from imblearn.over_sampling import SMOTE
from sklearn.utils import resample
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
import tensorflow as tf
from tensorflow.keras.models import load_model
```

In [ ]:
```
df_covid = pd.read_csv('./Covid_clean.csv')
```

In [ ]:
```
#dropeamos lo que no me sirve como final clasification

df_covid.drop(['USMER', 'MEDICAL_UNIT', 'CLASIFFICATION_FINAL', 'D
```

In [ ]:
```
# reemplazo los valores 2 por 0 en todo el dataset

df_covid.replace(2, 0, inplace=True)

df_covid.head()
```

Out[ ]:

	SEX	PNEUMONIA	AGE	DIABETES	COPD	ASTHMA	INMSUPR	HIPERTEN
0	1	1.0	65.0	0.0	0.0	0.0	0.0	
1	0	1.0	72.0	0.0	0.0	0.0	0.0	
2	0	0.0	55.0	1.0	0.0	0.0	0.0	
3	1	0.0	53.0	0.0	0.0	0.0	0.0	
4	0	0.0	68.0	1.0	0.0	0.0	0.0	

In [ ]:
```
df_covid = pd.read_csv('./Covid_clean_modelo.csv')
```

In [ ]:
```
df_covid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1024829 entries, 0 to 1024828
Data columns (total 14 columns):
 #   Column          Non-Null Count    Dtype  
---  ------          --------------    -----  
 0   SEX             1024829 non-null  int64  
 1   PNEUMONIA       1024829 non-null  float64
 2   AGE             1024829 non-null  float64
 3   DIABETES        1024829 non-null  float64
 4   COPD            1024829 non-null  float64
 5   ASTHMA          1024829 non-null  float64
 6   INMSUPR         1024829 non-null  float64
 7   HIPERTENSION    1024829 non-null  float64
 8   OTHER_DISEASE   1024829 non-null  float64
 9   CARDIOVASCULAR  1024829 non-null  float64
 10  OBESITY         1024829 non-null  float64
 11  RENAL_CHRONIC   1024829 non-null  float64
 12  TOBACCO         1024829 non-null  float64
 13  fallecidos      1024829 non-null  int64  
dtypes: float64(12), int64(2)
memory usage: 109.5 MB
```

comparacion de modelos

Sin rebalanceo

In [ ]:
```
# creamos el modelo de clasificacion

X = df_covid.drop('fallecidos', axis=1)
y = df_covid['fallecidos']

# dividimos el dataset en train y test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
# red neuronal entrenar

model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_tr
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metric

model.fit(X_train, y_train, epochs=10, batch_size=32, validation_d
```

```
Epoch 1/10
25621/25621 [==============================] - 25s 961us/step - lo
ss: 0.1593 - accuracy: 0.9325 - val_loss: 0.1564 - val_accuracy:
0.9334
Epoch 2/10
25621/25621 [==============================] - 23s 884us/step - lo
ss: 0.1576 - accuracy: 0.9330 - val_loss: 0.1567 - val_accuracy:
0.9332
Epoch 3/10
25621/25621 [==============================] - 23s 879us/step - lo
ss: 0.1573 - accuracy: 0.9330 - val_loss: 0.1564 - val_accuracy:
0.9332
Epoch 4/10
25621/25621 [==============================] - 23s 895us/step - lo
ss: 0.1571 - accuracy: 0.9331 - val_loss: 0.1562 - val_accuracy:
0.9330
Epoch 5/10
25621/25621 [==============================] - 22s 872us/step - lo
ss: 0.1571 - accuracy: 0.9331 - val_loss: 0.1570 - val_accuracy:
0.9331
Epoch 6/10
25621/25621 [==============================] - 23s 881us/step - lo
ss: 0.1570 - accuracy: 0.9332 - val_loss: 0.1564 - val_accuracy:
0.9336
Epoch 7/10
25621/25621 [==============================] - 23s 883us/step - lo
ss: 0.1570 - accuracy: 0.9332 - val_loss: 0.1562 - val_accuracy:
0.9333
Epoch 8/10
25621/25621 [==============================] - 22s 857us/step - lo
ss: 0.1570 - accuracy: 0.9332 - val_loss: 0.1561 - val_accuracy:
0.9331
Epoch 9/10
25621/25621 [==============================] - 23s 897us/step - lo
ss: 0.1570 - accuracy: 0.9331 - val_loss: 0.1563 - val_accuracy:
0.9332
Epoch 10/10
25621/25621 [==============================] - 23s 890us/step - lo
ss: 0.1569 - accuracy: 0.9333 - val_loss: 0.1564 - val_accuracy:
0.9333
```

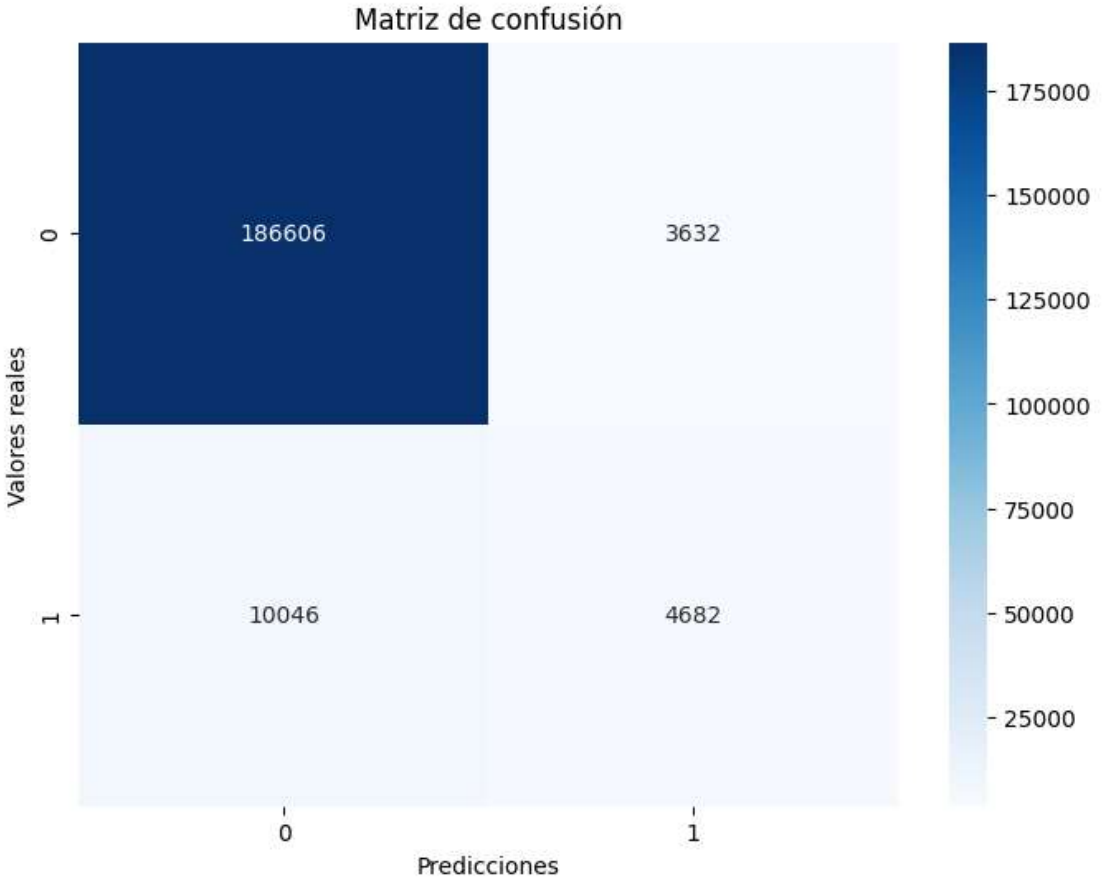
Out[ ]: <keras.callbacks.History at 0x16c2eecea10>

In [ ]:
```
#evaluamos el modelo

