Estimate chance of Bankruptcy from Qualitative parameters from experts

DISEÑO DEL PROYECTO

This *beginner* level data set has 250 rows and 7 columns.

This dataset contains 6 qualitative parameters from experts which could be used to predict the bankruptcy.

OBJETIVO

Level: Beginner
Recommended Use: Classification Models
Domain: Finance/Banking

Predict the bakruptcy.

This data set is recommended for learning and practicing your skills in **exploratory data analysis**, **data visualization**, and **classification modelling techniques**. Feel free to explore the data set with multiple **supervised** and **unsupervised** learning techniques.

ENTIDADES Y DATOS

The Following data dictionary gives more details on this data set:

Data Dictionary

Column Position	Atrribute Name	Definition	Data Type	Example	% Null Ratios
1	Industrial Risk	Industrial Risk (P: Positive, A: Average, N: Negative)	Qualitative	P, A, N	0
2	Management Risk	Management Risk (P: Positive, A: Average, N: Negative)	Qualitative	P, A, N	0
3	Financial Flexibility	Financial Flexibility (P: Positive, A: Average, N: Negative)	Qualitative	P, A, N	0
4	Credibility	Credibility (P: Positive, A: Average, N: Negative)	Qualitative	P, A, N	0
5	Competitiveness	Competitiveness (P: Positive, A: Average, N: Negative)	Qualitative	P, A, N	0
6	Operating Risk	Operating Risk (P: Positive, A: Average, N: Negative)	Qualitative	P, A, N	0

Column Position	Atrribute Name	Definition	Data Type	Example	% Null Ratios
7	Class	Class (B: Bankruptcy, NB: Non-Bankruptcy)	Qualitative	B, NB	0

Following are the details of the qualitative risk factors which mentions their risk components:

i.Industry risk (IR):

```
- Government policies and International agreements, <br>
- Cyclicality, <br>
- Degree of competition, <br>
- The price and stability of market supply, <br>
- The size and growth of market demand, <br>
- The sensitivity to changes in macroeconomic factors, <br>
- Domestic and international competitive power, <br>
- Product Life Cycle.<br>
```

ii.Management risk(MR):

```
- Ability and competence of management, <br>
- Stability of management,<br>
- The relationship between management/ owner, <br>
- Human resources management, <br>
- Growth process/business performance, <br>
- Short and long term business planning, <br>
- achievement and feasibility. <br>
```

iii.Financial Flexibility(FF):

```
- Direct financing, <br>
- Indirect financing, <br>
- Other financing <br>
```

iv.Credibility (CR):

```
- Credit history,  <br>
- reliability of information, <br>
- The relationship with financial institutes.<br>
```

v.Competitiveness (CO):

```
- Market position, <br>
- The level of core capacities, <br>
- Differentiated strategy, <br>
```

vi.Operating Risk (OP):

- The stability and diversity of procurement,
- The stability of transaction,
- The efficiency of production,
- The prospects for demand for product and service,
- Sales diversification,
- Sales price and settlement condition,
- Collection of A/R,
- Effectiveness of sale network.

1 - Set Up

IMPORTACIÓN DE PAQUETES

In [1]:
```
import pandas as pd
import numpy as np
```

CREAR LOS DATASETS INICIALES

CARGAR LOS DATOS

In [36]:
```
df = pd.read_csv('../../02_Datos/01_Originales/Qualitative_Bankruptcy.txt', names=
df
```

Out[36]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
0	P	P	A	A	A	
1	N	N	A	A	A	
2	A	A	A	A	A	
3	P	P	P	P	P	
4	N	N	P	P	P	
...	...	...	...	...	...	
245	N	N	A	N	N	
246	P	N	N	N	N	
247	A	N	N	N	N	
248	N	N	N	N	N	
249	P	N	N	N	A	

250 rows × 7 columns

EXTRAER Y RESERVAR EL DATASET DE PRUEBA

In [37]:
```
df_prueba = df.sample(frac=0.25)
```

In [38]:
```
df_prueba.shape
```

Out[38]:
```
(62, 7)
```

EXTRAER Y RESERVAR EL DATASET DE TRABAJO

In [39]:
```
df_trabajo = df[~df.index.isin(df_prueba.index)]
```

In [40]:
```
print('Dataset de prueba:', df_prueba.shape)
print('Dataset de trabajo:', df_trabajo.shape)
```

```
Dataset de prueba: (62, 7)
Dataset de trabajo: (188, 7)
```

GUARDAR LOS DATASETS DE TRABAJO Y PRUEBA

In [41]:
```
df_prueba.to_csv('../../02_Datos/02_Validacion/prueba.csv')
df_trabajo.to_csv('../../02_Datos/03_Trabajo/trabajo.csv')
```

2 - CALIDAD DE DATOS

IMPORTACIÓN DE PAQUETES

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

CARGAR LOS DATOS

In [3]:
```
df = pd.read_csv('../../02_Datos/03_Trabajo/trabajo.csv', index_col=0)
```

CORREGIR EL NOMBRE DE LAS VARIABLES

El nombre de las variables ha sido modificado manualmente en el notebook de Set up.

VISIÓN GENERAL

In [4]:
```
df.head()
```

Out[4]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_risk
1	N	N	A	A	A	N
2	A	A	A	A	A	A
3	P	P	P	P	P	P
4	N	N	P	P	P	N
5	A	A	P	P	P	A

TIPOS DE DATOS

In [5]:
```
df.dtypes
```

Out[5]:
```
industrial_risk          object
management_risk          object
financial_flexibility    object
credibility              object
competitiveness          object
operating_risk           object
target                   object
dtype: object
```

BALANCEO DE LA TARGET

In [6]:
```
df.target.value_counts(normalize=True)
```

Out[6]:
```
NB    0.547872
B     0.452128
Name: target, dtype: float64
```

The data is balanced

VALORES ÚNICOS

In [7]:
```
df.nunique()
```

Out[7]:
```
industrial_risk          3
management_risk          3
financial_flexibility    3
credibility              3
competitiveness          3
operating_risk           3
target                   2
dtype: int64
```

There aren't unique values

VALORES DUPLICADOS

In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
97
```

The data has duplicate value. We remove the duplicates:

In [9]:
```
df = df.drop_duplicates()
df
```

Out[9]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
1	N	N	A	A	A	
2	A	A	A	A	A	
3	P	P	P	P	P	
4	N	N	P	P	P	
5	A	A	P	P	P	
...	...	...	...	...	...	
178	A	P	N	A	N	
179	N	N	N	P	N	
182	N	N	N	P	N	
215	N	A	P	A	N	
234	N	A	N	N	N	

91 rows × 7 columns

We check the data balance again:

In [10]:
```
df.target.value_counts(normalize= True)
```

Out[10]:
```
NB    0.725275
B     0.274725
Name: target, dtype: float64
```

VALORES NULOS

In [11]:
```
df.isna().sum()
```

Out[11]:
```
industrial_risk          0
management_risk          0
financial_flexibility    0
credibility              0
competitiveness          0
operating_risk           0
target                   0
dtype: int64
```

The dataset does not have null values.

GESTIÓN DE CATEGÓRICAS

Estadísticos y gráficos de las variables.

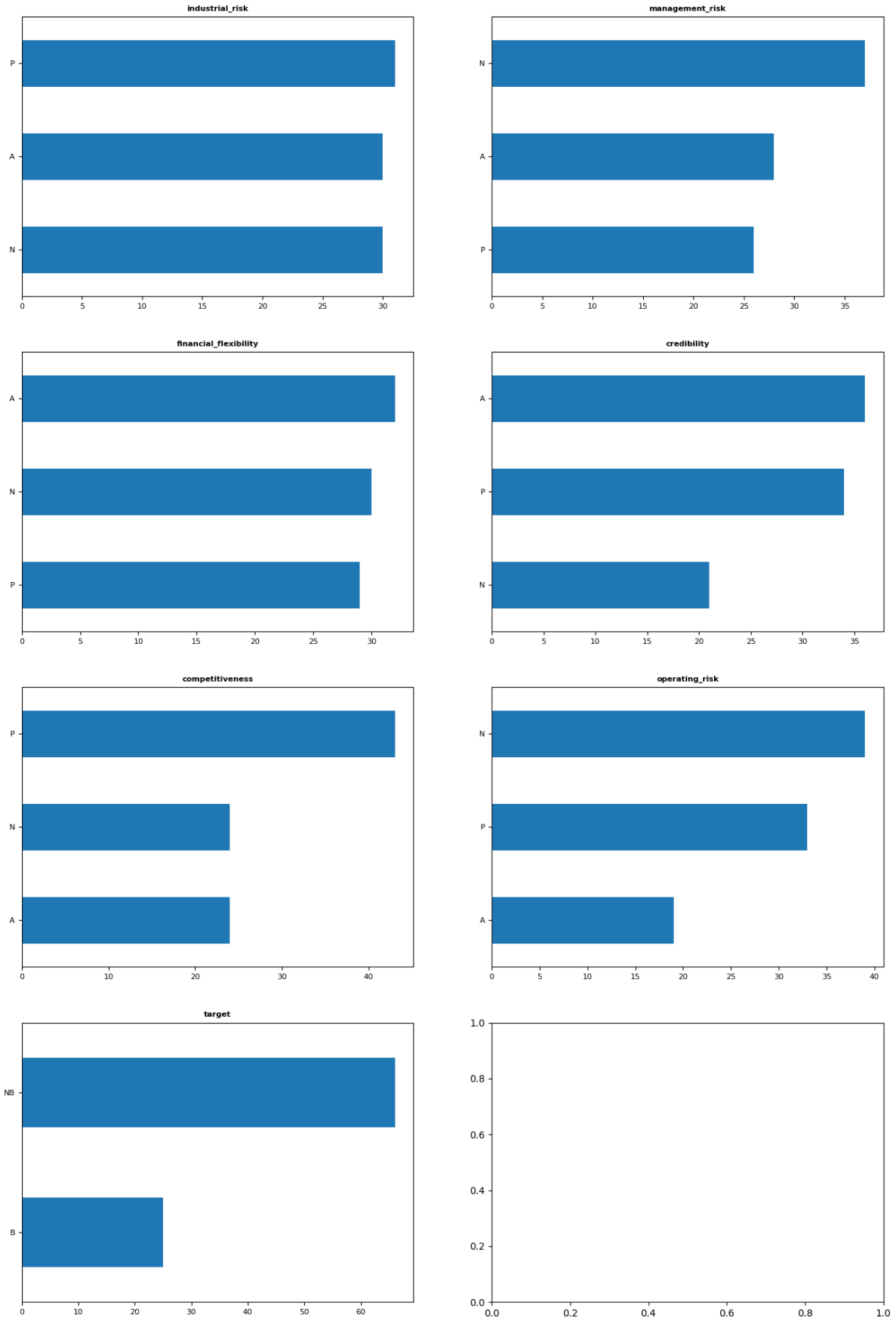
In [12]:
```
def estadisticos_cont(df_cat):
    #Calculamos describe
    estadisticos = df_cat.describe().T
    return(estadisticos)

estadisticos_cont(df)
```

Out[12]:

	count	unique	top	freq
industrial_risk	91	3	P	31
management_risk	91	3	N	37
financial_flexibility	91	3	A	32
credibility	91	3	A	36
competitiveness	91	3	P	43
operating_risk	91	3	N	39
target	91	2	NB	66

In [13]:
```
def graficos_eda_categoricos(df_cat):

    #Calculamos el número de filas que necesitamos
    from math import ceil
    filas = ceil(df_cat.shape[1] / 2)

    #Definimos el gráfico
    f, ax = plt.subplots(nrows = filas, ncols = 2, figsize = (16, filas * 6))

    #Aplanamos para iterar por el gráfico como si fuera de 1 dimensión en lugar de
```

```
    ax = ax.flat

    #Creamos el bucle que va añadiendo gráficos
    for cada, variable in enumerate(df_cat):
        df_cat[variable].value_counts(ascending = True).plot.barh(ax = ax[cada])
        ax[cada].set_title(variable, fontsize = 8, fontweight = "bold")
        ax[cada].tick_params(labelsize = 8)

graficos_eda_categoricos(df)
```



GESTIÓN DE ATÍPICOS

No se detecta ninguna categórica con poco frecuencia (salvo la target).

GUARDAR DATASET TRAS CALIDAD DE DATOS

In [14]:
```
df.to_csv('../../02_Datos/03_Trabajo/tablon_analitico.csv')
```

CONCLUSIÓN:

- Hemos reducido el dataset quitando registros duplicados.
- No se detectan valores atípicos en las variables categóricas.

3 - TRANSFORMACIÓN DE DATOS

IMPORTACIÓN DE PAQUETES

In [56]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import OrdinalEncoder
```

IMPORTAR LOS DATOS

In [57]:
```
df = pd.read_csv('../../02_Datos/03_Trabajo/tablon_analitico.csv', index_col= 0)
df.head()
```

Out[57]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_risk
1	N	N	A	A	A	N
2	A	A	A	A	A	A
3	P	P	P	P	P	P
4	N	N	P	P	P	N
5	A	A	P	P	P	A

In [58]:
```
x = df.drop(columns= 'target').copy()
y = df.target.reset_index().copy()
```

In [59]:
```
x.industrial_risk.value_counts()
```

Out[59]:
```
P    31
N    30
A    30
Name: industrial_risk, dtype: int64
```

In [60]:
```
y.drop(columns= 'index', inplace= True)
```

TRANSFORMACIÓN DE CATEGÓRICAS

Variables a aplicar Ordinal Encoder

In [61]:
```
var_oe = ['industrial_risk',
          'management_risk',
          'financial_flexibility',
          'credibility',
          'competitiveness',
          'operating_risk']
```

Orden y categoría de las variables

In [62]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [63]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [64]:
```
df_oe = oe.fit_transform(x[var_oe])
```

In [65]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
df_oe = pd.DataFrame(df_oe, columns= nombre_oe)
df_oe
```

Out[65]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_o
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [66]:
```
y = y.replace({'B':1 , 'NB': 0})
y
```

Out[66]:

	target
0	0
1	0
2	0
3	0
4	0
...	...
86	1
87	1
88	1
89	1
90	1

91 rows × 1 columns

AGRUPAR PREDICTORAS Y TARGET EN UN DATASET

In [69]:
```
df_tablon = pd.concat([df_oe, y], axis= 1, ignore_index= False)
```

GUARDAR DATASET TRAS TRANSFORMACIÓN DE DATOS

In [70]:
```
df_tablon.to_pickle('../../02_Datos/03_Trabajo/df_tablon_oe.pickle')
```

4 - PRESELECCIÓN DEL MODELO LAZY PREDICT

IMPORTACIÓN DE PAQUETES

In [1]:
```
import numpy as np
import pandas as pd

from lazypredict.Supervised import LazyClassifier
from sklearn.model_selection import train_test_split
```

CARGAR LOS DATOS

In [2]:
```
df = pd.read_pickle('../../02_Datos/03_Trabajo/df_tablon_oe.pickle')
df.head()
```

Out[2]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_oe
0	0.00	0.00	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	1.00
2	2.00	2.00	2.00	2.00	2.00
3	0.00	0.00	2.00	2.00	2.00
4	1.00	1.00	2.00	2.00	2.00

SEPARAR PREDICTORAS Y TARGET

In [4]:
```
x = df.drop(columns= 'target').copy()
y = df.target.copy()
```

In [5]:
```
x.head()
```

Out[5]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_oe
0	0.00	0.00	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	1.00
2	2.00	2.00	2.00	2.00	2.00
3	0.00	0.00	2.00	2.00	2.00
4	1.00	1.00	2.00	2.00	2.00

In [7]:
```
y.value_counts()
```

Out[7]:
```
0    66
1    25
Name: target, dtype: int64
```

SEPARAR DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [8]:
```
train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_stat
```

MODELIZAR SEGÚN LAZYPREDICT

In [9]:
```
reg = LazyClassifier(random_state= 42, verbose=0)
models, predictions = reg.fit(train_x, val_x, train_y, val_y)
```

```
100%|██████████| 29/29 [00:01<00:00, 24.21it/s]
```

```
[LightGBM] [Info] Number of positive: 18, number of negative: 45
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testi
ng was 0.001280 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 23
[LightGBM] [Info] Number of data points in the train set: 63, number of used featu
res: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.285714 -> initscore=-0.916291
[LightGBM] [Info] Start training from score -0.916291
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

In [10]:

```
models
```

Out[10]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LGBMClassifier	1.00	1.00	1.00	1.00	0.29
LabelPropagation	1.00	1.00	1.00	1.00	0.02
RidgeClassifier	1.00	1.00	1.00	1.00	0.03
NuSVC	1.00	1.00	1.00	1.00	0.01
RidgeClassifierCV	1.00	1.00	1.00	1.00	0.02
LogisticRegression	1.00	1.00	1.00	1.00	0.02
BaggingClassifier	1.00	1.00	1.00	1.00	0.03
LinearDiscriminantAnalysis	1.00	1.00	1.00	1.00	0.07
LabelSpreading	1.00	1.00	1.00	1.00	0.01
KNeighborsClassifier	1.00	1.00	1.00	1.00	0.03
QuadraticDiscriminantAnalysis	1.00	1.00	1.00	1.00	0.01
GaussianNB	1.00	1.00	1.00	1.00	0.01
ExtraTreesClassifier	1.00	1.00	1.00	1.00	0.09
ExtraTreeClassifier	1.00	1.00	1.00	1.00	0.01
SVC	1.00	1.00	1.00	1.00	0.01
DecisionTreeClassifier	1.00	1.00	1.00	1.00	0.01
CalibratedClassifierCV	1.00	1.00	1.00	1.00	0.06
XGBClassifier	1.00	1.00	1.00	1.00	0.06
RandomForestClassifier	1.00	1.00	1.00	1.00	0.12
AdaBoostClassifier	0.96	0.98	0.98	0.97	0.10
NearestCentroid	0.96	0.98	0.98	0.97	0.01
LinearSVC	0.96	0.98	0.98	0.97	0.01
Perceptron	0.93	0.95	0.95	0.93	0.01
PassiveAggressiveClassifier	0.93	0.95	0.95	0.93	0.01
SGDClassifier	0.93	0.95	0.95	0.93	0.01
BernoulliNB	0.82	0.88	0.88	0.83	0.12
DummyClassifier	0.75	0.50	0.50	0.64	0.01

CONCLUSIÓN

Probaremos los siguientes modelos propuestos por LazyPredict:


Alt text

PRÓXIMOS PASOS:

1. Realizar predicción con modelos bases (sin hiperparámetros) y incluiremos el XGBoost.

2. Preselección de variables según modelo base.
3. Realizar predicción con modelo + hiperparámetros + preselección de variables.
4. Evaluar métricas

5 THE BEST ESTIMATOR - MODELO BASE

MODELAR ALGORITMO DE CLASIFICACIÓN - MODELO BASE

Los modelos que van a competir mediante el GridSearch:

IMPORTACIÓN DE PAQUETES

In [20]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# Crear train and test
from sklearn.model_selection import train_test_split

#Modelos seleccionados por LazyPredict
from sklearn.semi_supervised import LabelPropagation
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import RidgeClassifierCV
from sklearn.linear_model import LogisticRegression
from lightgbm import LGBMClassifier
from sklearn.svm import NuSVC

#Optimizar modelos
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from sklearn.metrics import classification_report, confusion_matrix
import scikitplot as skplt

#Crear Pipeline

from sklearn.pipeline import Pipeline

import cloudpickle

import warnings
warnings.filterwarnings("ignore")
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

In [2]:
```
df = pd.read_pickle('../../02_Datos/03_Trabajo/df_tablon_oe.pickle')
df.head()
```

Out[2]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_oe
0	0.0	0.0	1.0	1.0	1.(
1	1.0	1.0	1.0	1.0	1.(
2	2.0	2.0	2.0	2.0	2.(
3	0.0	0.0	2.0	2.0	2.(
4	1.0	1.0	2.0	2.0	2.(

SEPARAR PREDICTORAS Y TARGET

In [3]:
```
x = df.drop(columns= 'target').copy()
y = df.target.copy()
```

In [4]:
```
x.head()
```

Out[4]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_oe
0	0.0	0.0	1.0	1.0	1.(
1	1.0	1.0	1.0	1.0	1.(
2	2.0	2.0	2.0	2.0	2.(
3	0.0	0.0	2.0	2.0	2.(
4	1.0	1.0	2.0	2.0	2.(

In [5]:
```
y.head()
```

Out[5]:
```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [6]:
```
train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_stat
```

CREAR PIPE Y EL DICCIONARIO CON LOS ALGORITMOS, PARÁMETROS Y VALORES

In [7]:
```
pipe = Pipeline([('algoritmo', LogisticRegression())])

grid = [

            {
                'algoritmo' : [LabelPropagation()]
```

```
        },

        {
            'algoritmo' : [RidgeClassifier()]
        },

        {
            'algoritmo' : [RidgeClassifierCV()]
        },

        {
            'algoritmo' : [LogisticRegression()]
        },

        {
            'algoritmo' : [LGBMClassifier()]
        },

        {
            'algoritmo' : [NuSVC()]
        }

    ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

In [8]:
```
grid_search = GridSearchCV(estimator= pipe,
                           param_grid= grid,
                           cv = 5,
                           scoring= 'roc_auc',
                           n_jobs= -1,
                           verbose=0)
```

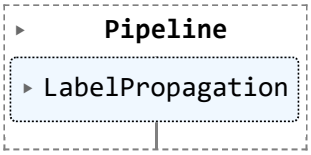
In [9]:
```
modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

Out[9]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	pa
0	0.017664	0.015424	0.006618	0.001981	LabelPropagation()	{'algori LabelPropagati
1	0.014150	0.006578	0.004234	0.000971	RidgeClassifier()	{'algori RidgeClassif
2	0.010732	0.006524	0.003570	0.000472	RidgeClassifierCV()	{'algori RidgeClassifier
3	0.009355	0.002684	0.003703	0.000848	LogisticRegression()	{'algori LogisticRegressi
5	0.004065	0.001463	0.004509	0.001267	NuSVC()	{'algori NuS
4	0.187360	0.029756	0.004786	0.000399	LGBMClassifier()	{'algori LGBMClassif

In [10]:
```
modelo.best_estimator_
```

Out[10]:



In [11]:
```
modelo.best_params_
```

Out[11]:
```
{'algoritmo': LabelPropagation()}
```

In [12]:
```
modelo.best_score_
```

Out[12]:
```
1.0
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [13]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [14]:
```
m_best_estimator = str(modelo.best_estimator_[0])
m_best_estimator = m_best_estimator.split('(')[0]
m_best_params = str(modelo.best_params_)
m_best_score = str(modelo.best_score_)
```