loss, accuracy = model.evaluate(X_test, y_test)
print('Loss:', loss)
print('Accuracy:', accuracy)
```

```
# Obtener las predicciones del modelo
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print('Precision:', precision)
print('Recall:', recall)
print('F1:', f1)
```

```
6406/6406 [==============================] - 4s 636us/step - loss:
0.1564 - accuracy: 0.9333
Loss: 0.1563640832901001
Accuracy: 0.9332669973373413
6406/6406 [==============================] - 3s 536us/step
Precision: 0.563146499879721
Recall: 0.3178978815860945
F1: 0.4063883343459769
```

In [ ]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(y_test, y_pred)

# Visualizar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicciones')
plt.ylabel('Valores reales')
plt.title('Matriz de confusión')
plt.show()
```



Rebalanceo

Sobremuestreo SMOTE

In [ ]:
```
# dvidimos en clase mayoritaria y minoritaria
clase_mayoritaria = df_covid[df_covid['fallecidos'] == 0]
clase_minoritaria = df_covid[df_covid['fallecidos'] == 1]

# upsampling de la clase minoritaria
smote = SMOTE(random_state=42)
features_balanceados, target_balanceados = smote.fit_resample(df_c

df_balanceado = pd.concat([pd.DataFrame(features_balanceados), pd.

df_balanceado['fallecidos'].value_counts()
```

Out[ ]:
```
1    950217
0    950217
Name: fallecidos, dtype: int64
```

In [ ]:
```
# creamos el modelo de clasificacion
X = df_balanceado.drop('fallecidos', axis=1)
```

```
y = df_balanceado['fallecidos']
```

In [ ]:

```
# dividimos el dataset en train y test
X_train, X_test_smote, y_train, y_test_smote = train_test_split(X,



# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test_smote = scaler.transform(X_test_smote)

# red neuronal entrenar
model_smote = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_tr
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_smote.compile(optimizer='adam', loss='binary_crossentropy',

model_smote.fit(X_train, y_train, epochs=10, batch_size=32, valida
```

```
Epoch 1/10
47511/47511 [==============================] - 42s 875us/step - lo
ss: 0.3394 - accuracy: 0.8594 - val_loss: 0.3331 - val_accuracy:
0.8614
Epoch 2/10
47511/47511 [==============================] - 41s 870us/step - lo
ss: 0.3300 - accuracy: 0.8627 - val_loss: 0.3281 - val_accuracy:
0.8627
Epoch 3/10
47511/47511 [==============================] - 41s 862us/step - lo
ss: 0.3269 - accuracy: 0.8638 - val_loss: 0.3270 - val_accuracy:
0.8630
Epoch 4/10
47511/47511 [==============================] - 41s 868us/step - lo
ss: 0.3252 - accuracy: 0.8645 - val_loss: 0.3241 - val_accuracy:
0.8647
Epoch 5/10
47511/47511 [==============================] - 41s 868us/step - lo
ss: 0.3239 - accuracy: 0.8650 - val_loss: 0.3239 - val_accuracy:
0.8651
Epoch 6/10
47511/47511 [==============================] - 42s 879us/step - lo
ss: 0.3229 - accuracy: 0.8653 - val_loss: 0.3215 - val_accuracy:
0.8658
Epoch 7/10
47511/47511 [==============================] - 41s 862us/step - lo
ss: 0.3220 - accuracy: 0.8659 - val_loss: 0.3249 - val_accuracy:
0.8646
Epoch 8/10
47511/47511 [==============================] - 42s 874us/step - lo
ss: 0.3213 - accuracy: 0.8660 - val_loss: 0.3202 - val_accuracy:
0.8663
Epoch 9/10
47511/47511 [==============================] - 42s 874us/step - lo
ss: 0.3208 - accuracy: 0.8664 - val_loss: 0.3202 - val_accuracy:
0.8663
Epoch 10/10
47511/47511 [==============================] - 42s 880us/step - lo
ss: 0.3203 - accuracy: 0.8666 - val_loss: 0.3199 - val_accuracy:
0.8664
```

Out[ ]: <keras.callbacks.History at 0x16c6271b310>

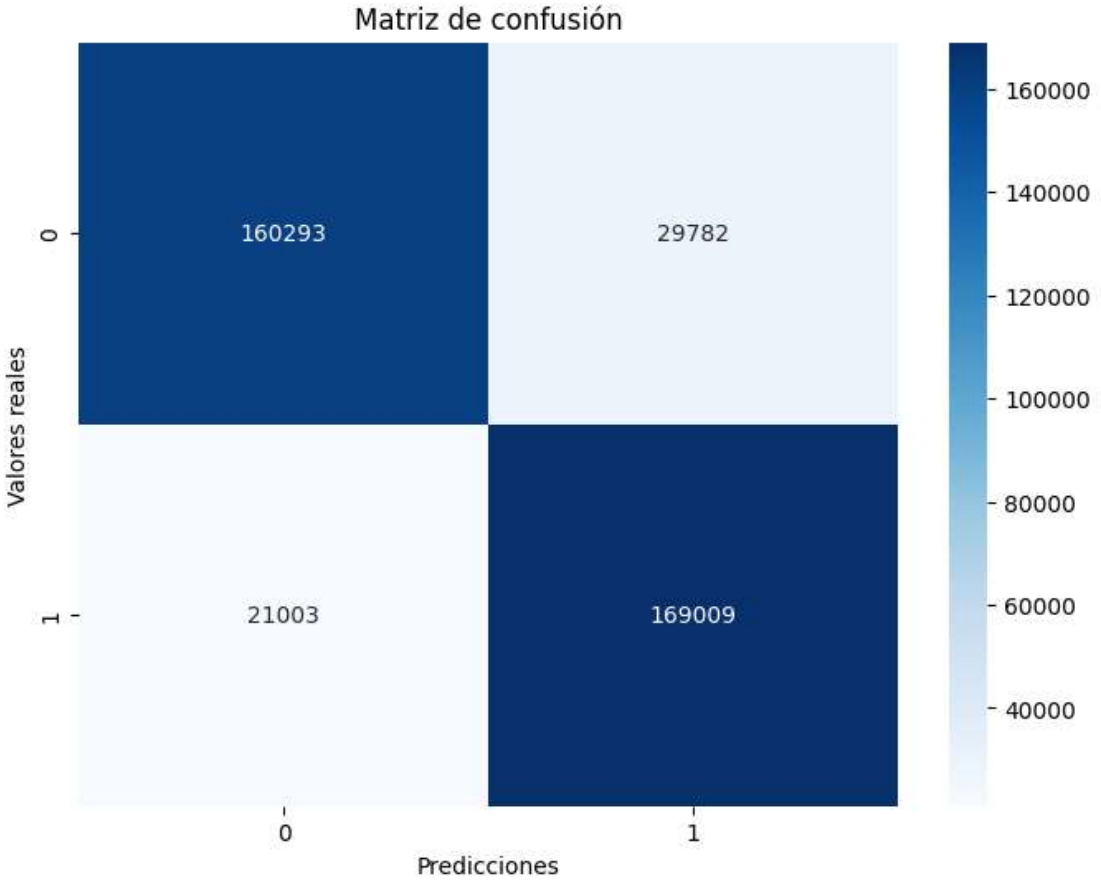
In [ ]:
```
#evaluamos el modelo
loss_smote, accuracy_smote = model_smote.evaluate(X_test_smote, y_
print('Loss:', loss_smote)
print('Accuracy:', accuracy_smote)

# Obtener las predicciones del modelo
y_pred_smote = model_smote.predict(X_test_smote)
y_pred_smote = (y_pred_smote > 0.5).astype(int)
```

```
precision_smote = precision_score(y_test_smote, y_pred_smote)
recall_smote = recall_score(y_test_smote, y_pred_smote)
f1_smote = f1_score(y_test_smote, y_pred_smote)

print('Precision:', precision_smote)
print('Recall:', recall_smote)
print('F1:', f1_smote)
```

```
11878/11878 [==============================] - 8s 630us/step - los
s: 0.3199 - accuracy: 0.8664
Loss: 0.3198527693748474
Accuracy: 0.8663858771324158
11878/11878 [==============================] - 6s 524us/step
Precision: 0.8501843644833016
Recall: 0.8894648759025746
F1: 0.8693811518944042
```

In [ ]:

```
# Calcular la matriz de confusión
cm = confusion_matrix(y_test_smote, y_pred_smote)

# Visualizar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicciones')
plt.ylabel('Valores reales')
plt.title('Matriz de confusión')
plt.show()
```



SUBMUESTREO

In [ ]:
```
# Dvidimos en clase mayoritaria y minoritaria
clase_mayoritaria = df_covid[df_covid['fallecidos'] == 0]
clase_minoritaria = df_covid[df_covid['fallecidos'] == 1]

# downsampling de la clase mayoritaria
clase_mayoritarial_downsampled = resample(clase_mayoritaria,
                                          replace = False,
                                          n_samples = len(clase_
                                          random_state = 42)

df_covid_downsampled = pd.concat([clase_mayoritarial_downsampled,

df_covid_downsampled['fallecidos'].value_counts()

# creamos el modelo de clasificacion
X = df_covid_downsampled.drop('fallecidos', axis=1)
y = df_covid_downsampled['fallecidos']


# dividimos el dataset en train y test
X_train, X_test_resample, y_train, y_test_resample = train_test_sp
```