EVALUAR

PRECEDIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el train

In [15]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [16]:
```
t_roc_auc = roc_auc_score(train_y, pred)
t_accuracy = accuracy_score(train_y, pred)
t_report = classification_report(train_y, pred)

print(f"Roc AUC: {t_roc_auc}")
print(f"Accuracy: {t_accuracy}")
print(f"Classification Report:\n{t_report}")
```

```
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        45
           1       1.00      1.00      1.00        18

    accuracy                           1.00        63
   macro avg       1.00      1.00      1.00        63
weighted avg       1.00      1.00      1.00        63
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

In [17]:
```
pred = modelo.best_estimator_.predict(val_x)
pred_proba = modelo.best_estimator_.predict_proba(val_x)[:,1]
```

Evaluar sobre la validación

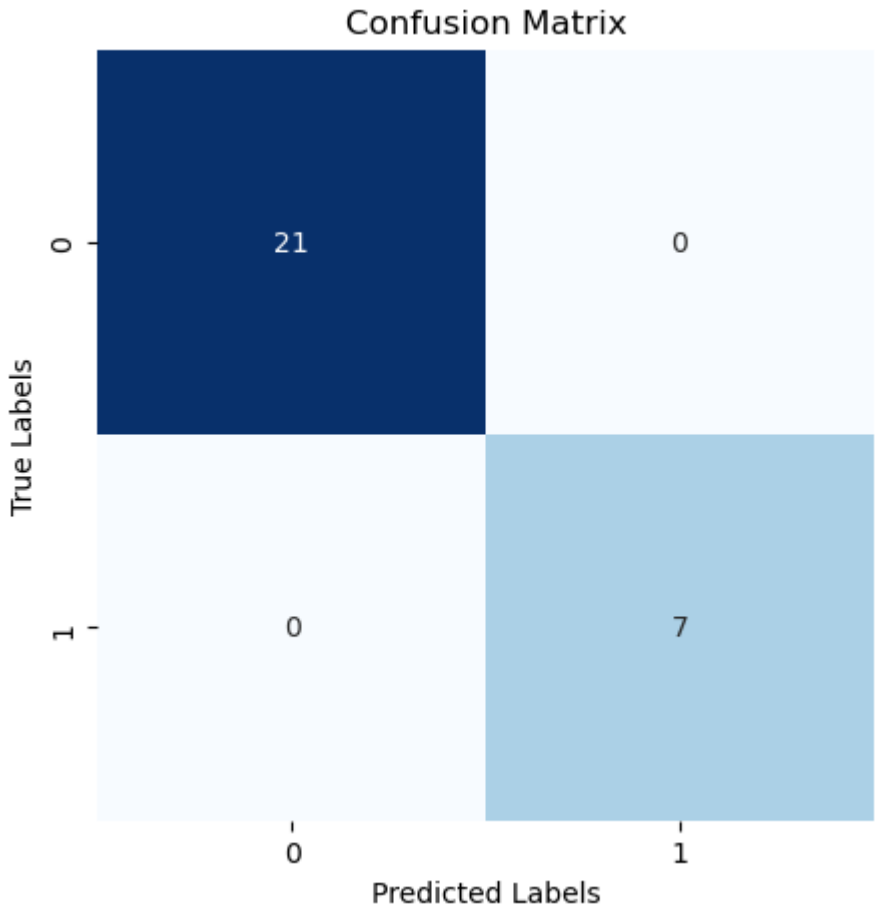
In [18]:
```
v_roc_auc_proba = roc_auc_score(val_y, pred_proba)
v_roc_auc = roc_auc_score(val_y, pred)
v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

```
Roc AUC_proba: 1.0
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00         7

    accuracy                           1.00        28
   macro avg       1.00      1.00      1.00        28
weighted avg       1.00      1.00      1.00        28
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass

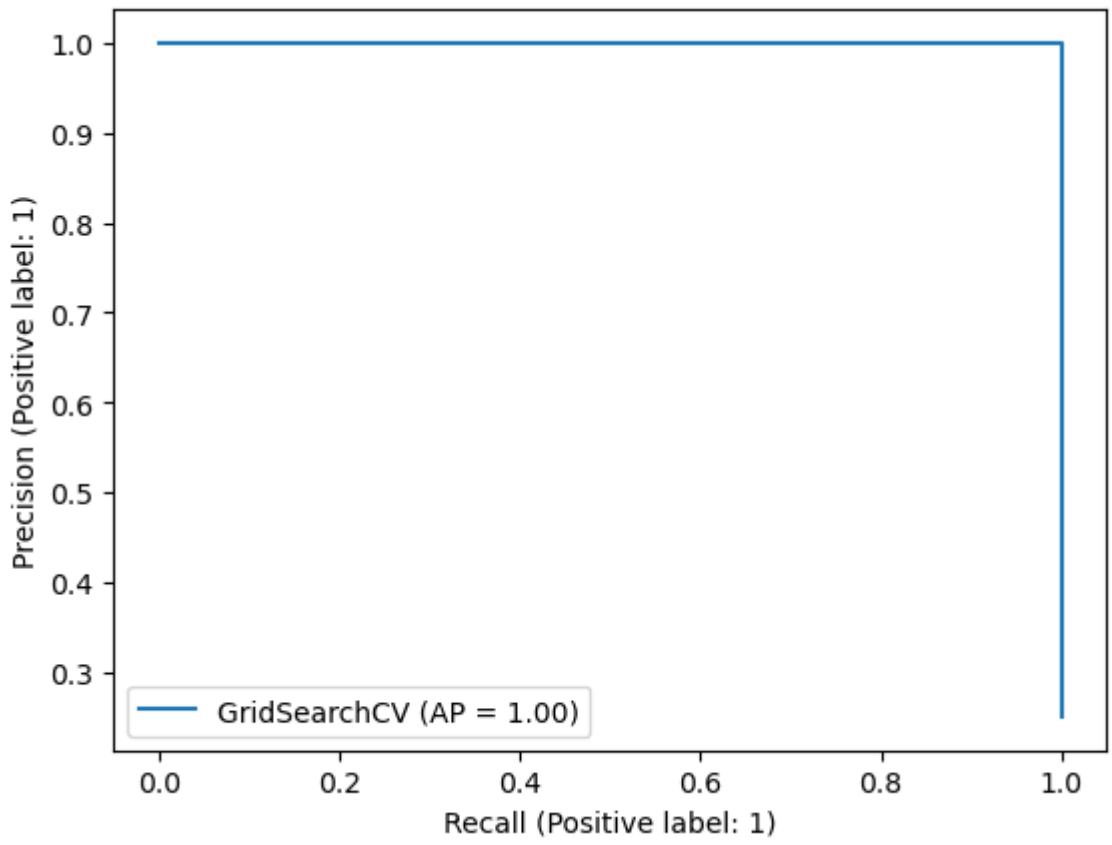
In [21]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)

# Configurar etiquetas y título del gráfico
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



Precision-Recall

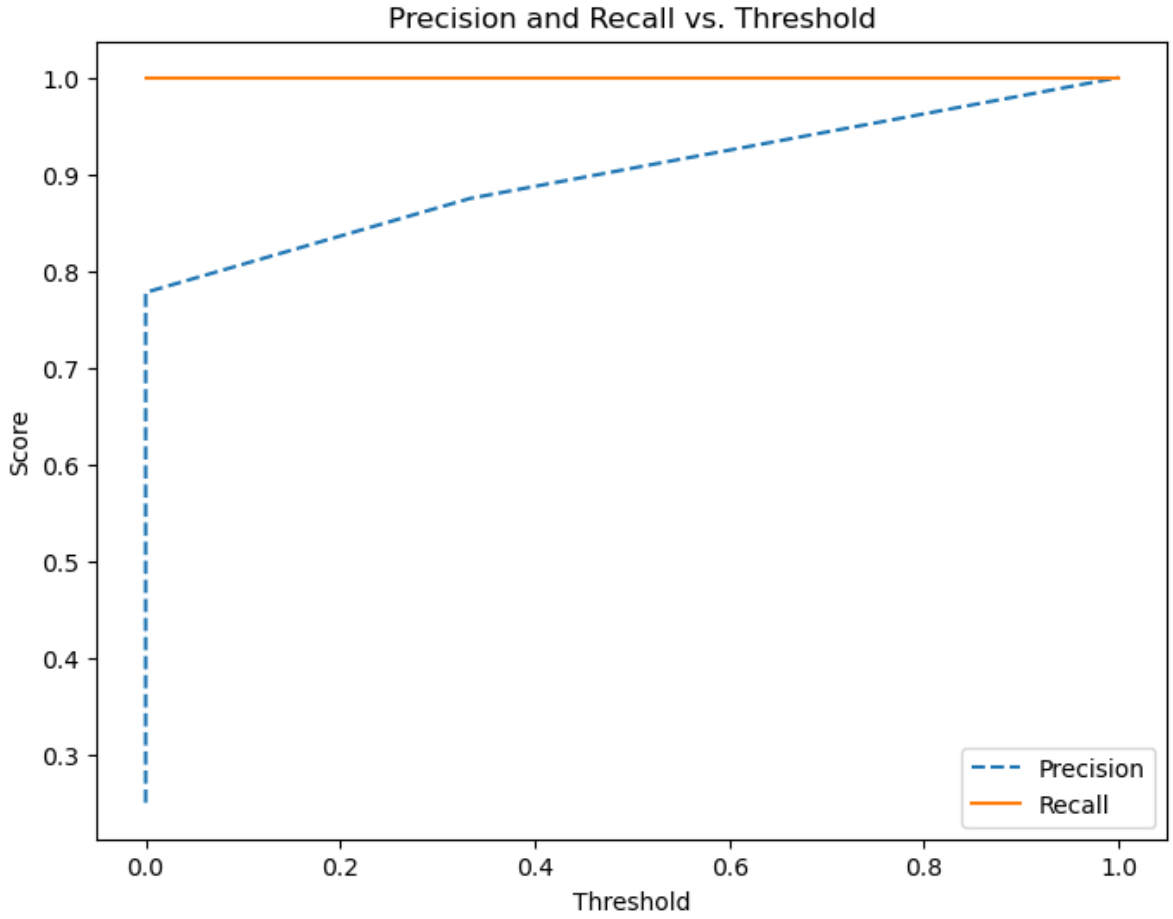
In [22]:
```
PrecisionRecallDisplay.from_estimator(modelo_best_estimator, val_x, val_y)
```

Out[22]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x2e65bafb
a50>
```



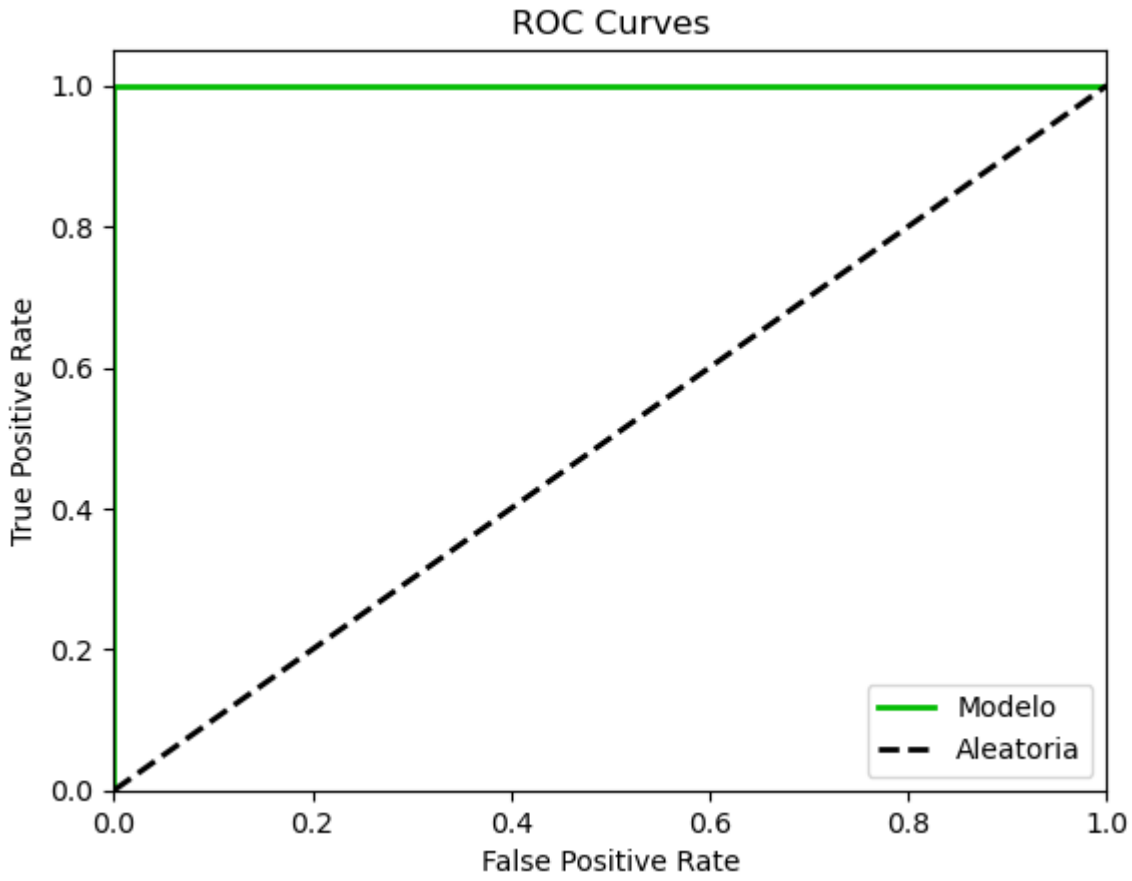
In [ ]:
```
# Calcula la curva de precisión y recall para diferentes umbrales de corte
precision, recall, thresholds = precision_recall_curve(val_y, pred_proba)

# Grafica la relación entre la precisión y el recall en función del umbral de corte
plt.figure(figsize=(8, 6))
plt.plot(thresholds, precision[:-1], label='Precision', linestyle='--')
plt.plot(thresholds, recall[:-1], label='Recall', linestyle='-')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision and Recall vs. Threshold')

# Encuentra el umbral que maximiza F1-score (o ajusta según tus necesidades)
f1_scores = 2 * (precision * recall) / (precision + recall)
best_threshold = thresholds[np.argmax(f1_scores)]
print("El mejor best_threshold:", best_threshold)
```

```
El mejor best_threshold: 1.0
```



ROC Chart

In [ ]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_roc(val_y, modelo.best_estimator_.predict_proba(val_x), ax=ax)

#Eliminamos la línea de los ceros y personalizamos la leyenda
ax.lines[0].remove()
ax.lines[1].remove()
ax.lines[1].remove()
plt.legend(labels = ['Modelo','Aleatoria']);
```



GUARDAR BEST_ESTIMATOR, PARÁMETROS Y RESULTADOS DEL TEST Y LA VALIDACIÓN

Guardar el mejor estimador

In [ ]:
```
version_estimator = '_v0'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[ ]:
```
'LabelPropagation_v0.pickle'
```

In [ ]:
```
m_best_estimator
```

Out[ ]:
```
'LabelPropagation'
```

In [ ]:
```
ruta_pipe_entrenamiento = '../../04_Modelos/Best_estimator/' + nombre_best_estimato

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(modelo_best_estimator, file)
```

In [ ]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Gridsearch con modelos bases"
x_columns = list(x.columns)
y_target = y.name
```

In [ ]:
```
#Cargamos la lista con los resultados
resultado  = {'m_Best_estimator': m_best_estimator,
              'm_Best_paramans' : m_best_params,
              'm_Best_Score': m_best_score,
```

```
                't_accuracy': t_accuracy,
                't_report': t_report,
                'v_roc_auc_proba': v_roc_auc_proba,
                'v_roc_auc': v_roc_auc,
                'v_accuracy': v_accuracy,
                'v_report': v_report,
                'comentarios': comentarios,
                'predictoras_X': x_columns,
                'target_y': y_target
            }
resultado= pd.Series(resultado,name=nombre_best_estimator)
resultado
```

Out[ ]:
```
m_Best_estimator                                    LabelPropagation
m_Best_paramans                       {'algoritmo': LabelPropagation()}
m_Best_Score                                                        1.0
t_accuracy                                                          1.0
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                                     1.0
v_roc_auc                                                           1.0
v_accuracy                                                          1.0
v_report                          precision    recall  f1-score   ...
comentarios                                 Gridsearch con modelos bases
predictoras_X       [industrial_risk_oe, management_risk_oe, finan...
target_y                                                       target
Name: LabelPropagation_v0.pickle, dtype: object
```

In [ ]:
```
df_best = pd.read_excel('../../04_Modelos/Best_estimator/Best_estimator.xlsx',index
```

In [ ]:
```
df_best = pd.concat([df_best, resultado], ignore_index= False,axis=1)
df_best.to_excel('../../04_Modelos/Best_estimator/Best_estimator.xlsx')
```

CONCLUSIÓN: Vemos que el modelo ha dado muy buenos resultados con tanto con el modelo base de LabelPropagation() tanto para los datos de entrenamiento como los de validación.

PRÓXIMOS PASOS:

Realizaremos el modelo definitivo con este modelo. Si detectamos que el modelo está sobre ajustado, haremos un nuevo modelo con hiperparámetros para evitar el sobre ajuste y evaluaremos nuevamente el modelo.

5_01 - MODELO CLASIFICACIÓN CON EL **LABEL PROPAGATION**

En este notebook vamos a crear los pipeline de entrenamiento y ejecución.

IMPORTACIÓN DE PAQUETES

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scikitplot as skplt

#transformación de datos
from sklearn.preprocessing import OrdinalEncoder

#modelo

from sklearn.semi_supervised import LabelPropagation

#metricas de evaluación
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTAR LOS DATOS

In [2]:
```
df = pd.read_csv('../../02_Datos/03_Trabajo/tablon_analitico.csv', index_col= 0)
df
```

Out[2]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
1	N	N	A	A	A	
2	A	A	A	A	A	
3	P	P	P	P	P	
4	N	N	P	P	P	
5	A	A	P	P	P	
...	...	...	...	...	...	
178	A	P	N	A	N	
179	N	N	N	P	N	
182	N	N	N	P	N	
215	N	A	P	A	N	
234	N	A	N	N	N	

91 rows × 7 columns

SEPARAMOS PREDICTORAS Y TARGET

In [3]:
```
x = df.drop(columns= 'target').copy()
y = df.target.copy().reset_index()
```

TRANSFORMACIÓN DE VARIABLES

Variables a aplicar Ordinal Encoder

In [4]:
```
var_oe = ['industrial_risk',
          'management_risk',
          'financial_flexibility',
          'credibility',
          'competitiveness',
          'operating_risk']
```

Orden y categoría de las variables

In [5]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [6]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [7]:
```
df_oe = oe.fit_transform(x[var_oe])
```

In [8]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
df_oe = pd.DataFrame(df_oe, columns= nombre_oe)
df_oe
```

Out[8]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [9]:
```
y = y.replace({'B':1 , 'NB': 0})
y.drop(columns= 'index', inplace= True)
y
```

Out[9]:

	target
0	0
1	0
2	0
3	0
4	0
...	...
86	1
87	1
88	1
89	1
90	1

91 rows × 1 columns

AGRUPAR PREDICTORAS Y TARGET EN UN DATASET

In [10]:
```
df_tablon = pd.concat([df_oe, y], axis= 1, ignore_index= False)
df_tablon
```

Out[10]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 7 columns

SEPARAMOS PREDICTORAS Y TARGET

In [11]:
```
x = df_tablon.drop(columns= 'target').copy()
y = df_tablon.target.copy().reset_index()
y.drop(columns= 'index', inplace= True)
```

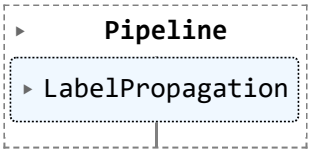
MODELIZAR

CARGAR EL MEJOR MODELO CON EL ALGORITMO, PARÁMETROS Y VALORES

In [12]:
```
modelo = pd.read_pickle('../../04_Modelos/Best_Estimator/LabelPropagation_v0.pickle
```

In [13]:
```
modelo.best_estimator_
```

Out[13]:



In [14]:
```
modelo.get_params
```

Out[14]:
```
<bound method BaseEstimator.get_params of GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('algoritmo', LogisticRegression())]),
             n_jobs=-1,
             param_grid=[{'algoritmo': [LabelPropagation()]},
                         {'algoritmo': [RidgeClassifier()]},
                         {'algoritmo': [RidgeClassifierCV()]},
                         {'algoritmo': [LogisticRegression()]},
                         {'algoritmo': [LGBMClassifier()]},
                         {'algoritmo': [NuSVC()]}],
             scoring='roc_auc')>
```

PREDECIR SOBRE LA VALIDACIÓN

In [15]:
```
pred = modelo.best_estimator_.predict(x)
pred_proba = modelo.best_estimator_.predict_proba(x)[:,1]
```

EVALUAR SOBRE LA VALIDACIÓN

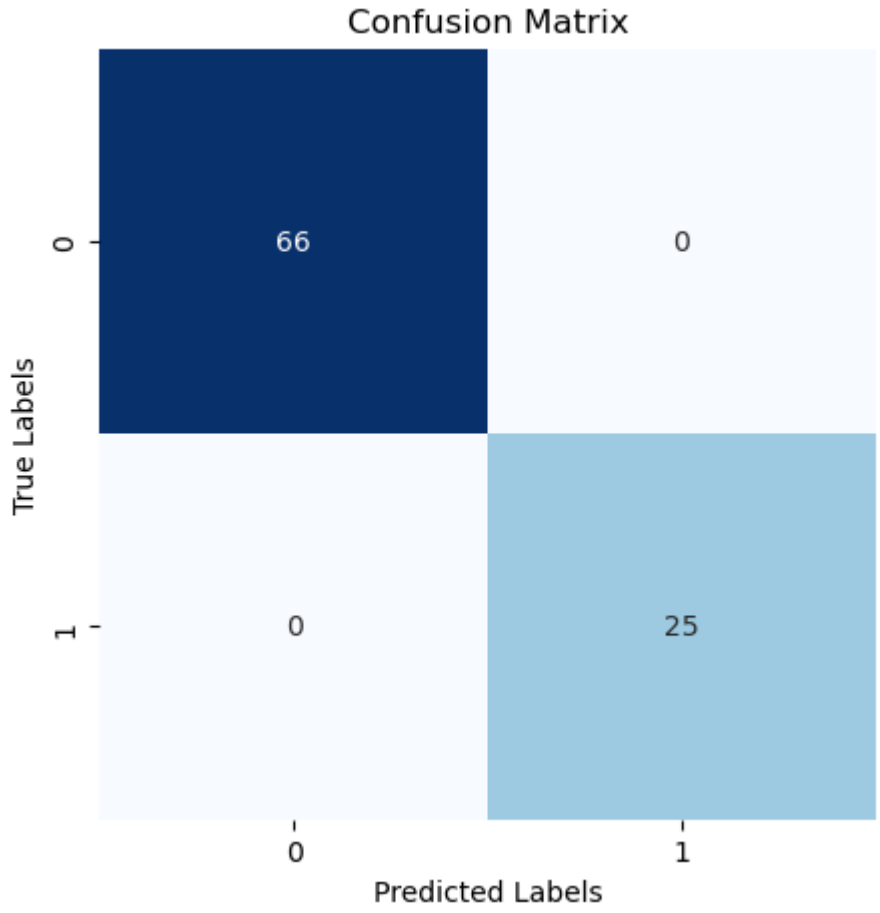
In [16]:
```
v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