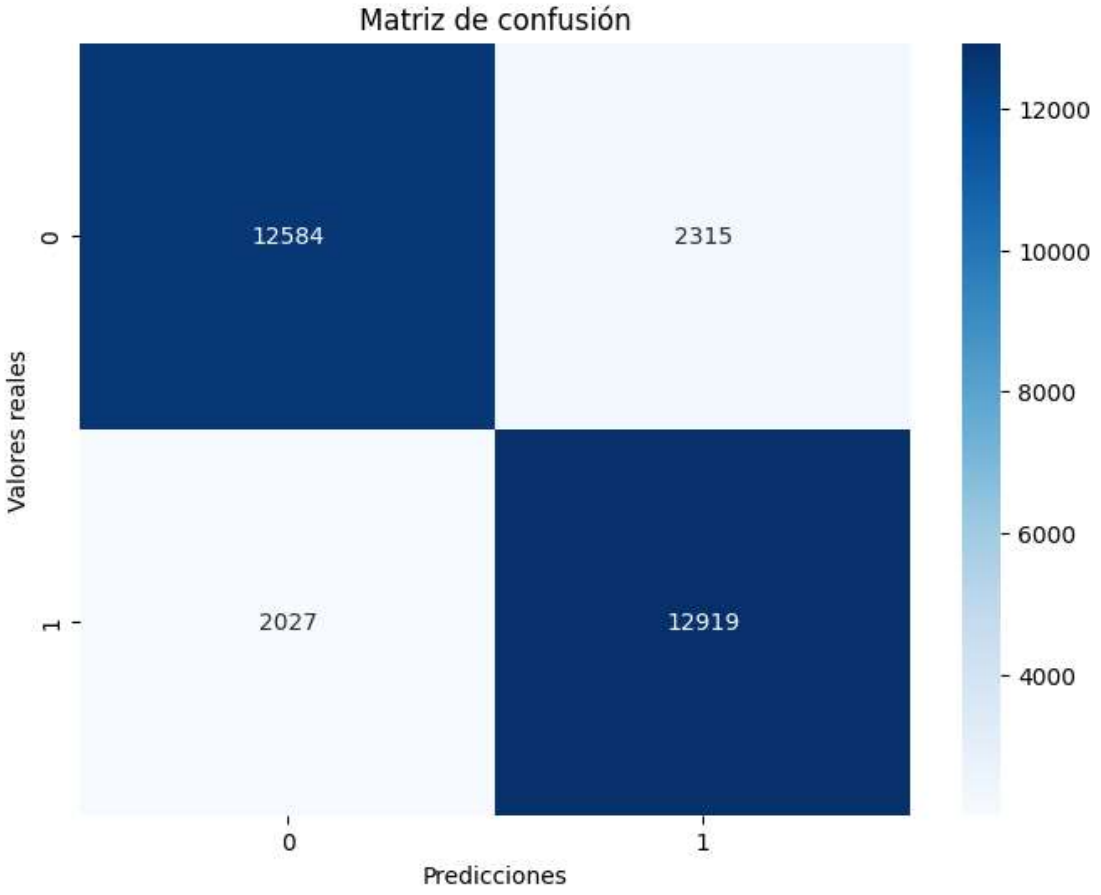
In [ ]:
```
# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test_resample = scaler.transform(X_test_resample)

# red neuronal entrenar
model_resample = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_tr
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_resample.compile(optimizer='adam', loss='binary_crossentropy

model_resample.fit(X_train, y_train, epochs=10, batch_size=32, val

#evaluamos el modelo
loss_resample, accuracy_resample = model_resample.evaluate(X_test_
print('Loss:', loss_resample)
print('Accuracy:', accuracy_resample)

# Obtener las predicciones del modelo
y_pred_resample = model_resample.predict(X_test_resample)
y_pred_resample = (y_pred_resample > 0.5).astype(int)
```

```
precision_resample = precision_score(y_test_resample, y_pred_resam
recall_resample = recall_score(y_test_resample, y_pred_resample)
f1_resample = f1_score(y_test_resample, y_pred_resample)

print('Precision:', precision_resample)
print('Recall:', recall_resample)
print('F1:', f1_resample)
```

```
Epoch 1/10
3731/3731 [==============================] - 4s 888us/step - loss:
0.3547 - accuracy: 0.8534 - val_loss: 0.3495 - val_accuracy: 0.855
0
Epoch 2/10
3731/3731 [==============================] - 3s 900us/step - loss:
0.3480 - accuracy: 0.8557 - val_loss: 0.3477 - val_accuracy: 0.854
5
Epoch 3/10
3731/3731 [==============================] - 3s 849us/step - loss:
0.3469 - accuracy: 0.8563 - val_loss: 0.3472 - val_accuracy: 0.855
4
Epoch 4/10
3731/3731 [==============================] - 3s 845us/step - loss:
0.3464 - accuracy: 0.8565 - val_loss: 0.3467 - val_accuracy: 0.855
3
Epoch 5/10
3731/3731 [==============================] - 3s 863us/step - loss:
0.3459 - accuracy: 0.8571 - val_loss: 0.3493 - val_accuracy: 0.855
0
Epoch 6/10
3731/3731 [==============================] - 3s 884us/step - loss:
0.3457 - accuracy: 0.8565 - val_loss: 0.3475 - val_accuracy: 0.855
6
Epoch 7/10
3731/3731 [==============================] - 3s 855us/step - loss:
0.3454 - accuracy: 0.8570 - val_loss: 0.3469 - val_accuracy: 0.854
7
Epoch 8/10
3731/3731 [==============================] - 3s 874us/step - loss:
0.3451 - accuracy: 0.8570 - val_loss: 0.3472 - val_accuracy: 0.854
4
Epoch 9/10
3731/3731 [==============================] - 3s 914us/step - loss:
0.3448 - accuracy: 0.8568 - val_loss: 0.3468 - val_accuracy: 0.854
8
Epoch 10/10
3731/3731 [==============================] - 3s 889us/step - loss:
0.3446 - accuracy: 0.8572 - val_loss: 0.3469 - val_accuracy: 0.854
5
933/933 [==============================] - 1s 639us/step - loss:
0.3469 - accuracy: 0.8545
Loss: 0.3468919098377228
Accuracy: 0.854515016078949
933/933 [==============================] - 1s 529us/step
Precision: 0.8480372850203493
Recall: 0.8643784290111066
F1: 0.8561298873426111
```

In [ ]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(y_test_resample, y_pred_resample)
```

```
# Visualizar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicciones')
plt.ylabel('Valores reales')
plt.title('Matriz de confusión')
plt.show()
```



Combinado smote-enn

In [ ]:
```
# # Dvidimos en clase mayoritaria y minoritaria
# clase_mayoritaria = df_covid[df_covid['fallecidos'] == 0]
# clase_minoritaria = df_covid[df_covid['fallecidos'] == 1]

# X = df_covid.drop('fallecidos', axis=1)
# y = df_covid['fallecidos']

# # Aplico smote-enn
# smote_enn = SMOTEENN(random_state=101)

# X_resampled, y_resampled = smote_enn.fit_resample(X, y)

# # creo el dataframe con los datos balanceados
# df_covid_balanceado = pd.concat([pd.DataFrame(X_resampled), pd.D

# # creamos el modelo de clasificacion
```

```
# X = df_covid_balanceado.drop('fallecidos', axis=1)
# y = df_covid_balanceado['fallecidos']


# # dividimos el dataset en train y test
# X_train, X_test_enn, y_train, y_test_enn = train_test_split(X, y
```

In [ ]:
```
df_covid_balanceado = pd.read_csv('df_covid_balanceado.csv')

# creamos el modelo de clasificacion
X = df_covid_balanceado.drop('fallecidos', axis=1)
y = df_covid_balanceado['fallecidos']


# dividimos el dataset en train y test
X_train, X_test_enn, y_train, y_test_enn = train_test_split(X, y,
```

In [ ]:
```
# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test_enn = scaler.transform(X_test_enn)
# red neuronal entrenar

model_enn = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_tr
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_enn.compile(optimizer='adam', loss='binary_crossentropy', me
model_enn.fit(X_train, y_train, epochs=10, batch_size=32, validati
```

```
Epoch 1/10
28144/28144 [==============================] - 25s 878us/step - lo
ss: 0.0905 - accuracy: 0.9686 - val_loss: 0.0748 - val_accuracy:
0.9741
Epoch 2/10
28144/28144 [==============================] - 25s 871us/step - lo
ss: 0.0719 - accuracy: 0.9741 - val_loss: 0.0708 - val_accuracy:
0.9751
Epoch 3/10
28144/28144 [==============================] - 25s 891us/step - lo
ss: 0.0667 - accuracy: 0.9762 - val_loss: 0.0653 - val_accuracy:
0.9785
Epoch 4/10
28144/28144 [==============================] - 25s 902us/step - lo
ss: 0.0633 - accuracy: 0.9775 - val_loss: 0.0648 - val_accuracy:
0.9766
Epoch 5/10
28144/28144 [==============================] - 25s 889us/step - lo
ss: 0.0609 - accuracy: 0.9786 - val_loss: 0.0635 - val_accuracy:
0.9782
Epoch 6/10
28144/28144 [==============================] - 25s 878us/step - lo
ss: 0.0591 - accuracy: 0.9792 - val_loss: 0.0587 - val_accuracy:
0.9802
Epoch 7/10
28144/28144 [==============================] - 24s 865us/step - lo
ss: 0.0575 - accuracy: 0.9797 - val_loss: 0.0564 - val_accuracy:
0.9809
Epoch 8/10
28144/28144 [==============================] - 25s 885us/step - lo
ss: 0.0562 - accuracy: 0.9803 - val_loss: 0.0559 - val_accuracy:
0.9801
Epoch 9/10
28144/28144 [==============================] - 25s 888us/step - lo
ss: 0.0551 - accuracy: 0.9805 - val_loss: 0.0545 - val_accuracy:
0.9812
Epoch 10/10
28144/28144 [==============================] - 25s 889us/step - lo
ss: 0.0543 - accuracy: 0.9809 - val_loss: 0.0560 - val_accuracy:
0.9806
```

Out[ ]: <keras.callbacks.History at 0x16c6660dea0>

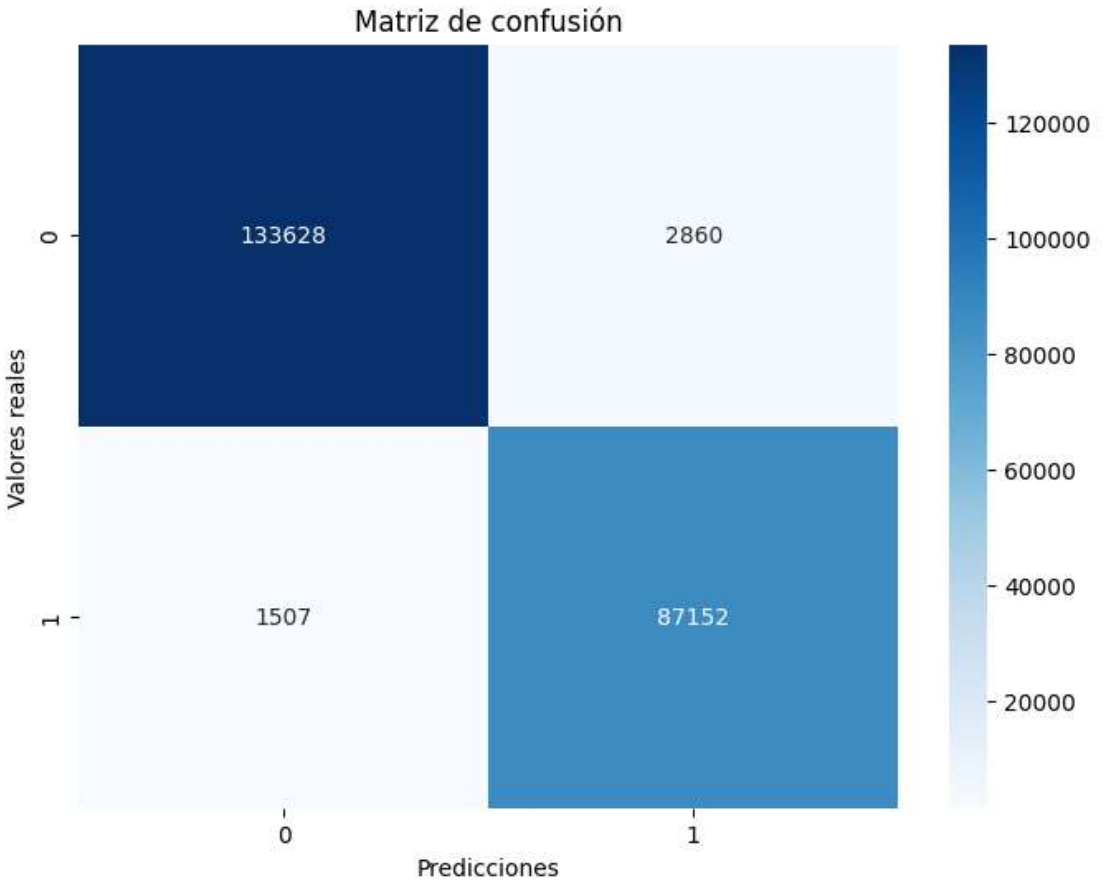
In [ ]:
```
#evaluamos el modelo
loss_enn, accuracy_enn = model_enn.evaluate(X_test_enn, y_test_enn
print('Loss:', loss_enn)
print('Accuracy:', accuracy_enn)

# Obtener las predicciones del modelo
y_pred_enn = model_enn.predict(X_test_enn)
y_pred_enn = (y_pred_enn > 0.5).astype(int)
```