```
Roc AUC_proba: 1.0
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:              precision    recall  f1-score   support

           0       1.00      1.00      1.00        66
           1       1.00      1.00      1.00        25

    accuracy                           1.00        91
   macro avg       1.00      1.00      1.00        91
weighted avg       1.00      1.00      1.00        91
```

REPORTING DEL MODELO

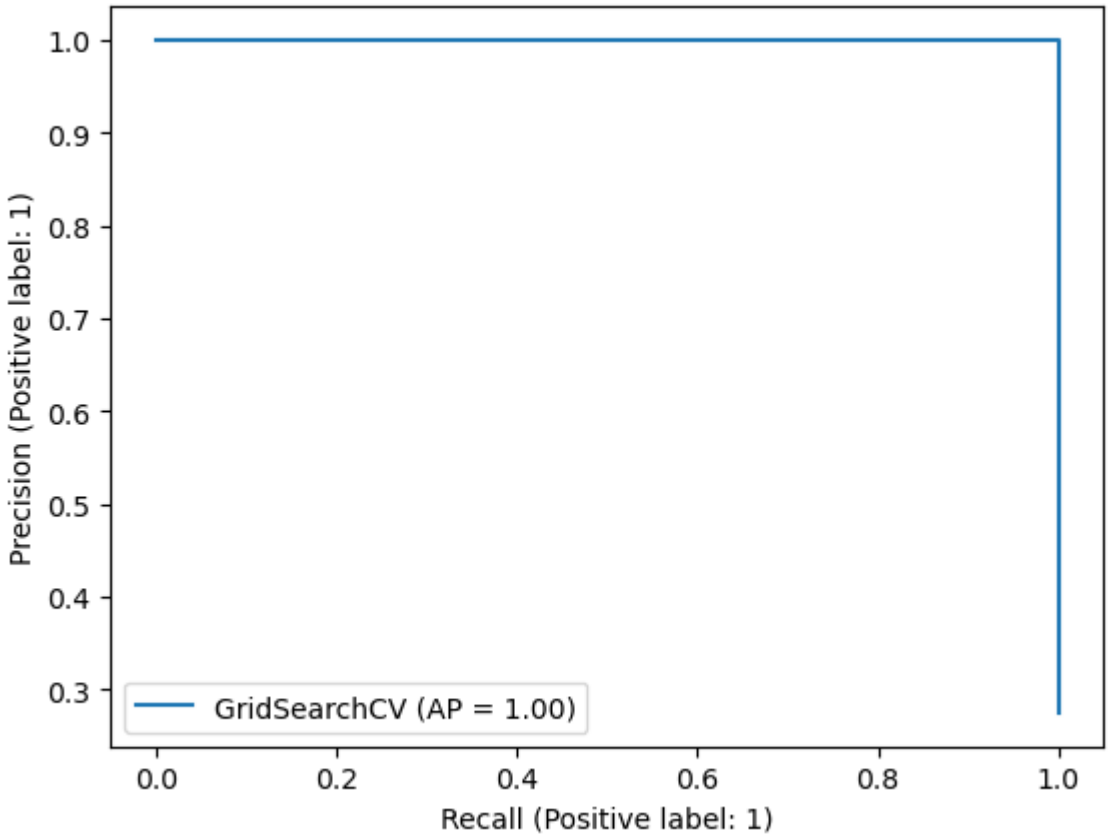
Matrix de Confusión MultiClass

In [17]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)

# Configurar etiquetas y título del gráfico
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



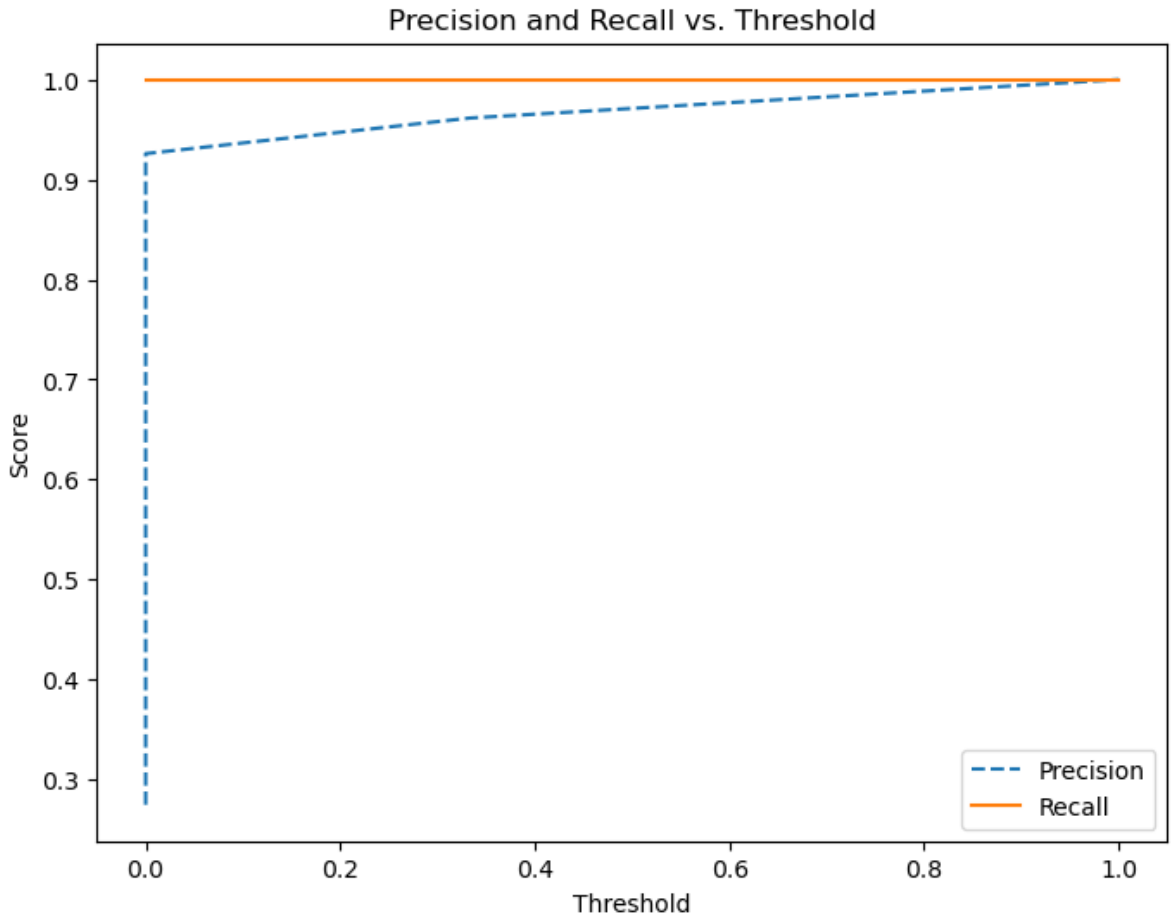
Precision-Recall

In [18]:
```
PrecisionRecallDisplay.from_estimator(modelo, x, y);
```



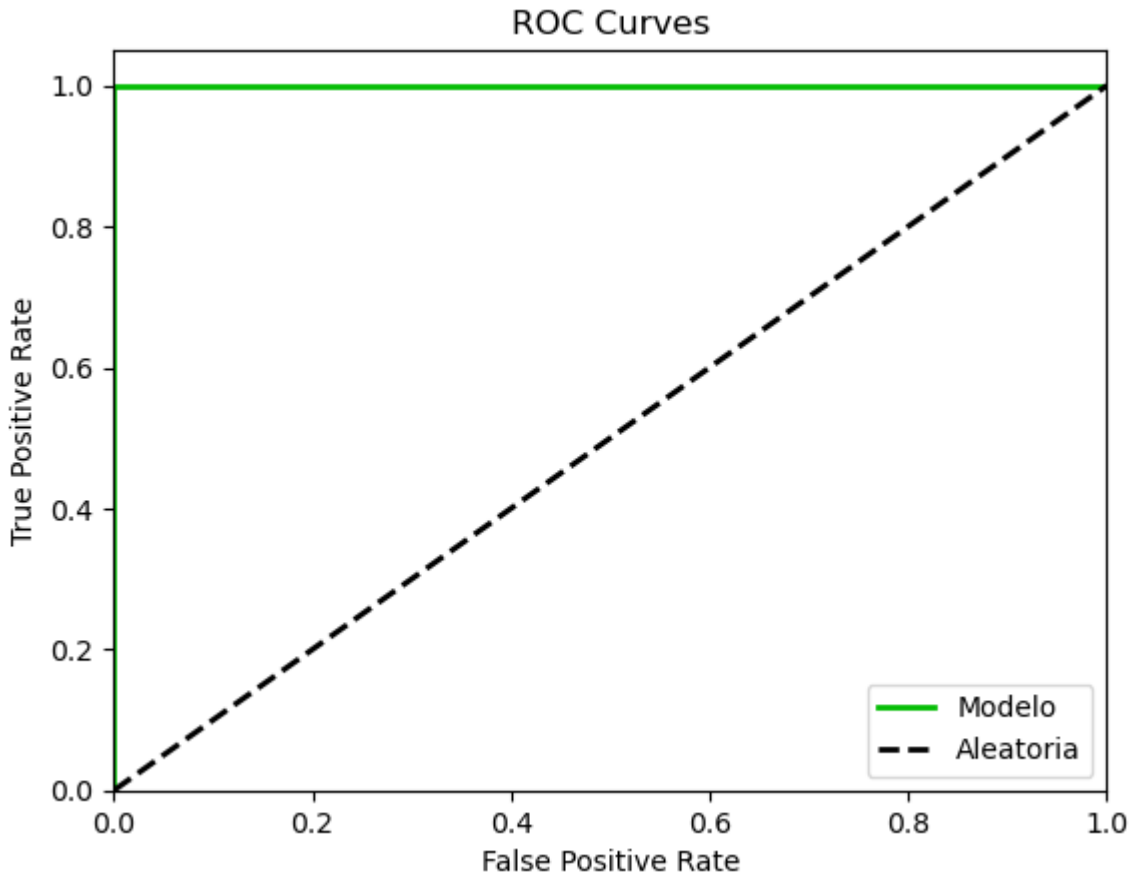
In [19]:
```
# Calcula la curva de precisión y recall para diferentes umbrales de corte
precision, recall, thresholds = precision_recall_curve(y, pred_proba)

# Grafica la relación entre la precisión y el recall en función del umbral de corte
plt.figure(figsize=(8, 6))
plt.plot(thresholds, precision[:-1], label='Precision', linestyle='--')
plt.plot(thresholds, recall[:-1], label='Recall', linestyle='-')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision and Recall vs. Threshold')

# Encuentra el umbral que maximiza F1-score (o ajusta según tus necesidades)
f1_scores = 2 * (precision * recall) / (precision + recall)
best_threshold = thresholds[np.argmax(f1_scores)]
print("El mejor best_threshold:", best_threshold)
```

```
El mejor best_threshold: 1.0
```



ROC Chart

In [20]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_roc(y, modelo.best_estimator_.predict_proba(x), ax=ax)

#Eliminamos la línea de los ceros y personalizamos la leyenda
ax.lines[0].remove()
ax.lines[1].remove()
ax.lines[1].remove()
plt.legend(labels = ['Modelo','Aleatoria']);
```



CREAR PIPELINE DE ENTRENAMIENTO Y EJECUCIÓN

INSTANCIAR EL MODELO

In [21]:
```
modelo = LabelPropagation()
```

CREAR Y GUARDAR EL PIPE FINAL DE ENTRENAMIENTO

In [22]:
```
# Crear pipe de entrenamiento

pipe_entrenamiento = make_pipeline(modelo)
```

In [23]:
```
# Guardar pipe de entrenamiento
nombre_pipe_entrenamiento = 'pipe_entrenamiento.pickle'
ruta_pipe_entrenamiento = '../../04_Modelos/' + nombre_pipe_entrenamiento

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(pipe_entrenamiento, file)
```

ENTRENAR Y GUARDAR EL PIPE DE EJECUCIÓN

In [24]:
```
#Entrenar pipe de entrenamiento

pipe_ejecucion = pipe_entrenamiento.fit(x,y)
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [25]:
```
nombre_pipe_ejecucion = 'pipe_ejecucion.pickle'
ruta_pipe_ejecucion = '../../04_Modelos/' + nombre_pipe_ejecucion

with open (ruta_pipe_ejecucion, mode= 'wb') as file:
    cloudpickle.dump(pipe_ejecucion, file)
```

5_ 2 - MODELO DE EJECUCIÓN CON LOS DATOS DE PRUEBA CON EL **LABEL PROPAGATION**

En este notebook vamos a cargar el dataset de prueba y realizaremos la calidad de datos, separaremos predictoras y target, predeciremos sobre los datos de prueba y evaluaremos el modelo definitivo.

IMPORTAR LOS PAQUETES

In [14]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scikitplot as skplt

#transformación de datos
from sklearn.preprocessing import OrdinalEncoder

#modelo

from sklearn.semi_supervised import LabelPropagation

#metricas de evaluación
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

CARGAR LOS DATOS

In [15]:
```
df = pd.read_csv('../../02_Datos/02_Validacion/prueba.csv', index_col= 0)
df
```

Out[15]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
190	N	N	N	N	N	
10	P	P	P	P	A	
191	P	N	N	N	A	
168	A	A	N	N	N	
130	A	A	A	A	P	
...	...	...	...	...	...	
60	A	A	A	P	P	
37	A	N	A	P	P	
22	A	A	A	A	P	
77	A	P	A	P	A	
200	N	N	N	N	N	

62 rows × 7 columns

TRANSFORMACIÓN DE DATOS

Variables a aplicar Ordinal Encoder

In [16]:
```
var_oe = ['industrial_risk',
            'management_risk',
            'financial_flexibility',
            'credibility',
            'competitiveness',
            'operating_risk']
```

Orden y categoría de las variables

In [17]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [18]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [19]:
```
df_oe = oe.fit_transform(df[var_oe])
```

In [20]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
x = pd.DataFrame(df_oe, columns= nombre_oe)
x
```

Out[20]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	0.0	0.0	0
1	2.0	2.0	2.0	2.0	1
2	2.0	0.0	0.0	0.0	1
3	1.0	1.0	0.0	0.0	0
4	1.0	1.0	1.0	1.0	2
...	...	...	...	...	
57	1.0	1.0	1.0	2.0	2
58	1.0	0.0	1.0	2.0	2
59	1.0	1.0	1.0	1.0	2
60	1.0	2.0	1.0	2.0	1
61	0.0	0.0	0.0	0.0	0

62 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [25]:
```
y = df.target.replace({'B':1 , 'NB': 0}).reset_index().copy()
y.drop(columns= 'index', inplace= True)
y
```

Out[25]:

	target
0	1
1	0
2	1
3	1
4	0
...	...
57	0
58	0
59	0
60	0
61	1

62 rows × 1 columns

MODELIZAR CON EL PIPE DE EJECUCIÓN

CARGAMOS EL PIPE DE EJECUCIÓN

In [29]:
```
modelo = pd.read_pickle('../../04_Modelos/pipe_ejecucion.pickle')
```

PREDECIR Y EVALUAR CON LOS DATOS DE PRUEBA

PREDECIR SOBRE LOS DATOS

In [30]:
```
pred = modelo.predict(x)
pred_proba = modelo.predict_proba(x)[:,1]
```

EVALUAR SOBRE LOS DATOS

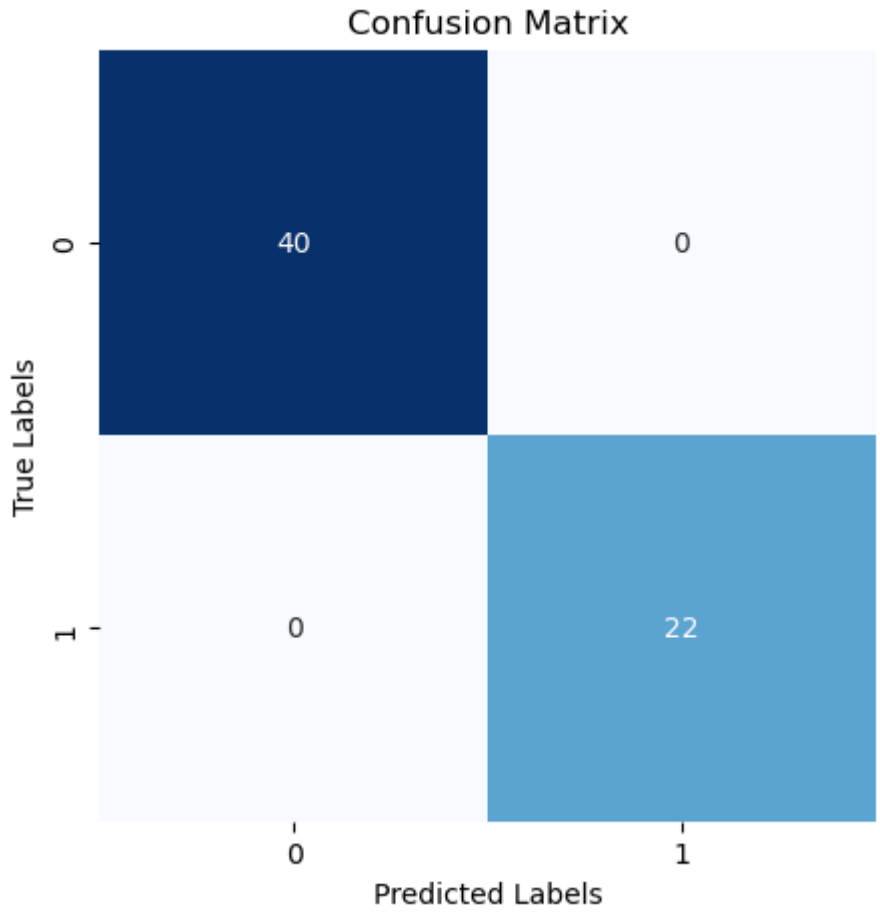
In [32]:
```
v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

```
Roc AUC_proba: 1.0
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        22

    accuracy                           1.00        62
   macro avg       1.00      1.00      1.00        62
weighted avg       1.00      1.00      1.00        62
```

REPORTING DEL MODELO

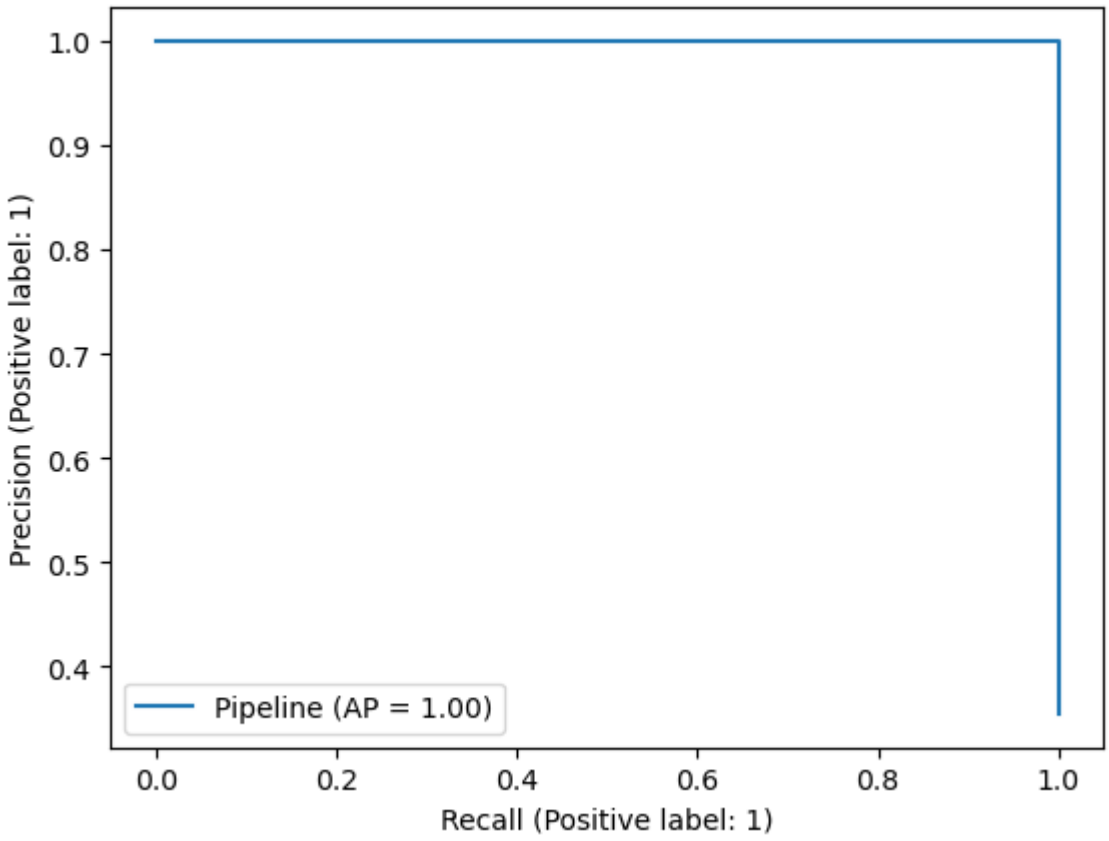
Matrix de Confusión MultiClass

In [33]:
```python
# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)