```
precision_enn = precision_score(y_test_enn, y_pred_enn)
recall_enn = recall_score(y_test_enn, y_pred_enn)
f1_enn = f1_score(y_test_enn, y_pred_enn)

print('Precision:', precision_enn)
print('Recall:', recall_enn)
print('F1:', f1_enn)
```

```
7036/7036 [==============================] - 4s 624us/step - loss:
0.0560 - accuracy: 0.9806
Loss: 0.05603421479463577
Accuracy: 0.9806037545204163
7036/7036 [==============================] - 4s 529us/step
Precision: 0.9682264586943963
Recall: 0.9830022896716634
F1: 0.9755584286202014
```

In [ ]:

```
# Calcular la matriz de confusión
cm = confusion_matrix(y_test_enn, y_pred_enn)

# Visualizar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicciones')
plt.ylabel('Valores reales')
plt.title('Matriz de confusión')
plt.show()
```



COMBINADO OVER Y UNDER SAMPLING

In [ ]:
```
X = df_covid.drop('fallecidos', axis=1)
y = df_covid['fallecidos']

oversampler = RandomOverSampler(random_state=42)
X_over, y_over = oversampler.fit_resample(X, y)

undersampler = RandomUnderSampler(random_state=42)
X_balanced, y_balanced = undersampler.fit_resample(X_over, y_over)

# nuevo dataset balanceado

df_balanceado = pd.concat([X_balanced, y_balanced], axis=1)

df_balanceado.value_counts('fallecidos')
```

Out[ ]:
```
fallecidos
0    950217
1    950217
dtype: int64
```

In [ ]:
```
# creamos el modelo de clasificacion

X = df_balanceado.drop('fallecidos', axis=1)
y = df_balanceado['fallecidos']
```

In [ ]:
```
# dividimos el dataset en train y test

X_train, X_test_ou, y_train, y_test_ou = train_test_split(X, y, te
```

In [ ]:
```
# Normalizar los datos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test_ou = scaler.transform(X_test_ou)

# red neuronal entrenar
model_ou = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_tr
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_ou.compile(optimizer='adam', loss='binary_crossentropy', met

model_ou.fit(X_train, y_train, epochs=10, batch_size=32, validatio
```

```
Epoch 1/10
47511/47511 [==============================] - 42s 882us/step - lo
ss: 0.3472 - accuracy: 0.8562 - val_loss: 0.3447 - val_accuracy:
0.8571
Epoch 2/10
47511/47511 [==============================] - 41s 866us/step - lo
ss: 0.3455 - accuracy: 0.8570 - val_loss: 0.3436 - val_accuracy:
0.8578
Epoch 3/10
47511/47511 [==============================] - 42s 874us/step - lo
ss: 0.3451 - accuracy: 0.8572 - val_loss: 0.3439 - val_accuracy:
0.8575
Epoch 4/10
47511/47511 [==============================] - 42s 880us/step - lo
ss: 0.3447 - accuracy: 0.8573 - val_loss: 0.3433 - val_accuracy:
0.8577
Epoch 5/10
47511/47511 [==============================] - 42s 883us/step - lo
ss: 0.3445 - accuracy: 0.8575 - val_loss: 0.3439 - val_accuracy:
0.8580
Epoch 6/10
47511/47511 [==============================] - 41s 868us/step - lo
ss: 0.3443 - accuracy: 0.8575 - val_loss: 0.3432 - val_accuracy:
0.8581
Epoch 7/10
47511/47511 [==============================] - 42s 884us/step - lo
ss: 0.3441 - accuracy: 0.8576 - val_loss: 0.3429 - val_accuracy:
0.8581
Epoch 8/10
47511/47511 [==============================] - 42s 882us/step - lo
ss: 0.3440 - accuracy: 0.8576 - val_loss: 0.3430 - val_accuracy:
0.8585
Epoch 9/10
47511/47511 [==============================] - 42s 890us/step - lo
ss: 0.3438 - accuracy: 0.8578 - val_loss: 0.3427 - val_accuracy:
0.8584
Epoch 10/10
47511/47511 [==============================] - 42s 889us/step - lo
ss: 0.3438 - accuracy: 0.8579 - val_loss: 0.3428 - val_accuracy:
0.8582
```

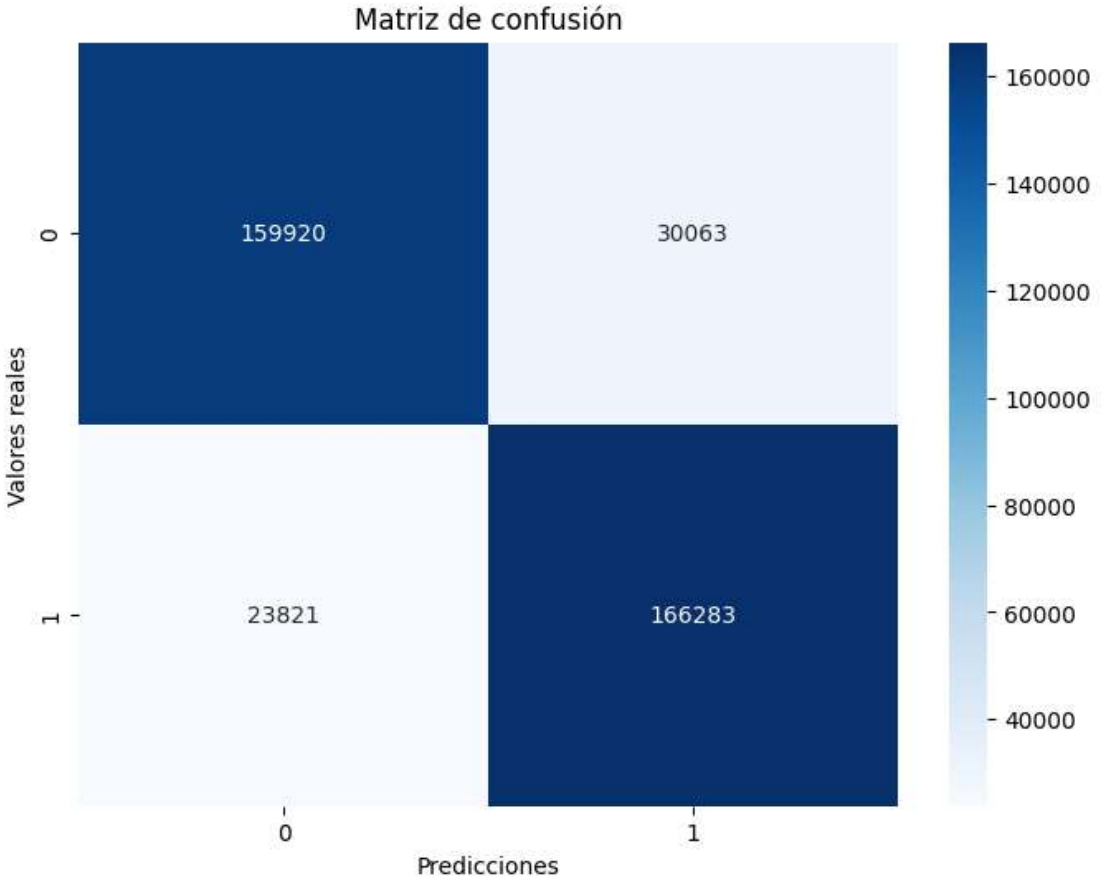
Out[ ]: <keras.callbacks.History at 0x16c66ef8fa0>

In [ ]:
```
#evaluamos el modelo
loss_ou, accuracy_ou = model_ou.evaluate(X_test_ou, y_test_ou)
print('Loss:', loss_ou)
print('Accuracy:', accuracy_ou)