# Configurar etiquetas y título del gráfico
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



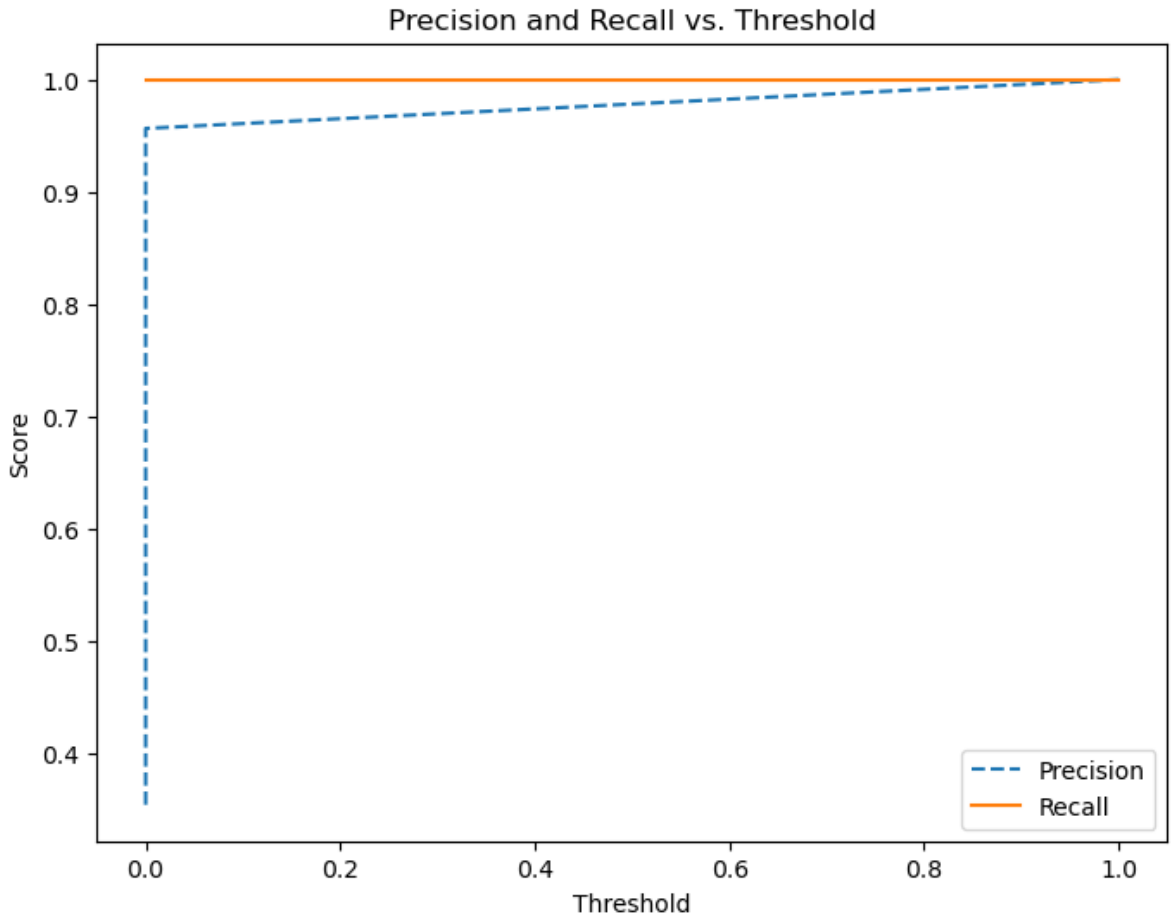
Precision-Recall

In [34]:
```
PrecisionRecallDisplay.from_estimator(modelo, x, y);
```



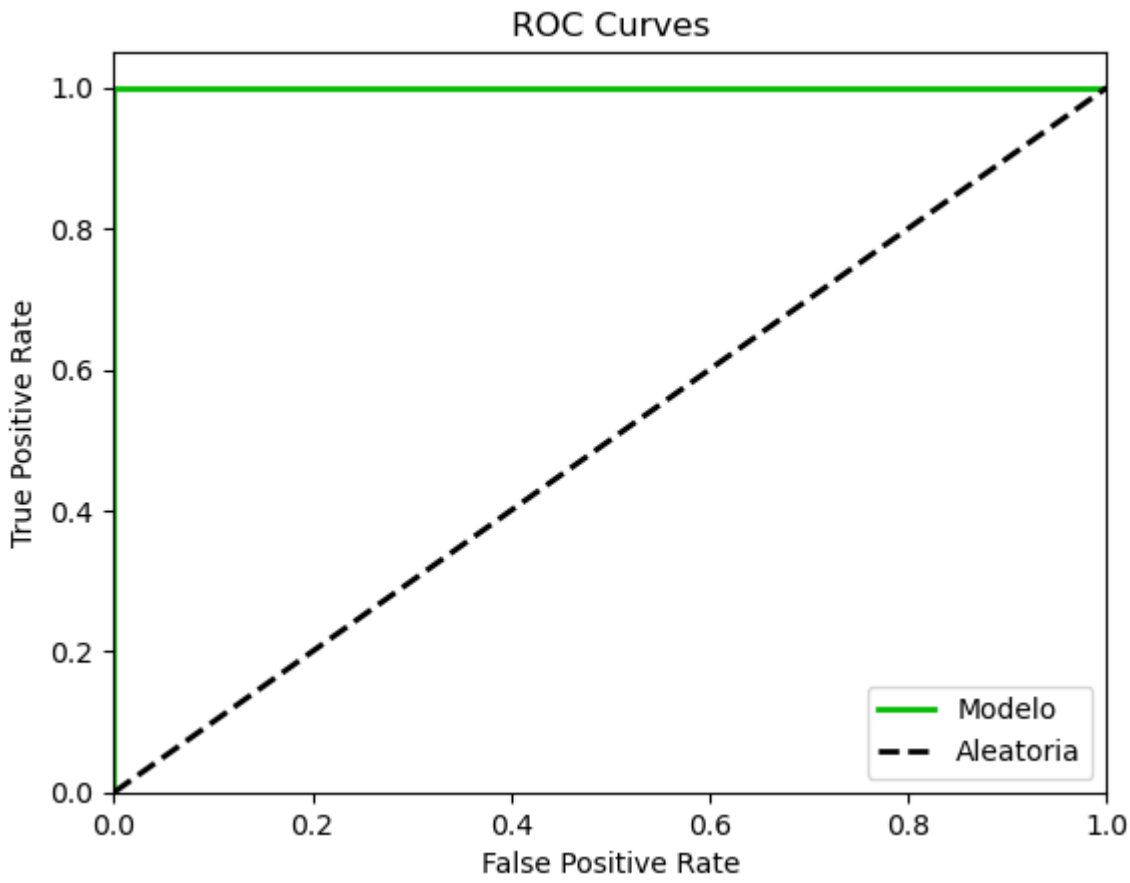
In [35]:
```
# Calcula la curva de precisión y recall para diferentes umbrales de corte
precision, recall, thresholds = precision_recall_curve(y, pred_proba)

# Grafica la relación entre la precisión y el recall en función del umbral de corte
plt.figure(figsize=(8, 6))
plt.plot(thresholds, precision[:-1], label='Precision', linestyle='--')
plt.plot(thresholds, recall[:-1], label='Recall', linestyle='-')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision and Recall vs. Threshold')

# Encuentra el umbral que maximiza F1-score (o ajusta según tus necesidades)
f1_scores = 2 * (precision * recall) / (precision + recall)
best_threshold = thresholds[np.argmax(f1_scores)]
print("El mejor best_threshold:", best_threshold)
```

```
El mejor best_threshold: 1.0
```



ROC Chart

In [36]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_roc(y, modelo.predict_proba(x), ax=ax)

#Eliminamos la línea de los ceros y personalizamos la leyenda
ax.lines[0].remove()
ax.lines[1].remove()
ax.lines[1].remove()
plt.legend(labels = ['Modelo','Aleatoria']);
```



CONCLUSIÓN:

El modelo ha funcionado muy bien con los datos de prueba obteniendo un 100% roc_auc sobre los [1].

PRÓXIMOS PASOS: Al ser un scoring de riesgo bancario muchas veces se precisa de una interpretación de los datos para comunicar el motivo al cliente. Para por interpretar podremos a competir a algoritmos como la Regresión Logística, Ridge, Lasso o Arboles de decisión.

6 THE BEST ESTIMATOR - MODELO BASE - INTERPRETACIÓN DE LOS DATOS (Fx)

Queremos poder interpretar los datos del modelo mediante una formula para exportarla. Vamos a poner a competir a los siguientes modelos.

- Ridge Classifier
- Logistic Regression
- MultinomialNB

MODELAR ALGORITMO DE CLASIFICACIÓN

IMPORTACIÓN DE PAQUETES

In [144…

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

# Crear train and test
from sklearn.model_selection import train_test_split

#Modelos seleccionados por LazyPredict
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import RidgeClassifierCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

#Optimizar modelos
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import RocCurveDisplay

import scikitplot as skplt

#Crear Pipeline

from sklearn.pipeline import Pipeline

import cloudpickle

import warnings
warnings.filterwarnings("ignore")
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

In [145…
```
df = pd.read_pickle('../../02_Datos/03_Trabajo/df_tablon_oe.pickle')
df.head()
```

Out[145]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_oe
0	0.0	0.0	1.0	1.0	1.(
1	1.0	1.0	1.0	1.0	1.(
2	2.0	2.0	2.0	2.0	2.(
3	0.0	0.0	2.0	2.0	2.(
4	1.0	1.0	2.0	2.0	2.(

SEPARAR PREDICTORAS Y TARGET

In [146…
```
x = df.drop(columns= 'target').copy()
y = df.target.copy()
```

In [147…
```
x.head()
```

Out[147]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_oe
0	0.0	0.0	1.0	1.0	1.(
1	1.0	1.0	1.0	1.0	1.(
2	2.0	2.0	2.0	2.0	2.(
3	0.0	0.0	2.0	2.0	2.(
4	1.0	1.0	2.0	2.0	2.(

In [148…
```
y.head()
```

Out[148]:
```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [149…
```
train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_stat
```

CREAR PIPE Y EL DICCIONARIO CON LOS ALGORITMOS, PARÁMETROS Y VALORES

In [150…]
```
pipe = Pipeline([('algoritmo', LogisticRegression())])

grid = [
            {
                'algoritmo' : [RidgeClassifier()]
            },

            {
                'algoritmo' : [RidgeClassifierCV()]
            },

            {
                'algoritmo' : [LogisticRegression()]
            },

            {
                'algoritmo' : [MultinomialNB()]
            }
        ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

In [151…]
```
grid_search = GridSearchCV(estimator= pipe,
                           param_grid= grid,
                           cv = 5,
                           scoring= 'roc_auc',
                           n_jobs= -1,
                           verbose=0)
```

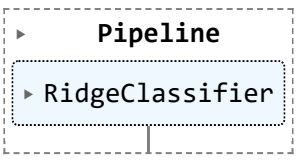
In [152…]
```
modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

Out[152]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	
1	0.014670	0.002945	0.005985	0.001092	RidgeClassifier()	{' Ridge
2	0.008387	0.001361	0.004988	0.000894	RidgeClassifierCV()	{' RidgeCla
3	0.008188	0.000400	0.004189	0.000399	LogisticRegression()	{' LogisticRe
0	0.006857	0.006498	0.005557	0.003328	DecisionTreeClassifier()	{' DecisionTree
4	0.004389	0.000488	0.003591	0.000798	MultinomialNB()	{' Multir

In [153…]
```
modelo.best_estimator_
```

Out[153]:



In [154…]
```
modelo.best_params_
```

Out[154]:
```
{'algoritmo': RidgeClassifier()}
```

In [155…
```
modelo.best_score_
```

Out[155]:
```
1.0
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [156…
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [157…
```
m_best_estimator = str(modelo.best_estimator_[0])
m_best_estimator = m_best_estimator.split('(')[0]
m_best_params = str(modelo.best_params_)
m_best_score = str(modelo.best_score_)
```

EVALUAR

PRECEDIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el train

In [158…
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [159…
```
t_roc_auc = roc_auc_score(train_y, pred)
t_accuracy = accuracy_score(train_y, pred)
t_report = classification_report(train_y, pred)

print(f"Roc AUC: {t_roc_auc}")
print(f"Accuracy: {t_accuracy}")
print(f"Classification Report:\n{t_report}")
```

```
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        45
           1       1.00      1.00      1.00        18

    accuracy                           1.00        63
   macro avg       1.00      1.00      1.00        63
weighted avg       1.00      1.00      1.00        63
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

In [160…
```
pred = modelo.best_estimator_.predict(val_x)
#pred_proba = modelo.best_estimator_.predict_proba(val_x)[:,1]
pred_proba = 'N/A'
```

Evaluar sobre la validación

In [161…

```
#v_roc_auc_proba = roc_auc_score(val_y, pred_proba)
v_roc_auc = roc_auc_score(val_y, pred)
v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

#print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

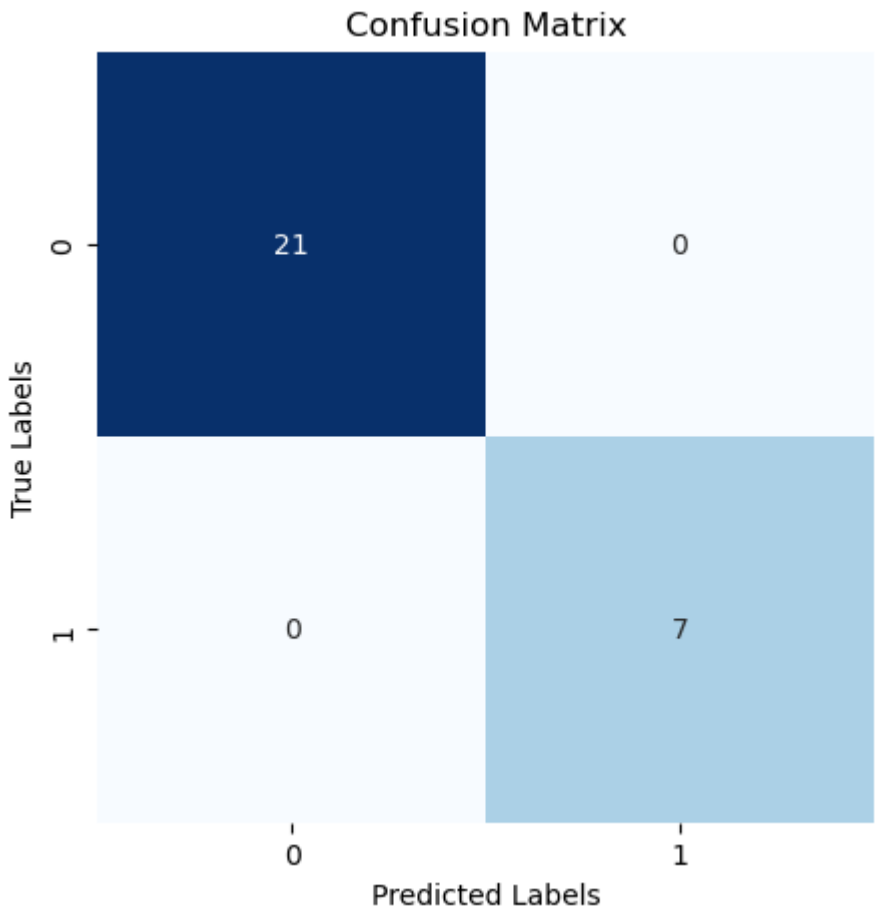
```
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00         7

    accuracy                           1.00        28
   macro avg       1.00      1.00      1.00        28
weighted avg       1.00      1.00      1.00        28
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass

In [164…

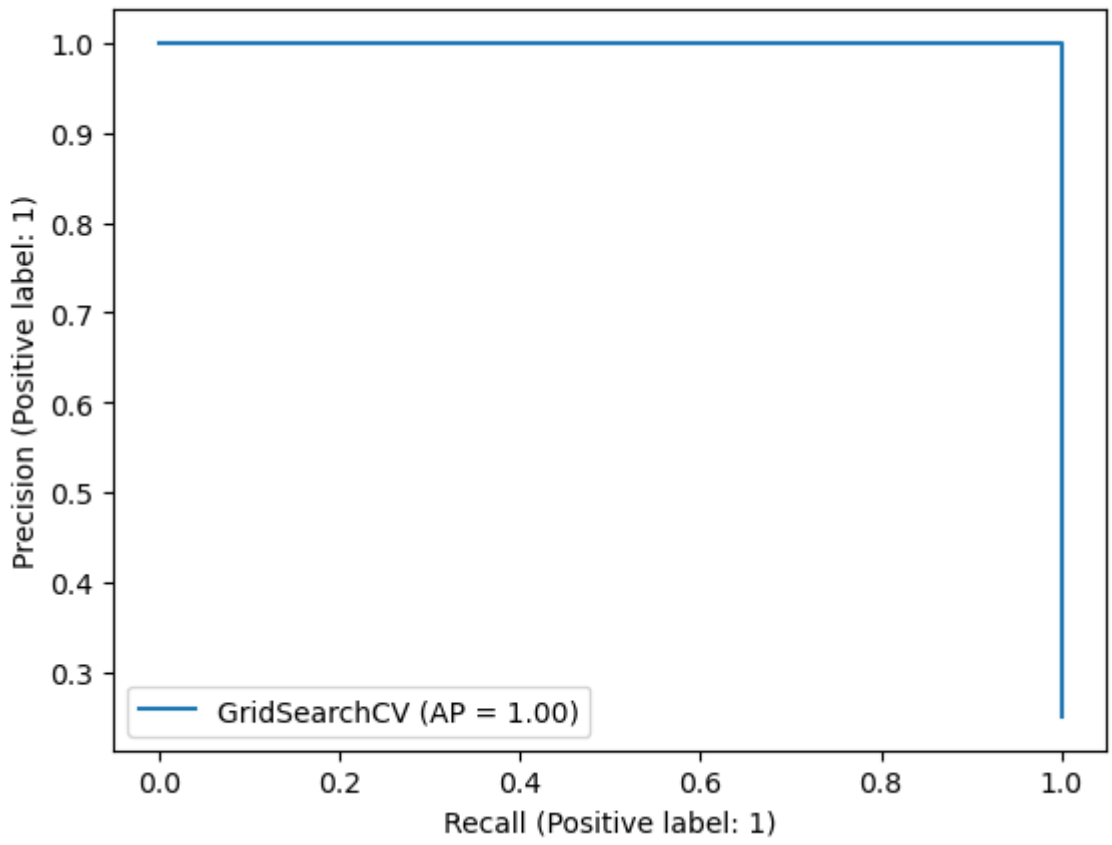
```
# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)

# Configurar etiquetas y título del gráfico
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



Precision-Recall

In [165…
```
PrecisionRecallDisplay.from_estimator(modelo_best_estimator, val_x, val_y)
```

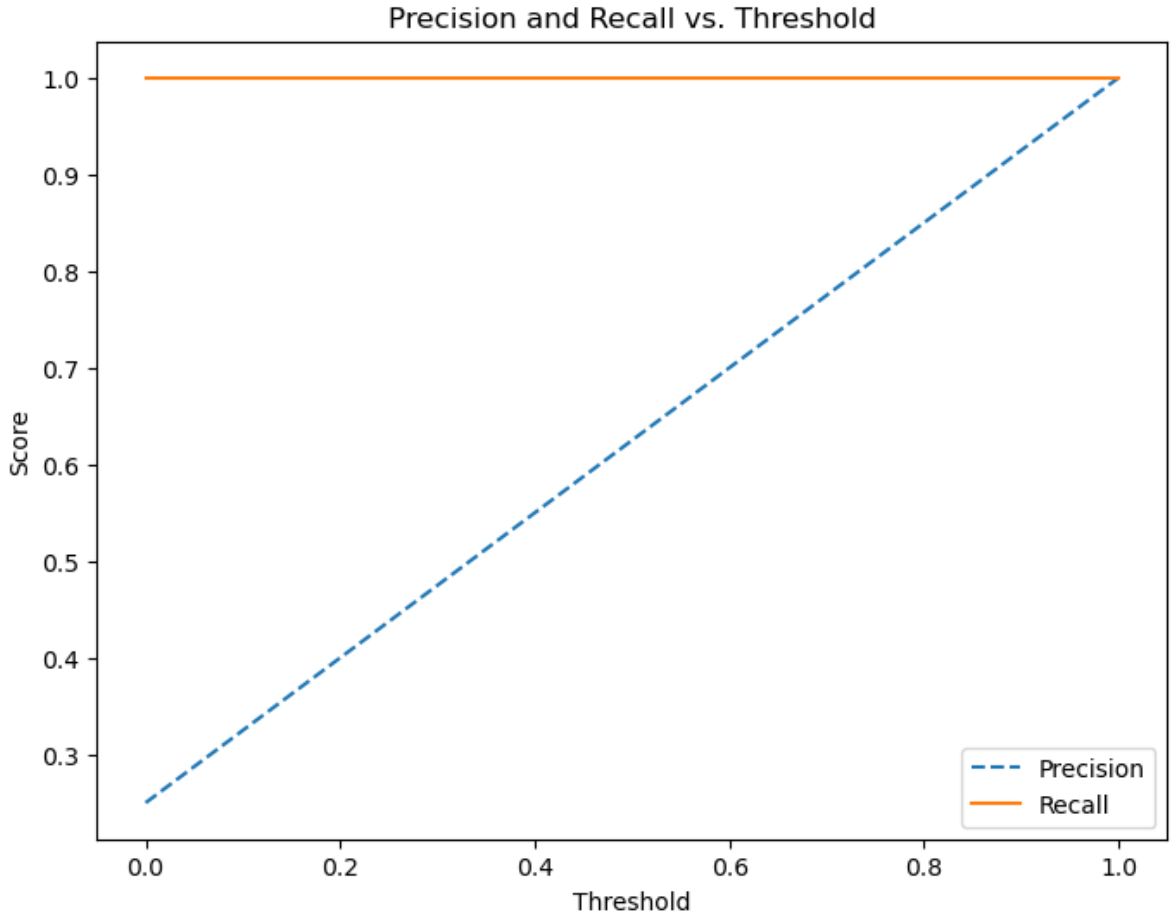
Out[165]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x20d19111a50>
```



In [166…

```
# Calcula la curva de precisión y recall para diferentes umbrales de corte
precision, recall, thresholds = precision_recall_curve(val_y, pred)

# Grafica la relación entre la precisión y el recall en función del umbral de corte
plt.figure(figsize=(8, 6))
plt.plot(thresholds, precision[:-1], label='Precision', linestyle='--')
plt.plot(thresholds, recall[:-1], label='Recall', linestyle='-')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.title('Precision and Recall vs. Threshold')

# Encuentra el umbral que maximiza F1-score (o ajusta según tus necesidades)
f1_scores = 2 * (precision * recall) / (precision + recall)
best_threshold = thresholds[np.argmax(f1_scores)]
print("El mejor best_threshold:", best_threshold)
```

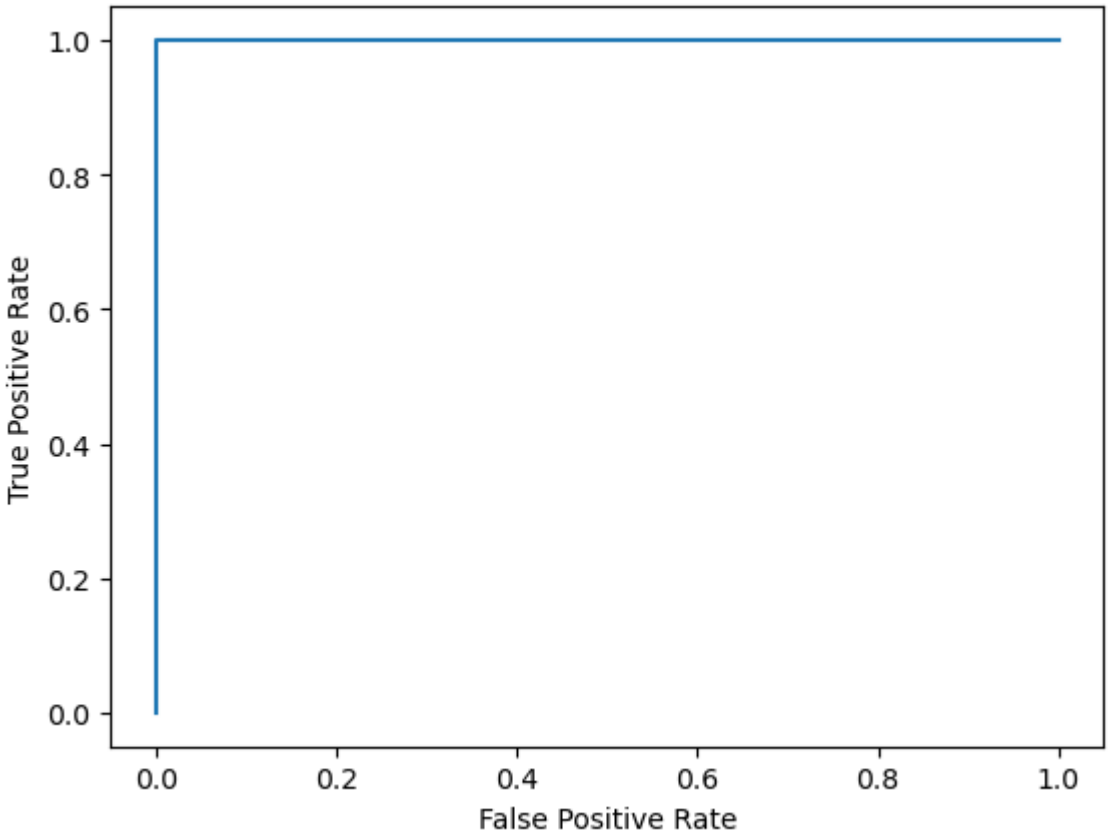
```
El mejor best_threshold: 1
```



ROC Chart

In [167…

```
y_score = modelo.best_estimator_.decision_function(val_x)
fpr, tpr, _ = roc_curve(val_y, y_score, pos_label=modelo.best_estimator_.classes_[1
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```



GUARDAR BEST_ESTIMATOR, PARÁMETROS Y RESULTADOS DEL TEST Y LA VALIDACIÓN

Guardar el mejor estimador

In [168…
```
version_estimator = '_v1'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[168]:
```
'RidgeClassifier_v1.pickle'
```

In [169…
```
m_best_estimator
```

Out[169]:
```
'RidgeClassifier'
```

In [170…
```
ruta_pipe_entrenamiento = '../../04_Modelos/Best_estimator/' + nombre_best_estimato

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(modelo_best_estimator, file)
```

In [171…
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Gridsearch con modelos bases - Intepretar resultado"
x_columns = list(x.columns)
y_target = y.name
```

In [172…
```
#Cargamos la lista con los resultados
resultado  = {'m_Best_estimator': m_best_estimator,
              'm_Best_paramans' : m_best_params,
              'm_Best_Score': m_best_score,
              't_accuracy': t_accuracy,
              't_report': t_report,
```

```
                'v_roc_auc_proba': 'n/a',
                'v_roc_auc': v_roc_auc,
                'v_accuracy': v_accuracy,
                'v_report': v_report,
                'comentarios': comentarios,
                'predictoras_X': x_columns,
                'target_y': y_target
            }
resultado= pd.Series(resultado,name=nombre_best_estimator)
resultado
```

Out[172]:

```
m_Best_estimator                                        RidgeClassifier
m_Best_paramans                        {'algoritmo': RidgeClassifier()}
m_Best_Score                                                       1.0
t_accuracy                                                         1.0
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                                    n/a
v_roc_auc                                                          1.0
v_accuracy                                                         1.0
v_report                          precision    recall  f1-score   ...
comentarios         Gridsearch con modelos bases - Intepretar resu...
predictoras_X       [industrial_risk_oe, management_risk_oe, finan...
target_y                                                       target
Name: RidgeClassifier_v1.pickle, dtype: object
```

In [173…

```
df_best = pd.read_excel('../../04_Modelos/Best_estimator/Best_estimator.xlsx',index
```

In [174…

```
df_best = pd.concat([df_best, resultado], ignore_index= False,axis=1)
df_best.to_excel('../../04_Modelos/Best_estimator/Best_estimator.xlsx')
```

CONCLUSIÓN: Vemos que el modelo ha dado muy buenos resultados para los datos de entrenamiento como los de validación.

PRÓXIMOS PASOS:

Realizaremos el modelo definitivo con este modelo.

6_01 - MODELO CLASIFICACIÓN CON EL **RIDGE CLASSIFIER**

En este notebook vamos a crear los pipeline de entrenamiento y ejecución.

IMPORTACIÓN DE PAQUETES

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scikitplot as skplt

#transformación de datos
from sklearn.preprocessing import OrdinalEncoder

#modelo

from sklearn.linear_model import RidgeClassifier

#Métricas de evaluación
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import RocCurveDisplay
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTAR LOS DATOS

In [3]:
```
df = pd.read_csv('../../02_Datos/03_Trabajo/tablon_analitico.csv', index_col= 0)
df
```

Out[3]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
1	N	N	A	A	A	
2	A	A	A	A	A	
3	P	P	P	P	P	
4	N	N	P	P	P	
5	A	A	P	P	P	
...	...	...	...	...	...	
178	A	P	N	A	N	
179	N	N	N	P	N	
182	N	N	N	P	N	
215	N	A	P	A	N	
234	N	A	N	N	N	

91 rows × 7 columns

SEPARAMOS PREDICTORAS Y TARGET

In [4]:
```
x = df.drop(columns= 'target').copy()
y = df.target.copy().reset_index()
```

TRANSFORMACIÓN DE VARIABLES

Variables a aplicar Ordinal Encoder

In [5]:
```
var_oe = ['industrial_risk',
          'management_risk',
          'financial_flexibility',
          'credibility',
          'competitiveness',
          'operating_risk']
```

Orden y categoría de las variables

In [6]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [7]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [8]:
```
df_oe = oe.fit_transform(x[var_oe])
```

In [9]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
df_oe = pd.DataFrame(df_oe, columns= nombre_oe)
df_oe
```

Out[9]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [10]:
```
y = y.replace({'B':1 , 'NB': 0})
y.drop(columns= 'index', inplace= True)
y
```

Out[10]:

	target
0	0
1	0
2	0
3	0
4	0
...	...
86	1
87	1
88	1
89	1
90	1

91 rows × 1 columns

AGRUPAR PREDICTORAS Y TARGET EN UN DATASET

In [11]:
```
df_tablon = pd.concat([df_oe, y], axis= 1, ignore_index= False)
df_tablon
```

Out[11]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 7 columns

SEPARAMOS PREDICTORAS Y TARGET

In [12]:
```
x = df_tablon.drop(columns= 'target').copy()
y = df_tablon.target.copy().reset_index()
y.drop(columns= 'index', inplace= True)
```

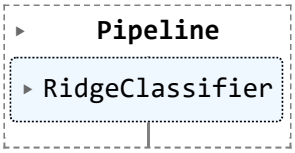
MODELIZAR

CARGAR EL MEJOR MODELO CON EL ALGORITMO, PARÁMETROS Y VALORES

In [13]:
```
modelo = pd.read_pickle('../../04_Modelos/Best_Estimator/RidgeClassifier_v1.pickle
```

In [14]:
```
modelo.best_estimator_
```

Out[14]:



In [15]:
```
modelo.get_params
```

Out[15]:
```
<bound method BaseEstimator.get_params of GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('algoritmo', LogisticRegression())]),
             n_jobs=-1,
             param_grid=[{'algoritmo': [DecisionTreeClassifier()]},
                         {'algoritmo': [RidgeClassifier()]},
                         {'algoritmo': [RidgeClassifierCV()]},
                         {'algoritmo': [LogisticRegression()]},
                         {'algoritmo': [MultinomialNB()]}],
             scoring='roc_auc')>
```

PREDECIR SOBRE LA VALIDACIÓN

In [17]:
```
pred = modelo.best_estimator_.predict(x)
#pred_proba = modelo.best_estimator_.predict_proba(x)[:,1]
```

EVALUAR SOBRE LA VALIDACIÓN

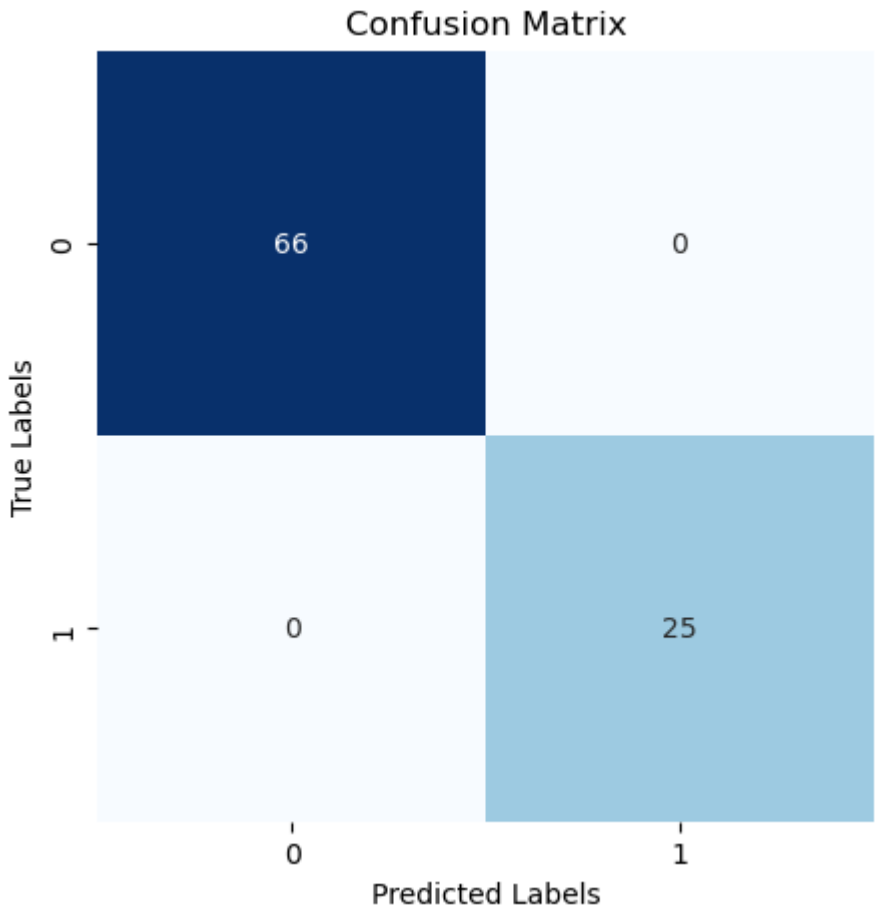
In [18]:
```
#v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

#print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

```
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:              precision    recall  f1-score   support

           0       1.00      1.00      1.00        66
           1       1.00      1.00      1.00        25

    accuracy                           1.00        91
   macro avg       1.00      1.00      1.00        91
weighted avg       1.00      1.00      1.00        91
```

REPORTING DEL MODELO

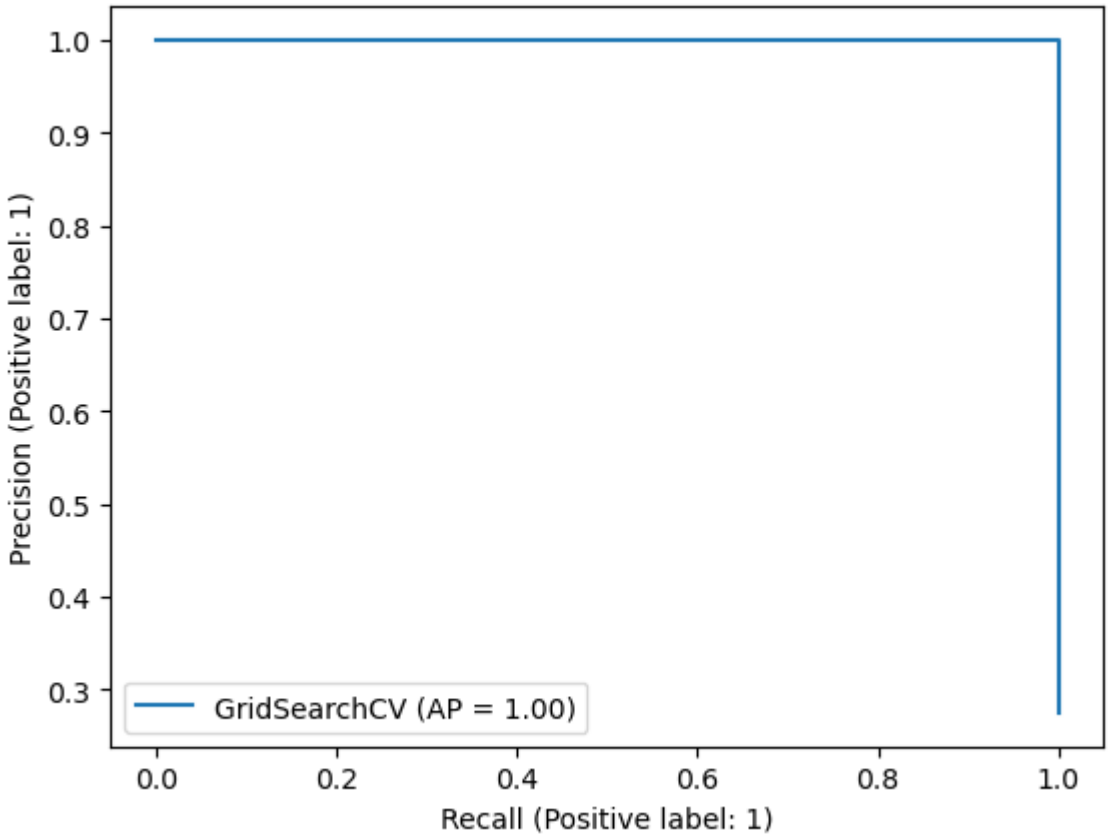
Matrix de Confusión MultiClass

In [19]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)

# Configurar etiquetas y título del gráfico
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



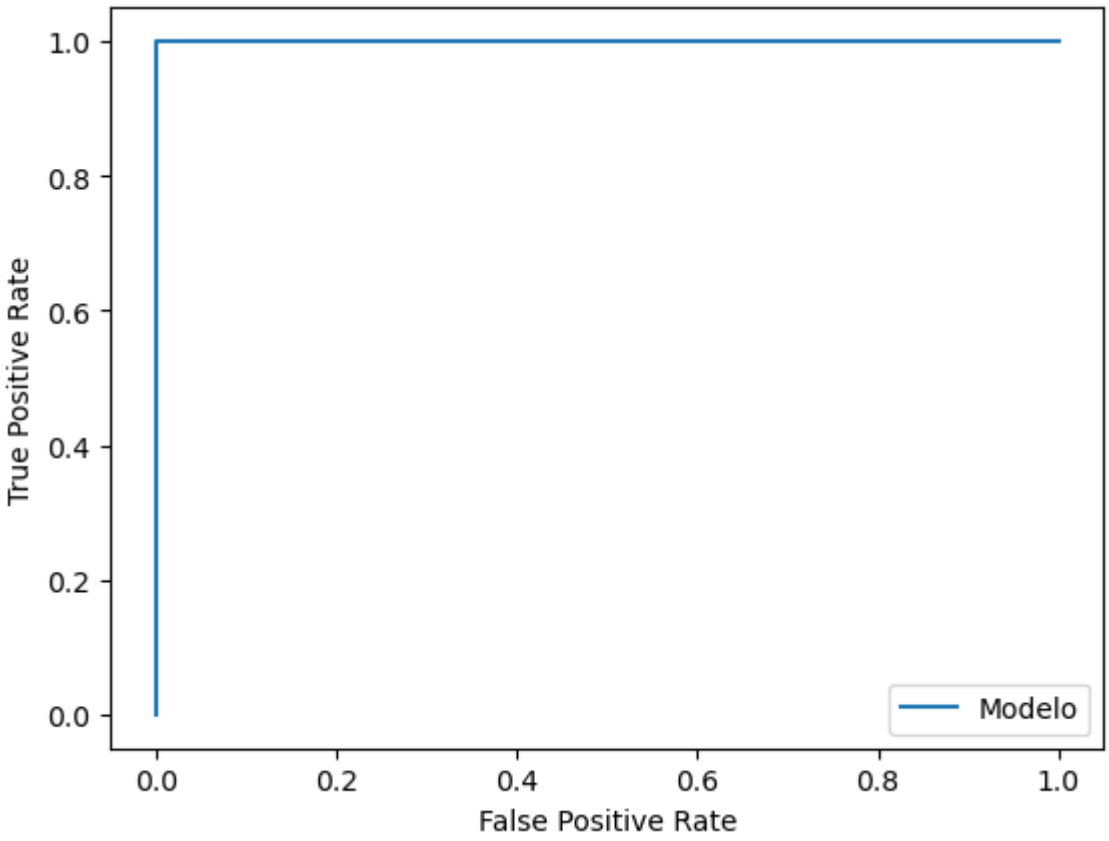
Precision-Recall

In [20]:
```
PrecisionRecallDisplay.from_estimator(modelo, x, y);
```



ROC Chart

In [24]:
```
y_score = modelo.best_estimator_.decision_function(x)
fpr, tpr, _ = roc_curve(y, y_score, pos_label=modelo.best_estimator_.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.legend(labels = ['Modelo']);
```



CREAR PIPELINE DE ENTRENAMIENTO Y EJECUCIÓN

INSTANCIAR EL MODELO

In [25]:
```
modelo = RidgeClassifier()
```

CREAR Y GUARDAR EL PIPE FINAL DE ENTRENAMIENTO

In [26]:
```
# Crear pipe de entrenamiento

pipe_entrenamiento = make_pipeline(modelo)
```

In [28]:
```
# Guardar pipe de entrenamiento
nombre_pipe_entrenamiento = 'pipe_entrenamiento_v1.pickle'
ruta_pipe_entrenamiento = '../../04_Modelos/' + nombre_pipe_entrenamiento

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(pipe_entrenamiento, file)
```

ENTRENAR Y GUARDAR EL PIPE DE EJECUCIÓN

In [29]:
```
#Entrenar pipe de entrenamiento

pipe_ejecucion = pipe_entrenamiento.fit(x,y)
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\linear_model\_ridge.py:1182: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [30]:
```
nombre_pipe_ejecucion = 'pipe_ejecucion_v1.pickle'
ruta_pipe_ejecucion = '../../04_Modelos/' + nombre_pipe_ejecucion

with open (ruta_pipe_ejecucion, mode= 'wb') as file:
    cloudpickle.dump(pipe_ejecucion, file)
```

6_ 2 - MODELO DE EJECUCIÓN CON LOS DATOS DE PRUEBA CON EL **RIDGE CLASSIFIER**

En este notebook vamos a cargar el dataset de prueba y realizaremos la calidad de datos, separaremos predictoras y target, predeciremos sobre los datos de prueba y evaluaremos el modelo definitivo.