# Obtener las predicciones del modelo
y_pred_ou = model_ou.predict(X_test_ou)
y_pred_ou = (y_pred_ou > 0.5).astype(int)
```

```
precision_ou = precision_score(y_test_ou, y_pred_ou)
recall_ou = recall_score(y_test_ou, y_pred_ou)
f1_ou = f1_score(y_test_ou, y_pred_ou)

print('Precision:', precision_ou)
print('Recall:', recall_ou)
print('F1:', f1_ou)
```

```
11878/11878 [==============================] - 7s 605us/step - los
s: 0.3428 - accuracy: 0.8582
Loss: 0.34279581904411316
Accuracy: 0.8582324385643005
11878/11878 [==============================] - 6s 518us/step
Precision: 0.8468876371303719
Recall: 0.8746949038421075
F1: 0.8605666968559968
```

In [ ]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(y_test_ou, y_pred_ou)

# Visualizar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicciones')
plt.ylabel('Valores reales')
plt.title('Matriz de confusión')
plt.show()
```



Grafioc de curvas de precision-recall

Imprimo todas las metricas

In [ ]:
```
# hago una tabla con los resultados de los modelos

resultados = pd.DataFrame({'Modelo': ['Modelo Base', 'Modelo Smote
                                      'Accuracy': [accuracy, accuracy_smote,
                                      'Precision': [precision, precision_smo
                                      'Recall': [recall, recall_smote, recal
                                      'F1': [f1, f1_smote, f1_resample, f1_e
                                      'Loss': [loss, loss_smote, loss_resamp
resultados
```

Out[ ]:

	Modelo	Accuracy	Precision	Recall	F1	Loss
0	Modelo Base	0.933267	0.563146	0.317898	0.406388	0.156364
1	Modelo Smote	0.866386	0.850184	0.889465	0.869381	0.319853
2	Modelo Resample	0.854515	0.848037	0.864378	0.856130	0.346892
3	Modelo ENN	0.980604	0.968226	0.983002	0.975558	0.056034
4	Modelo Over Under	0.858232	0.846888	0.874695	0.860567	0.342796

In [ ]:
```
# mapa de calor de los resultados

plt.figure(figsize=(10, 6))
sns.heatmap(resultados.set_index('Modelo'), annot=True, cmap='Blue
plt.title('Comparación de los modelos')

plt.show()
```