IMPORTAR LOS PAQUETES

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scikitplot as skplt

#transformación de datos
from sklearn.preprocessing import OrdinalEncoder

#modelo

from sklearn.linear_model import RidgeClassifier

#metricas de evaluación
from sklearn.metrics import PrecisionRecallDisplay, RocCurveDisplay
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

CARGAR LOS DATOS

In [3]:
```
df = pd.read_csv('../../02_Datos/02_Validacion/prueba.csv', index_col= 0)
df
```

Out[3]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
190	N	N	N	N	N	
10	P	P	P	P	A	
191	P	N	N	N	A	
168	A	A	N	N	N	
130	A	A	A	A	P	
...	...	...	...	...	...	
60	A	A	A	P	P	
37	A	N	A	P	P	
22	A	A	A	A	P	
77	A	P	A	P	A	
200	N	N	N	N	N	

62 rows × 7 columns

TRANSFORMACIÓN DE DATOS

Variables a aplicar Ordinal Encoder

In [4]:
```
var_oe = ['industrial_risk',
            'management_risk',
            'financial_flexibility',
            'credibility',
            'competitiveness',
            'operating_risk']
```

Orden y categoría de las variables

In [5]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [6]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [7]:
```
df_oe = oe.fit_transform(df[var_oe])
```

In [8]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
x = pd.DataFrame(df_oe, columns= nombre_oe)
x
```

Out[8]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	0.0	0.0	0
1	2.0	2.0	2.0	2.0	1
2	2.0	0.0	0.0	0.0	1
3	1.0	1.0	0.0	0.0	0
4	1.0	1.0	1.0	1.0	2
...	...	...	...	...	
57	1.0	1.0	1.0	2.0	2
58	1.0	0.0	1.0	2.0	2
59	1.0	1.0	1.0	1.0	2
60	1.0	2.0	1.0	2.0	1
61	0.0	0.0	0.0	0.0	0

62 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [9]:
```
y = df.target.replace({'B':1 , 'NB': 0}).reset_index().copy()
y.drop(columns= 'index', inplace= True)
y
```

Out[9]:

	target
0	1
1	0
2	1
3	1
4	0
...	...
57	0
58	0
59	0
60	0
61	1

62 rows × 1 columns

MODELIZAR CON EL PIPE DE EJECUCIÓN

CARGAMOS EL PIPE DE EJECUCIÓN

In [10]:
```
modelo = pd.read_pickle('../../04_Modelos/pipe_ejecucion_v1.pickle')
```

PREDECIR Y EVALUAR CON LOS DATOS DE PRUEBA

PREDECIR SOBRE LOS DATOS

In [11]:
```
pred = modelo.predict(x)
#pred_proba = modelo.predict_proba(x)[:,1]
```

EVALUAR SOBRE LOS DATOS

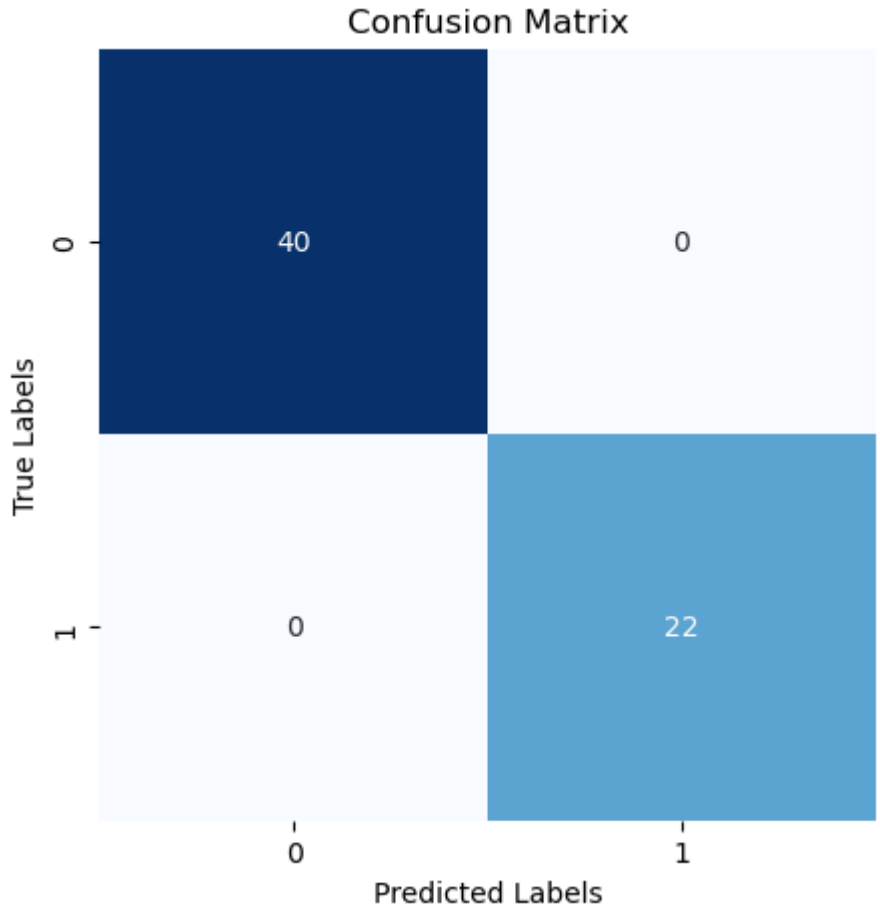
In [12]:
```
#v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

#print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

```
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        22

    accuracy                           1.00        62
   macro avg       1.00      1.00      1.00        62
weighted avg       1.00      1.00      1.00        62
```

REPORTING DEL MODELO

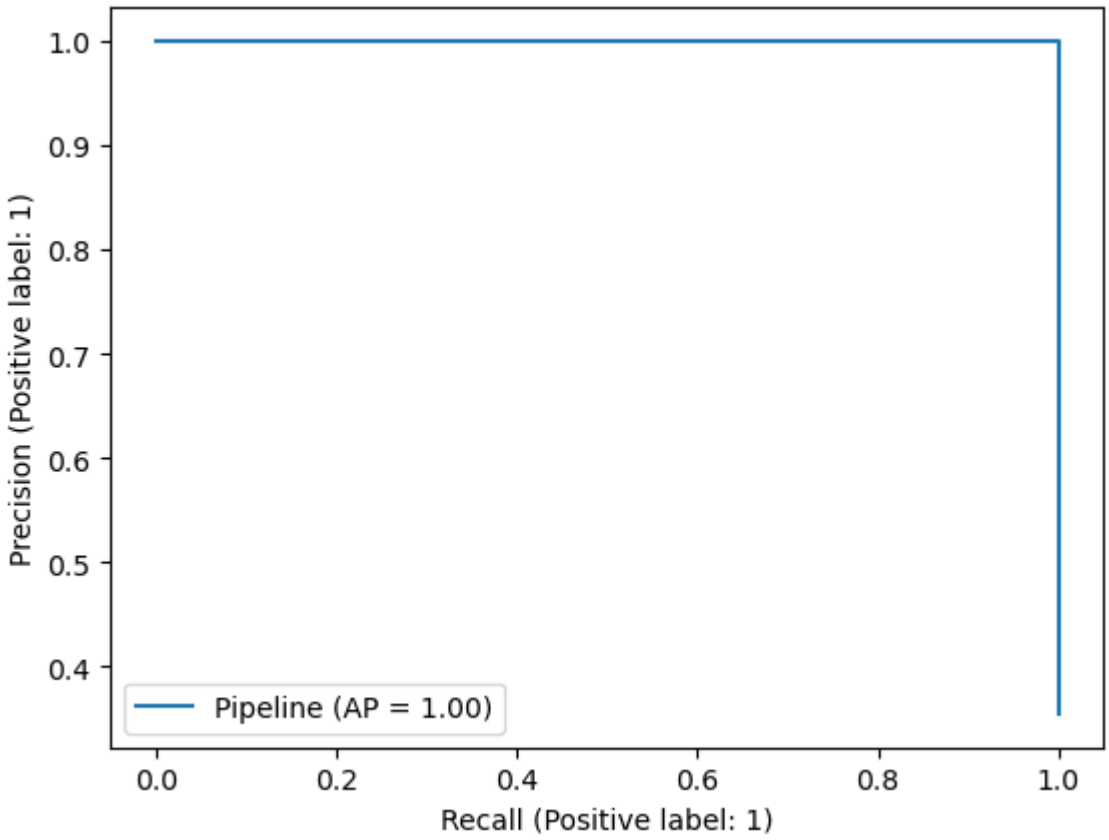
Matrix de Confusión MultiClass

In [13]:
```python
# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)

# Configurar etiquetas y título del gráfico
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



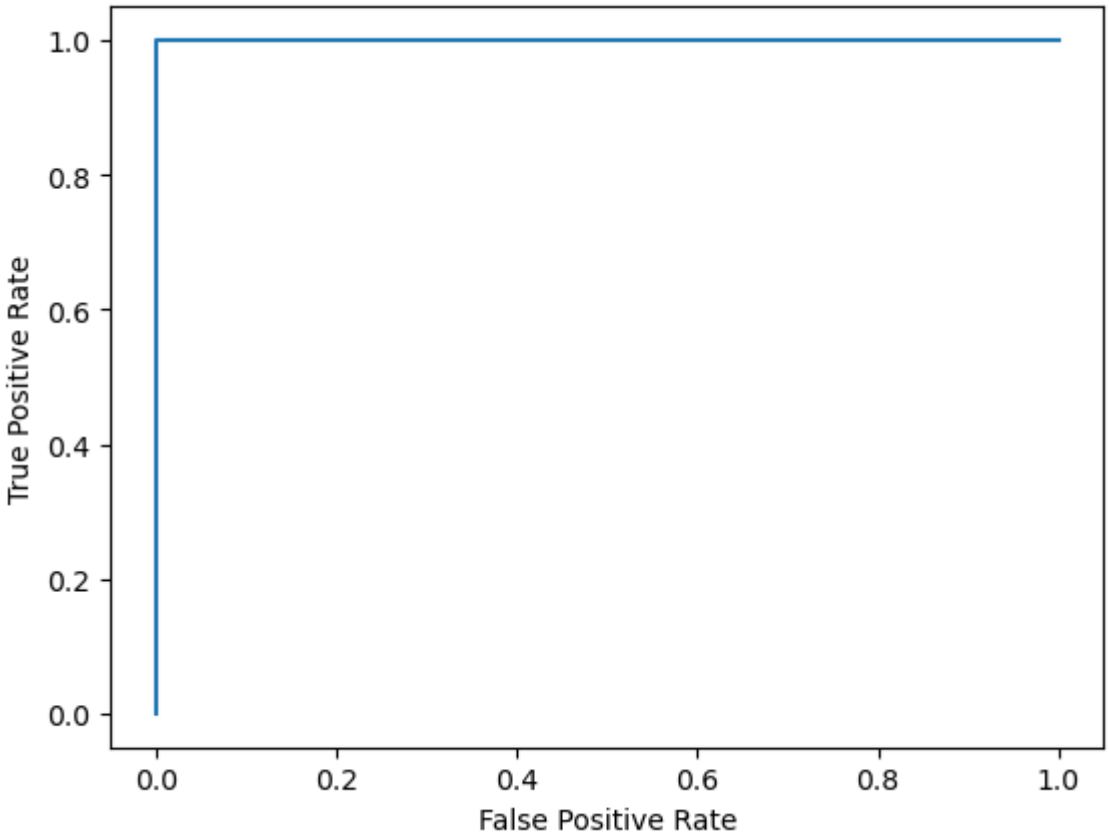
Precision-Recall

In [14]:
```
PrecisionRecallDisplay.from_estimator(modelo, x, y);
```



ROC Chart

In [15]:
```
y_score = modelo.decision_function(x)
fpr, tpr, _ = roc_curve(y, y_score, pos_label=modelo.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```



CONCLUSIÓN:

El modelo ha funcionado muy bien con los datos de prueba obteniendo un 100% roc_auc sobre los [1].

Al ser un scoring de riesgo bancario muchas veces se precisa de una interpretación de los datos para comunicar el motivo al cliente. Vamos a solicitar los coeficientes y el intercepto para calcular el scoring según las variables del cliente.

COEFICIENTE Y INTERCEPTO DEL MODELO

In [28]:
```
# Obtén el estimador interno (RidgeClassifier) del pipeline
ridge_classifier = modelo.named_steps['ridgeclassifier']

# Obtén el coeficiente y intercepto
coeficientes = ridge_classifier.coef_
intercepcion = ridge_classifier.intercept_

# Imprimir los coeficientes de las características
print("Coeficientes de las características:")
for i, coef in enumerate(coeficientes[0]):
    print(f"Característica {i + 1}: {coef:.4f}")

# Imprimir el término de intercepción (sesgo)
print("Término de intercepción (sesgo):", intercepcion[0])
```

```
Coeficientes de las características:
Característica 1: -0.0438
Característica 2: -0.0531
Característica 3: -0.2422
Característica 4: -0.2066
Característica 5: -0.6489
Característica 6: -0.0787
Término de intercepción (sesgo): 0.974061742352917
```

FUNCIÓN PARA CALCULAR EL SCORING DEL CLIENTE

In [32]:
```
def de_logit_a_prob(modelo, test_x):
    test_x = np.array(test_x)
    logit = modelo.intercept_ + np.sum(modelo.coef_ * test_x, axis=1)
    proba = 1 / (1 + np.exp(-logit))
    return proba

# Obtén el estimador interno (RidgeClassifier) del pipeline
modelo_especifico = modelo.named_steps['ridgeclassifier']

# Valores manuales de un nuevo cliente
valores_manuales = [[2, 0, 0, 0, 0, 0]]  # Ejemplo de valores manuales para caracte

# Calcular las probabilidades utilizando la función de_logit_a_prob
probabilidades = de_logit_a_prob(modelo_especifico, valores_manuales)

# Imprimir las probabilidades calculadas
print("Probabilidades:", probabilidades)
```

```
Probabilidades: [0.70815425]
```

PRÓXIMOS PASOS:

También realizaremos un modelo de tipo Árbol de decisión para ver el flujo de las variables y los valores para una mejor interpretación.

7 THE BEST ESTIMATOR CON ÁRBOL DE DECISIÓN (Fx)

Queremos poder interpretar los datos del modelo mediante una formula para exportarla. Vamos a poner a competir a los siguientes modelos.

- DecisiónTreeClassifier

MODELAR ALGORITMO DE CLASIFICACIÓN

IMPORTACIÓN DE PAQUETES

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# Crear train and test
from sklearn.model_selection import train_test_split

#Modelos seleccionados por LazyPredict
from sklearn.tree import DecisionTreeClassifier

#Optimizar modelos
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import RocCurveDisplay
import scikitplot as skplt

#Crear Pipeline

from sklearn.pipeline import Pipeline

import cloudpickle

import warnings
warnings.filterwarnings("ignore")
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

In [4]:
```
df = pd.read_pickle('../../02_Datos/03_Trabajo/df_tablon_oe.pickle')
df.head()
```

Out[4]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_oe
0	0.0	0.0	1.0	1.0	1.(
1	1.0	1.0	1.0	1.0	1.(
2	2.0	2.0	2.0	2.0	2.(
3	0.0	0.0	2.0	2.0	2.(
4	1.0	1.0	2.0	2.0	2.(

SEPARAR PREDICTORAS Y TARGET

In [5]:
```
x = df.drop(columns= 'target').copy()
y = df.target.copy()
```

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [6]:
```
train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_stat
```

MODELIZAR CON ALGORITMO SIN HIPERPARÁMETROS.

MODELIZAR SOBRE EL MODELO BASE

Instanciar modelo

In [93]:
```
modelo = DecisionTreeClassifier(random_state=42)
```

Entrenar modelo

In [94]:
```
modelo.fit(train_x,train_y)
```

Out[94]:
```
DecisionTreeClassifier(random_state=42)
```

Predecir con datos de validación

In [95]:
```
pred = modelo.predict_proba(val_x)[:,1]
pred
```

Out[95]:
```
array([0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1.])
```

Evaluar con datos de validación

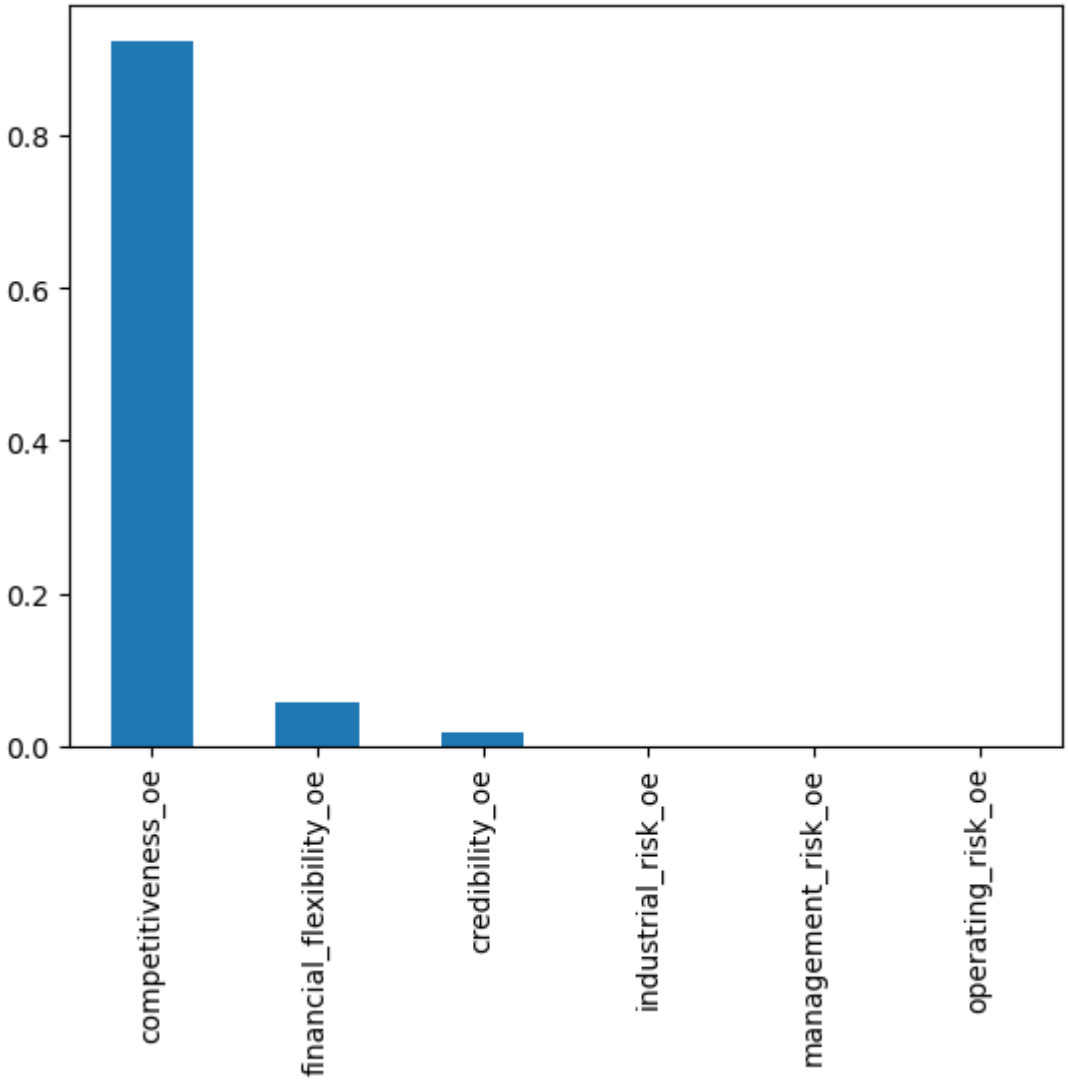
In [10]:
```
t_roc_auc = roc_auc_score(val_y, pred)
t_accuracy = accuracy_score(val_y, pred)
```

```
t_report = classification_report(val_y, pred)

print(f"Roc AUC: {t_roc_auc}")
print(f"Accuracy: {t_accuracy}")
print(f"Classification Report:\n{t_report}")
```

```
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00         7

    accuracy                           1.00        28
   macro avg       1.00      1.00      1.00        28
weighted avg       1.00      1.00      1.00        28
```

Importación de las variables

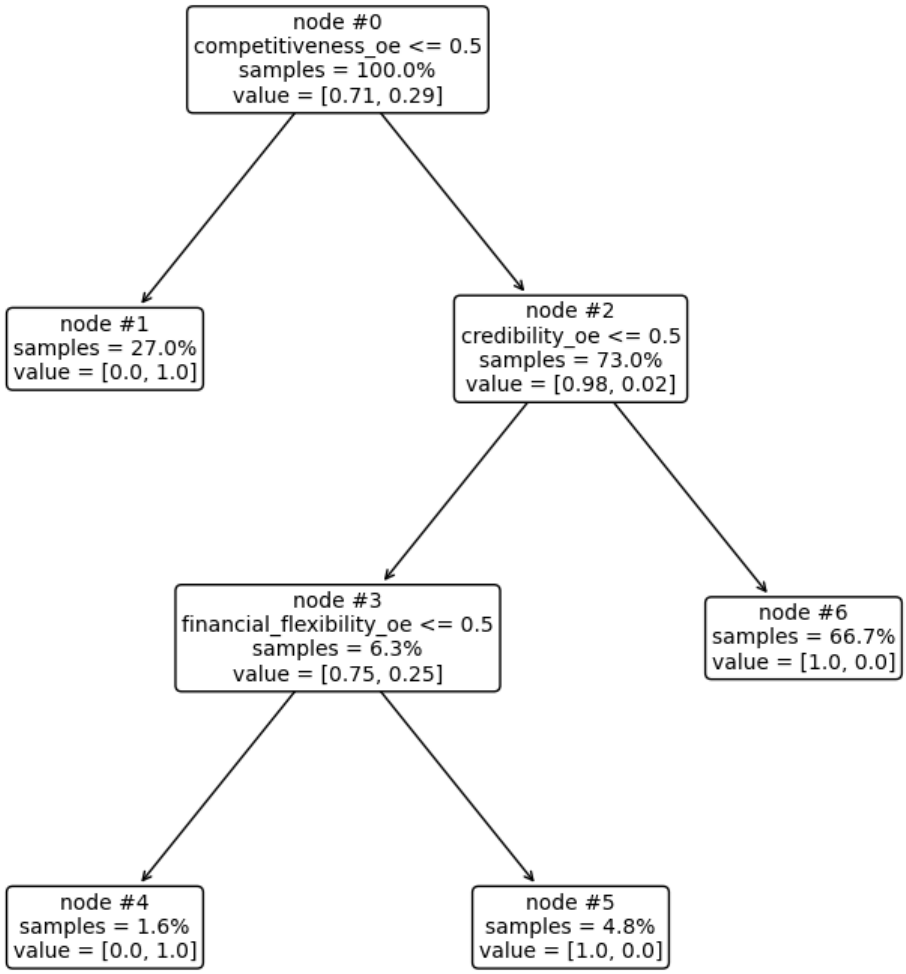
In [11]:
```
pd.Series(modelo.feature_importances_,index = val_x.columns).sort_values(ascending
```



Árbol de interpretación

In [21]:
```
from sklearn.tree import plot_tree

plt.figure(figsize = (10,10))
```

```
plot_tree(modelo,
          feature_names= val_x.columns,
          impurity = False,
          node_ids = True,
          proportion = True,
          rounded = True,
          precision = 2,
          fontsize= 10);
```



MODELIZAR CON ALGORITMO CON HIPERPARÁMETROS

INSTANCIAR EL MODELO

In [90]:
```
modelo = DecisionTreeClassifier(random_state= 42)
```

CREAR PIPE Y EL DICCIONARIO CON LOS ALGORITMOS, PARÁMETROS Y VALORES

In [24]:
```
grid = {
        'criterion': ['gini', 'entropy'],             # Criterio de selección de c
```

```
        'splitter': ['best', 'random'],              # Estrategia para dividir nod
        'max_depth': [None, 3, 4, 5, 10, 15],          # Profundidad máxima del árb
        'min_samples_split': [2, 5, 10],            # Número mínimo de muestras re
        'min_samples_leaf': [1, 2, 4, 8],           # Número mínimo de muestras re
        'max_features': ['auto', 'sqrt', 'log2'],    # Número máximo de caracterís
        'max_leaf_nodes': [None, 10, 20, 30, 40],    # Número máximo de nodos hoja
        'random_state': [42]                         # Semilla aleatoria para repr
        }
```

OPTIMIZAR LOS HIPERPARÁMETROS

In [40]:
```
grid_search = GridSearchCV(estimator= modelo,
                           param_grid= grid,
                           cv = 5,
                           scoring= 'roc_auc',
                           n_jobs= -1,
                           verbose=0)
```

In [ ]:
```
modelo_p = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')[:5]
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_d
1755	0.000202	0.000405	0.007800	0.000395	gini	
99	0.002803	0.003492	0.005267	0.003423	gini	N
1827	0.008197	0.000406	0.001599	0.003198	gini	
101	0.001902	0.003804	0.004796	0.003916	gini	N
1011	0.005598	0.003197	0.003999	0.007999	gini	

5 rows × 21 columns

In [ ]:
```
print('Best Estimator:', modelo.best_estimator_)
print('\nBest Params:', modelo.best_params_)
print('\nBest Score:', modelo.best_score_)
```

```
Best Estimator: DecisionTreeClassifier(max_features='auto', min_samples_split=10,
                       random_state=42, splitter='random')

Best Params: {'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max
_leaf_nodes': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'random_stat
e': 42, 'splitter': 'random'}