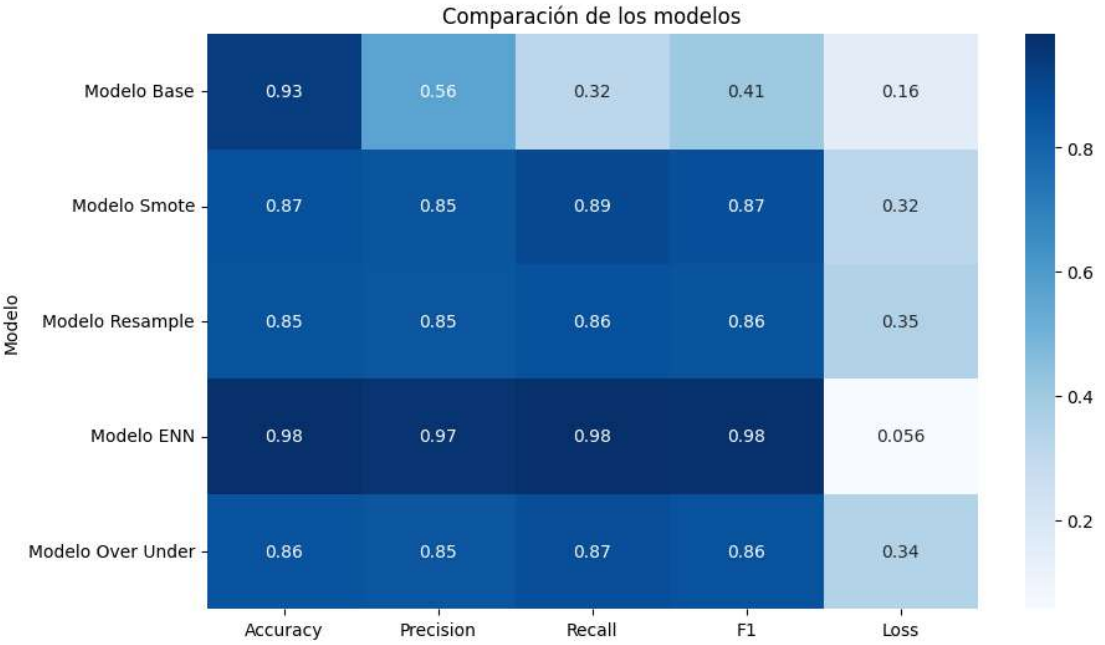
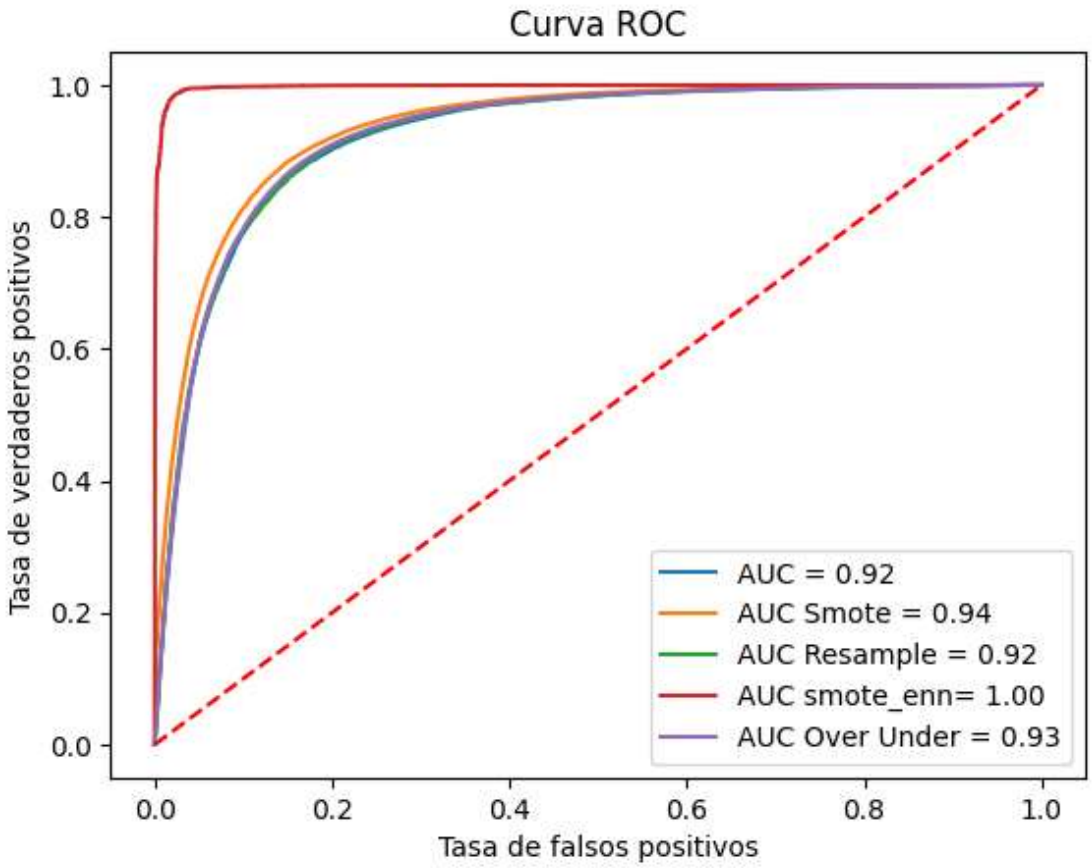


Grafico de curvas ROC

In [ ]:
```
# comparacion de los modelos

# grafico todas las curvas ROC en un solo grafico
y_pred = model.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

y_pred_smote = model_smote.predict(X_test_smote)
fpr_smote, tpr_smote, thresholds_smote = roc_curve(y_test_smote, y
auc_smote = roc_auc_score(y_test_smote, y_pred_smote)

y_pred_resample = model_resample.predict(X_test_resample)
fpr_resample, tpr_resample, thresholds_resample = roc_curve(y_test
auc_resample = roc_auc_score(y_test_resample, y_pred_resample)

y_pred_enn = model_enn.predict(X_test_enn)
fpr_enn, tpr_enn, thresholds_enn = roc_curve(y_test_enn, y_pred_en
auc_enn = roc_auc_score(y_test_enn, y_pred_enn)

y_pred_ou = model_ou.predict(X_test_ou)
fpr_ou, tpr_ou, thresholds_ou = roc_curve(y_test_ou, y_pred_ou)
auc_ou = roc_auc_score(y_test_ou, y_pred_ou)


plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
plt.plot(fpr_smote, tpr_smote, label=f'AUC Smote = {auc_smote:.2f}
plt.plot(fpr_resample, tpr_resample, label=f'AUC Resample = {auc_r
plt.plot(fpr_enn, tpr_enn, label=f'AUC smote_enn= {auc_enn:.2f}')
plt.plot(fpr_ou, tpr_ou, label=f'AUC Over Under = {auc_ou:.2f}')
```

```
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('Tasa de falsos positivos')
plt.ylabel('Tasa de verdaderos positivos')
plt.title('Curva ROC')

plt.legend()

plt.show()
```

```
6406/6406 [==============================] - 4s 609us/step
11878/11878 [==============================] - 6s 536us/step
933/933 [==============================] - 1s 521us/step
7036/7036 [==============================] - 4s 548us/step
11878/11878 [==============================] - 6s 544us/step
```



In [ ]:
```
# saco los recall y precision para graficar
y_pred = model.predict(X_test)
precision_original, recall_original, tresholds_original = precisio

y_pred_smote = model_smote.predict(X_test_smote)
precision_smote, recall_smote, tresholds_smote = precision_recall_

y_pred_resample = model_resample.predict(X_test_resample)
precision_resample, recall_resample, tresholds_resample = precisio

y_pred_enn = model_enn.predict(X_test_enn)
precision_enn, recall_enn, tresholds_enn = precision_recall_curve(

y_pred_ou = model_ou.predict(X_test_ou)
```

```
precision_ou, recall_ou, tresholds_ou = precision_recall_curve(y_t



# grafico todas las curvas precision recall en un solo grafico
plt.plot(recall, precision, label=f'Precision Recall = {auc:.2f}')
plt.plot(recall_smote, precision_smote, label=f'Precision Recall S
plt.plot(recall_resample, precision_resample, label=f'Precision Re
plt.plot(recall_enn, precision_enn, label=f'Precision Recall ENN =
plt.plot(recall_ou, precision_ou, label=f'Precision Recall Over Un

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Curva Precision Recall')

plt.legend()
```

```
6406/6406 [==============================] - 4s 584us/step
11878/11878 [==============================] - 6s 543us/step
933/933 [==============================] - 1s 544us/step
7036/7036 [==============================] - 4s 529us/step
11878/11878 [==============================] - 6s 509us/step
```

Out[ ]: <matplotlib.legend.Legend at 0x16c4d0df7f0>