Best Score: 0.9851851851851852
```

PREDECIR SOBRE VALIDACIÓN

In [69]:
```
predv1 = modelo_p.best_estimator_.predict(val_x)
predv1_p = modelo_p.best_estimator_.predict_proba(val_x)[:,1]
```

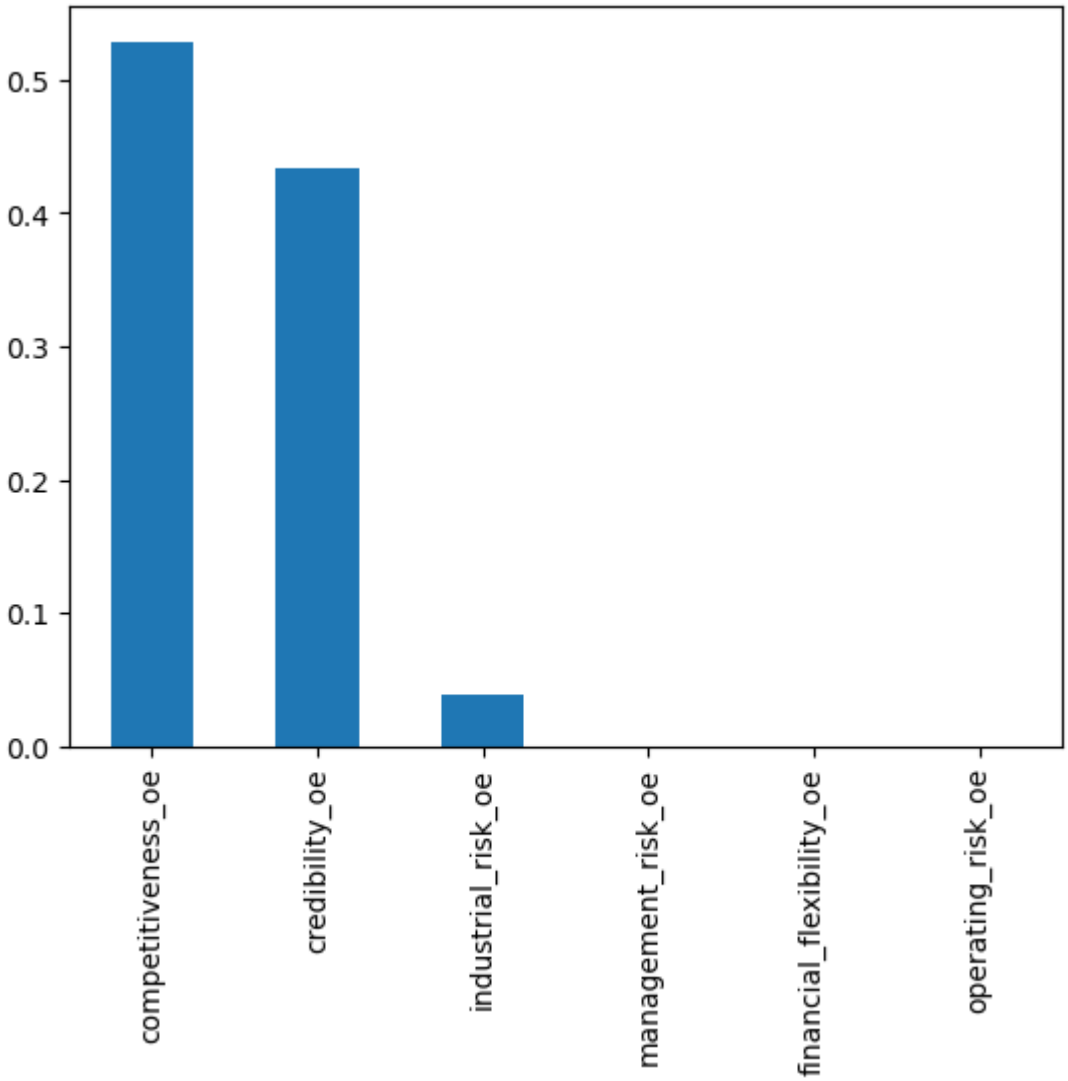
EVALUAR SOBRE LA VALIDACIÓN

In [70]:
```
v_roc_auc_proba = roc_auc_score(val_y, predv1_p)
v_roc_auc = roc_auc_score(val_y, predv1)
v_accuracy = accuracy_score(val_y, predv1)
v_report = classification_report(val_y, predv1)

print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

```
Roc AUC_proba: 1.0
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00         7

    accuracy                           1.00        28
   macro avg       1.00      1.00      1.00        28
weighted avg       1.00      1.00      1.00        28
```

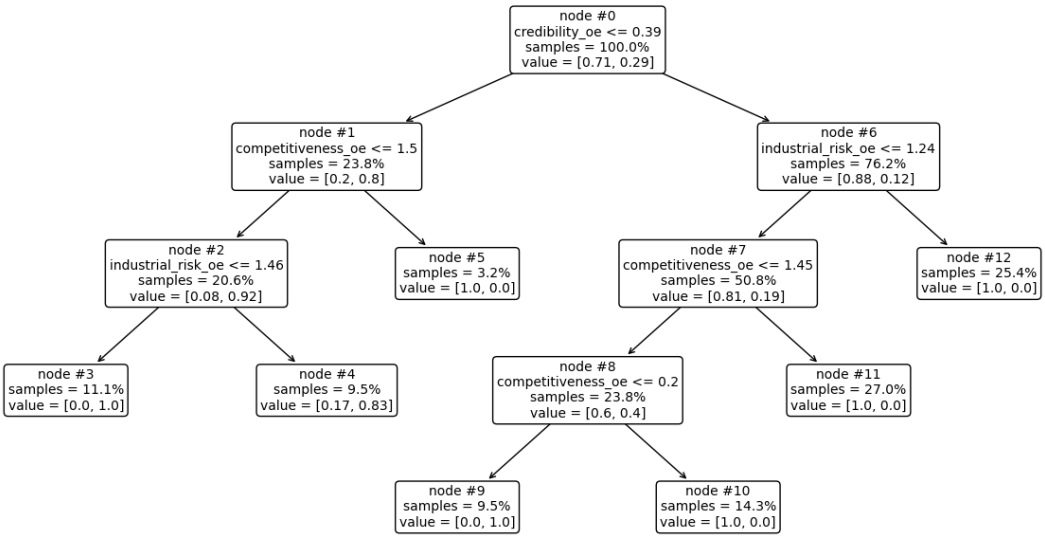
Importación de las variables

In [ ]:
```
pd.Series(modelo_p.best_estimator_.feature_importances_,index = val_x.columns).sort
```



Árbol de interpretación

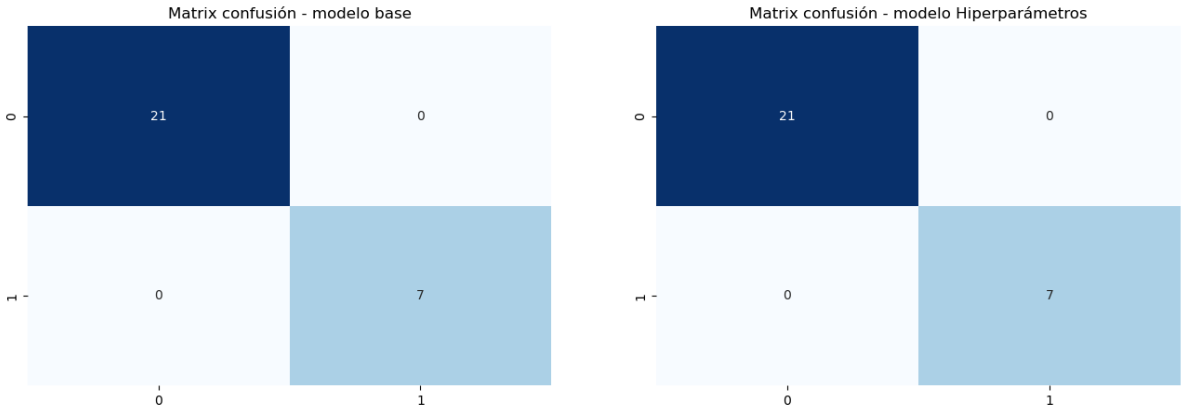
In [ ]:
```
from sklearn.tree import plot_tree

plt.figure(figsize = (16,8))

plot_tree(modelo_p.best_estimator_,
          feature_names= val_x.columns,
          impurity = False,
          node_ids = True,
          proportion = True,
          rounded = True,
          precision = 2,
          fontsize= 10);
```



REPORTING DE LOS MODELOS

Matrix de Confusión

In [86]:
```
# Crear subplots
f, ax = plt.subplots(1, 2, figsize=(16, 5))
ax = ax.flat

# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)
cmv1 = confusion_matrix(val_y,predv1)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False, ax=ax[0])
sns.heatmap(cmv1, annot=True, cmap="Blues", fmt="d", cbar=False, ax=ax[1])
ax[0].set_title('Matrix confusión - modelo base')
ax[1].set_title('Matrix confusión - modelo Hiperparámetros');
```

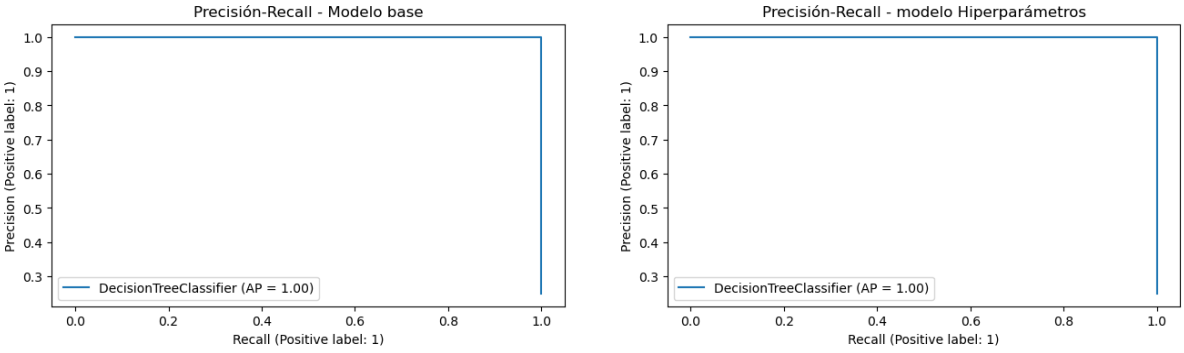



```
<Figure size 500x500 with 0 Axes>
```

Precision-Recall

In [104…
```
f, ax = plt.subplots(1, 2, figsize = (16, 4))
ax.flat

PrecisionRecallDisplay.from_estimator(modelo, val_x, val_y, ax= ax[0])
```

```
PrecisionRecallDisplay.from_estimator(modelo_p.best_estimator_, val_x, val_y, ax= a


ax[0].set_title('Precisión-Recall - Modelo base')
ax[1].set_title('Precisión-Recall - modelo Hiperparámetros');
```

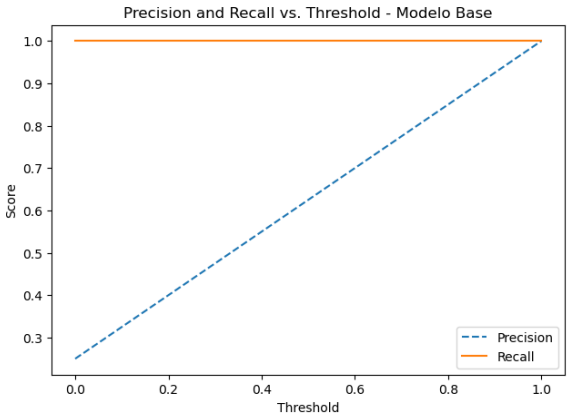


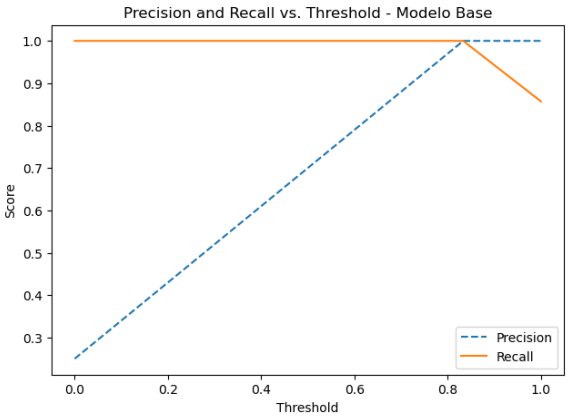
UMBRAL DE CORTE PRECISIÓN-RECALL

In [130…

```
#Función para calcular el precisión y recall de cada gráfica
def precision_recall(y, pred):
    result = []
    result = precision_recall_curve(y, pred)
    return result

# Calcula la curva de precisión y recall para diferentes umbrales de corte
umbral = precision_recall(val_y, pred)
umbralv1 = precision_recall(val_y, predv1_p)

#Creamos los rejilla de los gráficos
f, ax = plt.subplots(1,2, figsize = (16,5))
ax.flat

# Grafica la relación entre la precisión y el recall en función del umbral de corte
plt.figure(figsize=(8, 6))
ax[0].plot(umbral[2], umbral[0][:-1], label='Precision', linestyle='--')
ax[0].plot(umbral[2], umbral[1][:-1], label='Recall', linestyle='-')
ax[0].set_xlabel('Threshold')
ax[0].set_ylabel('Score')
ax[0].legend()
ax[0].set_title('Precision and Recall vs. Threshold - Modelo Base')

# Grafica la relación entre la precisión y el recall en función del umbral de corte
plt.figure(figsize=(8, 6))
ax[1].plot(umbralv1[2], umbralv1[0][:-1], label='Precision', linestyle='--')
ax[1].plot(umbralv1[2], umbralv1[1][:-1], label='Recall', linestyle='-')
ax[1].set_xlabel('Threshold')
ax[1].set_ylabel('Score')
ax[1].legend()
ax[1].set_title('Precision and Recall vs. Threshold - Modelo Base');

# Encuentra el umbral que maximiza F1-score (o ajusta según tus necesidades)

precision, recall, thresholds = umbral
f1_scores = 2 * (precision * recall) / (precision + recall)
best_threshold = thresholds[np.argmax(f1_scores)]
print("El mejor best_threshold:", best_threshold)

precision, recall, thresholds = umbralv1
f1_scores = 2 * (precision * recall) / (precision + recall)
best_threshold = thresholds[np.argmax(f1_scores)]
print("El mejor best_threshold:", best_threshold);
```

```
El mejor best_threshold: 1.0
El mejor best_threshold: 0.8333333333333334
```



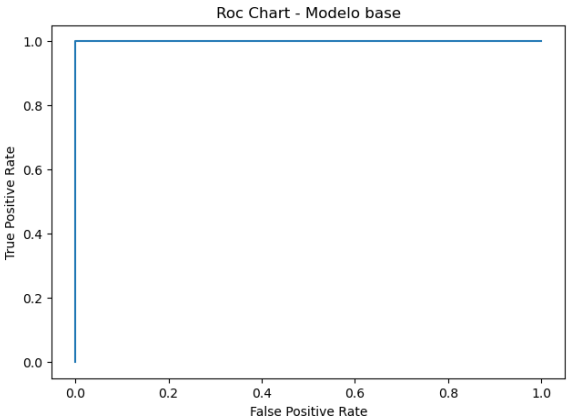


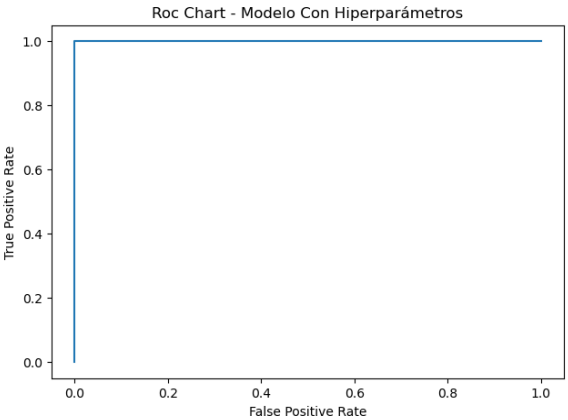

```
<Figure size 800x600 with 0 Axes>
<Figure size 800x600 with 0 Axes>
```

ROC Chart

In [140…

```
#Creamos la rejilla del gráfico
f, ax = plt.subplots(1, 2, figsize = (16,5))
ax.flat


#Calculamos Roc Chart del modelo base
y_score = modelo.predict_proba(val_x)[:,1]
fpr, tpr, _ = roc_curve(val_y, y_score, pos_label=modelo.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot(ax= ax[0])
ax[0].set_title('Roc Chart - Modelo base')

#Calculamos Roc Chart del modelo con hiperparámetros
y_score = modelo_p.best_estimator_.predict_proba(val_x)[:,1]
fpr, tpr, _ = roc_curve(val_y, y_score, pos_label=modelo_p.best_estimator_.classes_
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot(ax= ax[1])
ax[1].set_title('Roc Chart - Modelo Con Hiperparámetros');
```





GUARDAR BEST_ESTIMATOR, PARÁMETROS Y RESULTADOS DEL TEST Y LA VALIDACIÓN

Guardar el estimador con el modelo base

In [145…

```
modelo
```

Out[145]:

```
DecisionTreeClassifier(random_state=42)
```

In [147…

```
version_estimator = '_7_v0'
m_best_estimator = str(modelo)
m_best_estimator = m_best_estimator.split('(')[0]
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator

ruta_pipe_entrenamiento = '../../04_Modelos/Best_estimator/' + nombre_best_estimato

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(modelo, file)
```

Guardar el estimador con el modelo con hiperparámetros

In [148…

```
version_estimator = '_7_v1'
m_best_estimator = str(modelo_p.best_estimator_)
m_best_estimator = m_best_estimator.split('(')[0]
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator

ruta_pipe_entrenamiento = '../../04_Modelos/Best_estimator/' + nombre_best_estimato

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(modelo_p, file)
```

CONCLUSIÓN: Los dos modelos tienen muy buenos resultados con los datos de validación. El modelo v2 tiene una diversificación más amplia y podría tender a sobre ajustarse. El modelo base puede tender a generalizar los resultados. Pero basa el resultado solo en una variable.

PRÓXIMOS PASOS:

Realizaremos el modelo definitivo con este modelo. Si detectamos que el modelo está sobre ajustado, haremos un nuevo modelo con hiperparámetros para evitar el sobre ajuste y evaluaremos nuevamente el modelo.

7_01 - MODELO CLASIFICACIÓN CON ÁRBOL DE DECISIÓN CON HIPERPARÁMETROS PARA EVITAR EL SOBRE AJUSTE

En este notebook vamos a crear los pipeline de entrenamiento y ejecución.

IMPORTACIÓN DE PAQUETES

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#transformación de datos
from sklearn.preprocessing import OrdinalEncoder

#modelo

from sklearn.tree import DecisionTreeClassifier

#Métricas de evaluación
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import RocCurveDisplay
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTAR LOS DATOS

In [2]:
```
df = pd.read_csv('../../02_Datos/03_Trabajo/tablon_analitico.csv', index_col= 0)
df
```

Out[2]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
1	N	N	A	A	A	
2	A	A	A	A	A	
3	P	P	P	P	P	
4	N	N	P	P	P	
5	A	A	P	P	P	
...	...	...	...	...	...	
178	A	P	N	A	N	
179	N	N	N	P	N	
182	N	N	N	P	N	
215	N	A	P	A	N	
234	N	A	N	N	N	

91 rows × 7 columns

SEPARAMOS PREDICTORAS Y TARGET

In [3]:
```
x = df.drop(columns= 'target').copy()
y = df.target.copy().reset_index()
```

TRANSFORMACIÓN DE VARIABLES

Variables a aplicar Ordinal Encoder

In [4]:
```
var_oe = ['industrial_risk',
          'management_risk',
          'financial_flexibility',
          'credibility',
          'competitiveness',
          'operating_risk']
```

Orden y categoría de las variables

In [5]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [6]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [7]:
```
df_oe = oe.fit_transform(x[var_oe])
```

In [8]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
df_oe = pd.DataFrame(df_oe, columns= nombre_oe)
df_oe
```

Out[8]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [9]:
```
y = y.replace({'B':1 , 'NB': 0})
y.drop(columns= 'index', inplace= True)
y
```

Out[9]:

	target
0	0
1	0
2	0
3	0
4	0
...	...
86	1
87	1
88	1
89	1
90	1

91 rows × 1 columns

AGRUPAR PREDICTORAS Y TARGET EN UN DATASET

In [10]:
```
df_tablon = pd.concat([df_oe, y], axis= 1, ignore_index= False)
df_tablon
```

Out[10]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 7 columns

SEPARAMOS PREDICTORAS Y TARGET

In [11]:
```
x = df_tablon.drop(columns= 'target').copy()
y = df_tablon.target.copy().reset_index()
y.drop(columns= 'index', inplace= True)
```

MODELIZAR CON DECISIONTREECLASSIFIER - MODELO BASE

CARGAR EL MEJOR MODELO CON EL ALGORITMO, PARÁMETROS Y VALORES

In [12]:
```
modelo = pd.read_pickle('../../04_Modelos/Best_Estimator/DecisionTreeClassifier_7_v
```

In [13]:
```
modelo.best_estimator_
```

Out[13]:
```
DecisionTreeClassifier(max_features='auto', min_samples_split=10,
                       random_state=42, splitter='random')
```

In [14]:
```
modelo.best_estimator_.get_params
```

Out[14]:
```
<bound method BaseEstimator.get_params of DecisionTreeClassifier(max_features='aut
o', min_samples_split=10,
                       random_state=42, splitter='random')>
```

PREDECIR SOBRE LA VALIDACIÓN

In [15]:
```
pred = modelo.predict(x)
pred_proba = modelo.predict_proba(x)[:,1]
```

EVALUAR SOBRE LA VALIDACIÓN

In [16]:
```
#v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

#print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

```
Roc AUC: 0.9924242424242424
Accuracy: 0.989010989010989
Classification Report:              precision    recall  f1-score   support

           0       1.00      0.98      0.99        66
           1       0.96      1.00      0.98        25

    accuracy                           0.99        91
   macro avg       0.98      0.99      0.99        91
weighted avg       0.99      0.99      0.99        91
```

In [25]:
```
#v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

#print(f"Roc AUC_proba: {v_roc_auc_proba}")
```

```
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

```
Roc AUC: 0.9924242424242424
Accuracy: 0.989010989010989
Classification Report:              precision    recall  f1-score   support

           0       1.00      0.98      0.99        66
           1       0.96      1.00      0.98        25

    accuracy                           0.99        91
   macro avg       0.98      0.99      0.99        91
weighted avg       0.99      0.99      0.99        91
```

CREAR PIPELINE DE ENTRENAMIENTO Y EJECUCIÓN

INSTANCIAR EL MODELO

In [17]:
```
modelo = DecisionTreeClassifier(max_features='auto',
                                min_samples_split=10,
                                random_state=42,
                                splitter='random')
```

CREAR Y GUARDAR EL PIPE FINAL DE ENTRENAMIENTO

In [20]:
```
# Crear pipe de entrenamiento

pipe_entrenamiento = make_pipeline(modelo)
```

In [21]:
```
# Guardar pipe de entrenamient_o
nombre_pipe_entrenamiento = 'pipe_entrenamiento_7_v1.pickle'
ruta_pipe_entrenamiento = '../../04_Modelos/' + nombre_pipe_entrenamiento

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(pipe_entrenamiento, file)
```

ENTRENAR Y GUARDAR EL PIPE DE EJECUCIÓN

In [22]:
```
#Entrenar pipe de entrenamiento

pipe_ejecucion = pipe_entrenamiento.fit(x,y)
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\tree\_classes.py:269: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'`.
  warnings.warn(
```

In [23]:
```
nombre_pipe_ejecucion = 'pipe_ejecucion_7_v1.pickle'
ruta_pipe_ejecucion = '../../04_Modelos/' + nombre_pipe_ejecucion

with open (ruta_pipe_ejecucion, mode= 'wb') as file:
    cloudpickle.dump(pipe_ejecucion, file)
```

7_01 - MODELO CLASIFICACIÓN CON ÁRBOL DE DECISIÓN

En este notebook vamos a crear los pipeline de entrenamiento y ejecución.

IMPORTACIÓN DE PAQUETES

In [18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#transformación de datos
from sklearn.preprocessing import OrdinalEncoder

#modelo

from sklearn.tree import DecisionTreeClassifier

#Métricas de evaluación
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import RocCurveDisplay
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTAR LOS DATOS

In [3]:
```
df = pd.read_csv('../../02_Datos/03_Trabajo/tablon_analitico.csv', index_col= 0)
df
```

Out[3]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
1	N	N	A	A	A	
2	A	A	A	A	A	
3	P	P	P	P	P	
4	N	N	P	P	P	
5	A	A	P	P	P	
...	...	...	...	...	...	
178	A	P	N	A	N	
179	N	N	N	P	N	
182	N	N	N	P	N	
215	N	A	P	A	N	
234	N	A	N	N	N	

91 rows × 7 columns

SEPARAMOS PREDICTORAS Y TARGET

In [4]:
```
x = df.drop(columns= 'target').copy()
y = df.target.copy().reset_index()
```

TRANSFORMACIÓN DE VARIABLES

Variables a aplicar Ordinal Encoder

In [5]:
```
var_oe = ['industrial_risk',
          'management_risk',
          'financial_flexibility',
          'credibility',
          'competitiveness',
          'operating_risk']
```

Orden y categoría de las variables

In [6]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [7]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [8]:
```
df_oe = oe.fit_transform(x[var_oe])
```

In [9]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
df_oe = pd.DataFrame(df_oe, columns= nombre_oe)
df_oe
```

Out[9]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [10]:
```
y = y.replace({'B':1 , 'NB': 0})
y.drop(columns= 'index', inplace= True)
y
```

Out[10]:

	target
0	0
1	0
2	0
3	0
4	0
...	...
86	1
87	1
88	1
89	1
90	1

91 rows × 1 columns

AGRUPAR PREDICTORAS Y TARGET EN UN DATASET

In [11]:
```
df_tablon = pd.concat([df_oe, y], axis= 1, ignore_index= False)
df_tablon
```

Out[11]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	1.0	1.0	1
1	1.0	1.0	1.0	1.0	1
2	2.0	2.0	2.0	2.0	2
3	0.0	0.0	2.0	2.0	2
4	1.0	1.0	2.0	2.0	2
...	...	...	...	...	
86	1.0	2.0	0.0	1.0	0
87	0.0	0.0	0.0	2.0	0
88	0.0	0.0	0.0	2.0	0
89	0.0	1.0	2.0	1.0	0
90	0.0	1.0	0.0	0.0	0

91 rows × 7 columns

SEPARAMOS PREDICTORAS Y TARGET

In [12]:
```
x = df_tablon.drop(columns= 'target').copy()
y = df_tablon.target.copy().reset_index()
y.drop(columns= 'index', inplace= True)
```

MODELIZAR CON DECISIONTREECLASSIFIER - MODELO BASE

CARGAR EL MEJOR MODELO CON EL ALGORITMO, PARÁMETROS Y VALORES

In [13]:
```
modelo = pd.read_pickle('../../04_Modelos/Best_Estimator/DecisionTreeClassifier_7_v
```

In [14]:
```
modelo
```

Out[14]:
```
DecisionTreeClassifier(random_state=42)
```

PREDECIR SOBRE LA VALIDACIÓN

In [15]:
```
pred = modelo.predict(x)
pred_proba = modelo.predict_proba(x)[:,1]
```

EVALUAR SOBRE LA VALIDACIÓN

In [16]:
```
#v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

#print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

```
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:              precision    recall  f1-score   support

           0       1.00      1.00      1.00        66
           1       1.00      1.00      1.00        25

    accuracy                           1.00        91
   macro avg       1.00      1.00      1.00        91
weighted avg       1.00      1.00      1.00        91
```

CREAR PIPELINE DE ENTRENAMIENTO Y EJECUCIÓN

INSTANCIAR EL MODELO

In [19]:
```
modelo = DecisionTreeClassifier(random_state=42)
```

CREAR Y GUARDAR EL PIPE FINAL DE ENTRENAMIENTO

In [20]:
```
# Crear pipe de entrenamiento

pipe_entrenamiento = make_pipeline(modelo)
```

In [21]:
```
# Guardar pipe de entrenamiento
nombre_pipe_entrenamiento = 'pipe_entrenamiento_7_v0.pickle'
ruta_pipe_entrenamiento = '../../04_Modelos/' + nombre_pipe_entrenamiento

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(pipe_entrenamiento, file)
```

ENTRENAR Y GUARDAR EL PIPE DE EJECUCIÓN

In [22]:
```
#Entrenar pipe de entrenamiento

pipe_ejecucion = pipe_entrenamiento.fit(x,y)
```

In [23]:
```
nombre_pipe_ejecucion = 'pipe_ejecucion_7_v0.pickle'
ruta_pipe_ejecucion = '../../04_Modelos/' + nombre_pipe_ejecucion

with open (ruta_pipe_ejecucion, mode= 'wb') as file:
    cloudpickle.dump(pipe_ejecucion, file)
```

7_ 2 - MODELO DE EJECUCIÓN CON LOS DATOS DE PRUEBA CON ÁRBOL DE DECISIÓN CON HIPERPARÁMETROS

En este notebook vamos a cargar el dataset de prueba y realizaremos la calidad de datos, separaremos predictoras y target, predeciremos sobre los datos de prueba y evaluaremos el modelo definitivo.

IMPORTAR LOS PAQUETES

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scikitplot as skplt

#transformación de datos
from sklearn.preprocessing import OrdinalEncoder

#modelo

from sklearn.tree import DecisionTreeClassifier

#metricas de evaluación
from sklearn.metrics import PrecisionRecallDisplay, RocCurveDisplay
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

CARGAR LOS DATOS

In [4]:
```
df = pd.read_csv('../../02_Datos/02_Validacion/prueba.csv', index_col= 0)
df
```

Out[4]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
190	N	N	N	N	N	
10	P	P	P	P	A	
191	P	N	N	N	A	
168	A	A	N	N	N	
130	A	A	A	A	P	
...	...	...	...	...	...	
60	A	A	A	P	P	
37	A	N	A	P	P	
22	A	A	A	A	P	
77	A	P	A	P	A	
200	N	N	N	N	N	

62 rows × 7 columns

TRANSFORMACIÓN DE DATOS

Variables a aplicar Ordinal Encoder

In [5]:
```
var_oe = ['industrial_risk',
            'management_risk',
            'financial_flexibility',
            'credibility',
            'competitiveness',
            'operating_risk']
```

Orden y categoría de las variables

In [6]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [7]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [8]:
```
df_oe = oe.fit_transform(df[var_oe])
```

In [9]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
x = pd.DataFrame(df_oe, columns= nombre_oe)
x
```

Out[9]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	0.0	0.0	0
1	2.0	2.0	2.0	2.0	1
2	2.0	0.0	0.0	0.0	1
3	1.0	1.0	0.0	0.0	0
4	1.0	1.0	1.0	1.0	2
...	...	...	...	...	
57	1.0	1.0	1.0	2.0	2
58	1.0	0.0	1.0	2.0	2
59	1.0	1.0	1.0	1.0	2
60	1.0	2.0	1.0	2.0	1
61	0.0	0.0	0.0	0.0	0

62 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [10]:
```
y = df.target.replace({'B':1 , 'NB': 0}).reset_index().copy()
y.drop(columns= 'index', inplace= True)
y
```

Out[10]:

	target
0	1
1	0
2	1
3	1
4	0
...	...
57	0
58	0
59	0
60	0
61	1

62 rows × 1 columns

MODELIZAR CON EL PIPE DE EJECUCIÓN

CARGAMOS EL PIPE DE EJECUCIÓN

In [11]:
```
modelo = pd.read_pickle('../../04_Modelos/pipe_ejecucion_7_v1.pickle')
```

In [18]:
```
modelo.get_params
```

Out[18]:
```
<bound method Pipeline.get_params of Pipeline(steps=[('decisiontreeclassifier',
                 DecisionTreeClassifier(max_features='auto',
                                        min_samples_split=10, random_state=42,
                                        splitter='random'))])>
```

PREDECIR Y EVALUAR CON LOS DATOS DE PRUEBA

PREDECIR SOBRE LOS DATOS

In [12]:
```
pred = modelo.predict(x)
pred_proba = modelo.predict_proba(x)[:,1]
```

EVALUAR SOBRE LOS DATOS

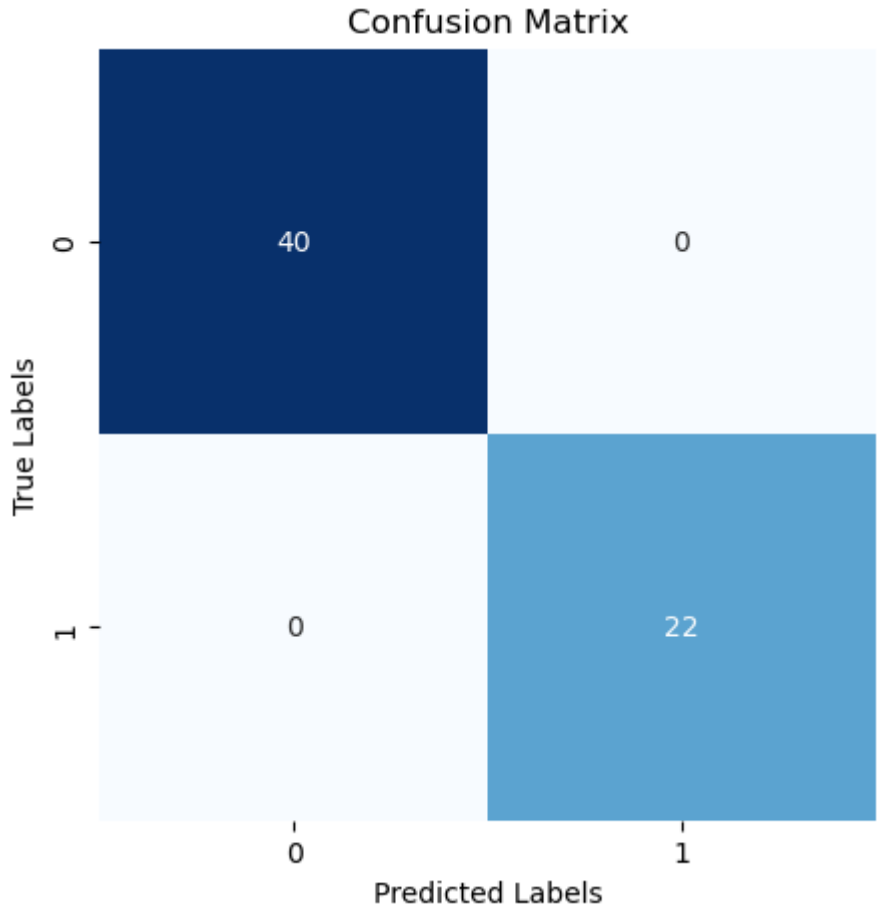
In [13]:
```
v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

#print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

```
Roc AUC: 1.0
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        22

    accuracy                           1.00        62
   macro avg       1.00      1.00      1.00        62
weighted avg       1.00      1.00      1.00        62
```

REPORTING DEL MODELO

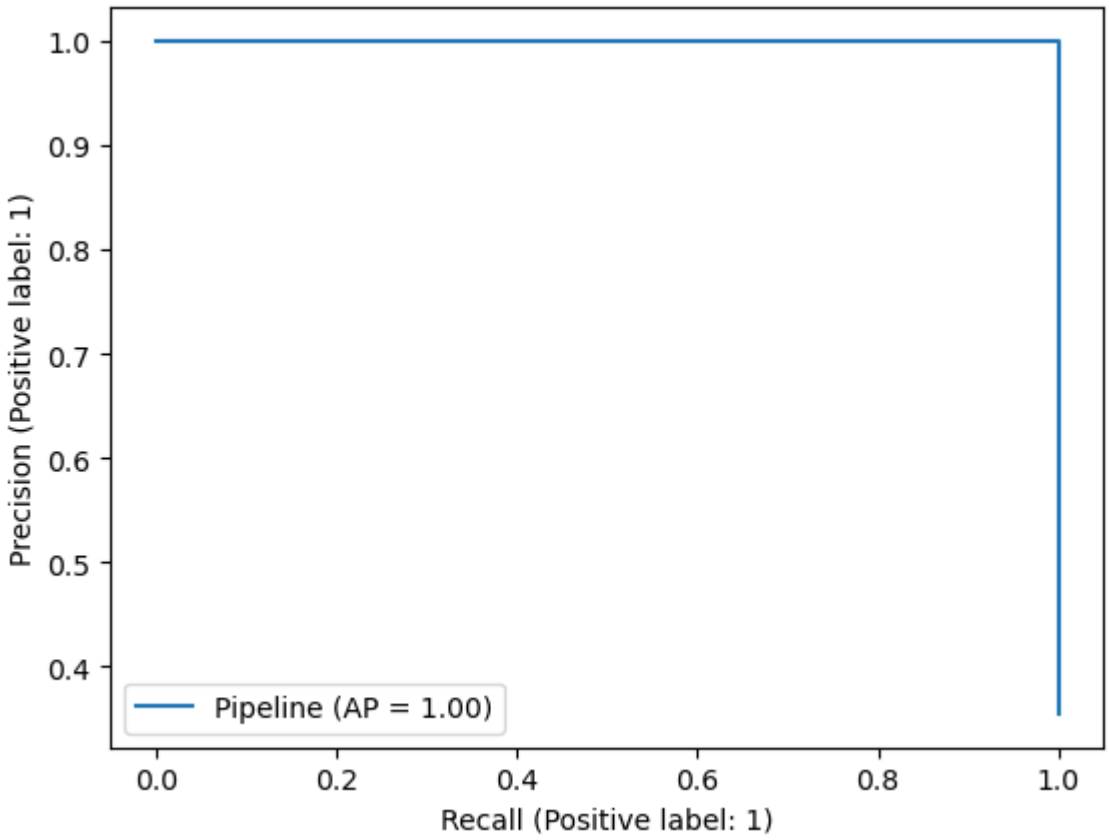
Matrix de Confusión MultiClass

In [14]:
```python
# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)

# Configurar etiquetas y título del gráfico
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



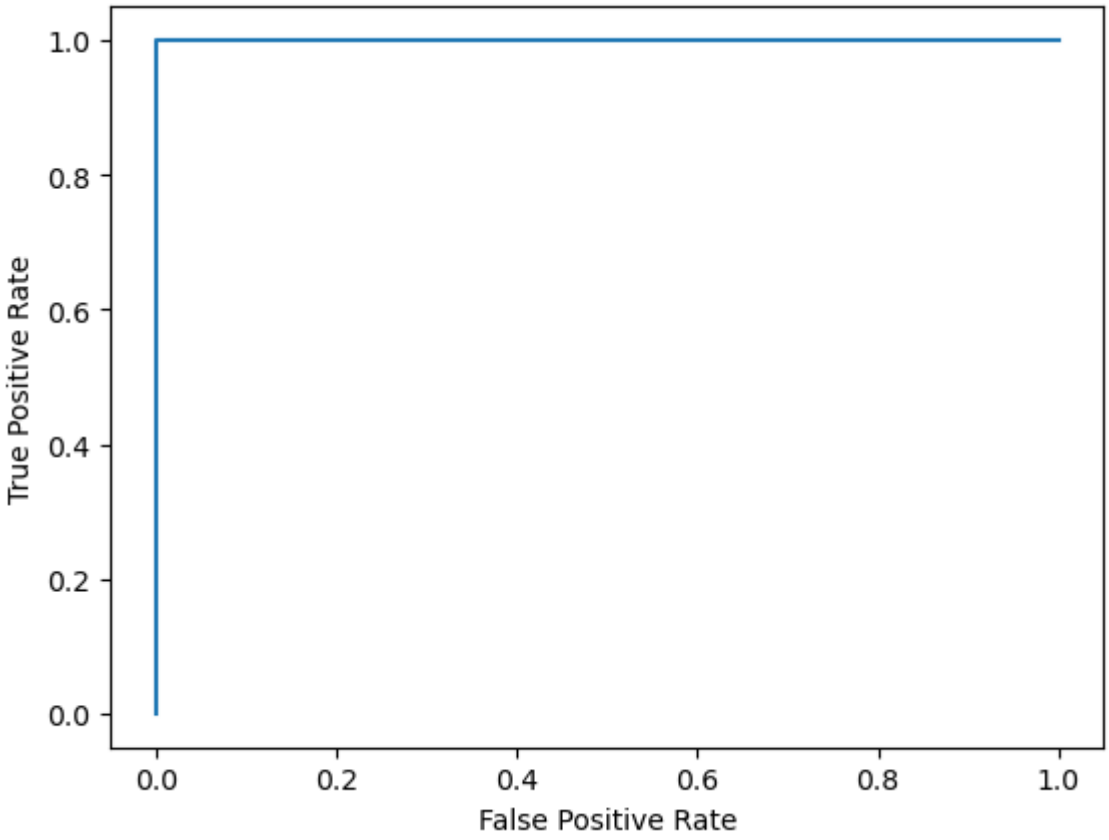
Precision-Recall

In [15]:
```
PrecisionRecallDisplay.from_estimator(modelo, x, y);
```



ROC Chart

In [16]:
```
y_score = modelo.predict_proba(x)[:,1]
fpr, tpr, _ = roc_curve(y, y_score, pos_label=modelo.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```



CONCLUSIÓN:

El modelo ha funcionado muy bien con los datos de prueba obteniendo un 100% roc_auc sobre los [1].

Al ser un scoring de riesgo bancario muchas veces se precisa de una interpretación de los datos para comunicar el motivo al cliente. Vamos a solicitar los coeficientes y el intercepto para calcular el scoring según las variables del cliente.

7_ 2 - MODELO DE EJECUCIÓN CON LOS DATOS DE PRUEBA CON ÁRBOL DE DECISIÓN

En este notebook vamos a cargar el dataset de prueba y realizaremos la calidad de datos, separaremos predictoras y target, predeciremos sobre los datos de prueba y evaluaremos el modelo definitivo.

IMPORTAR LOS PAQUETES

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scikitplot as skplt

#transformación de datos
from sklearn.preprocessing import OrdinalEncoder

#modelo

from sklearn.tree import DecisionTreeClassifier

#metricas de evaluación
from sklearn.metrics import PrecisionRecallDisplay, RocCurveDisplay
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

CARGAR LOS DATOS

In [3]:
```
df = pd.read_csv('../../02_Datos/02_Validacion/prueba.csv', index_col= 0)
df
```

Out[3]:

	industrial_risk	management_risk	financial_flexibility	credibility	competitiveness	operating_r
190	N	N	N	N	N	
10	P	P	P	P	A	
191	P	N	N	N	A	
168	A	A	N	N	N	
130	A	A	A	A	P	
...	...	...	...	...	...	
60	A	A	A	P	P	
37	A	N	A	P	P	
22	A	A	A	A	P	
77	A	P	A	P	A	
200	N	N	N	N	N	

62 rows × 7 columns

TRANSFORMACIÓN DE DATOS

Variables a aplicar Ordinal Encoder

In [4]:
```
var_oe = ['industrial_risk',
            'management_risk',
            'financial_flexibility',
            'credibility',
            'competitiveness',
            'operating_risk']
```

Orden y categoría de las variables

In [5]:
```
orden_industrial = ['N','A','P']
orden_management = ['N','A','P']
orden_financial = ['N','A','P']
orden_credibility = ['N','A','P']
orden_competitiveness = ['N','A','P']
orden_operating = ['N','A','P']

categorias = [orden_industrial,
              orden_management,
              orden_financial,
              orden_credibility,
              orden_competitiveness,
              orden_operating]
```

Instanciar Ordinal Encoder

In [6]:
```
oe = OrdinalEncoder(categories= categorias,
                    handle_unknown= 'use_encoded_value',
                    unknown_value= -99)
```

Entrenar y aplicar

In [7]:
```
df_oe = oe.fit_transform(df[var_oe])
```

In [8]:
```
nombre_oe = [variable + '_oe' for variable in var_oe]
x = pd.DataFrame(df_oe, columns= nombre_oe)
x
```

Out[8]:

	industrial_risk_oe	management_risk_oe	financial_flexibility_oe	credibility_oe	competitiveness_c
0	0.0	0.0	0.0	0.0	0
1	2.0	2.0	2.0	2.0	1
2	2.0	0.0	0.0	0.0	1
3	1.0	1.0	0.0	0.0	0
4	1.0	1.0	1.0	1.0	2
...	...	...	...	...	
57	1.0	1.0	1.0	2.0	2
58	1.0	0.0	1.0	2.0	2
59	1.0	1.0	1.0	1.0	2
60	1.0	2.0	1.0	2.0	1
61	0.0	0.0	0.0	0.0	0

62 rows × 6 columns

APLICAR TRANSFORMACIÓN BINARIA A LA TARGET

In [9]:
```
y = df.target.replace({'B':1 , 'NB': 0}).reset_index().copy()
y.drop(columns= 'index', inplace= True)
y
```

Out[9]:

	target
0	1
1	0
2	1
3	1
4	0
...	...
57	0
58	0
59	0
60	0
61	1

62 rows × 1 columns

MODELIZAR CON EL PIPE DE EJECUCIÓN

CARGAMOS EL PIPE DE EJECUCIÓN

In [10]:
```
modelo = pd.read_pickle('../../04_Modelos/pipe_ejecucion_7_v0.pickle')
```

In [18]:
```
modelo.get_params
```

Out[18]:
```
<bound method Pipeline.get_params of Pipeline(steps=[('decisiontreeclassifier',
                 DecisionTreeClassifier(random_state=42))])>
```

PREDECIR Y EVALUAR CON LOS DATOS DE PRUEBA

PREDECIR SOBRE LOS DATOS

In [11]:
```
pred = modelo.predict(x)
pred_proba = modelo.predict_proba(x)[:,1]
```

EVALUAR SOBRE LOS DATOS

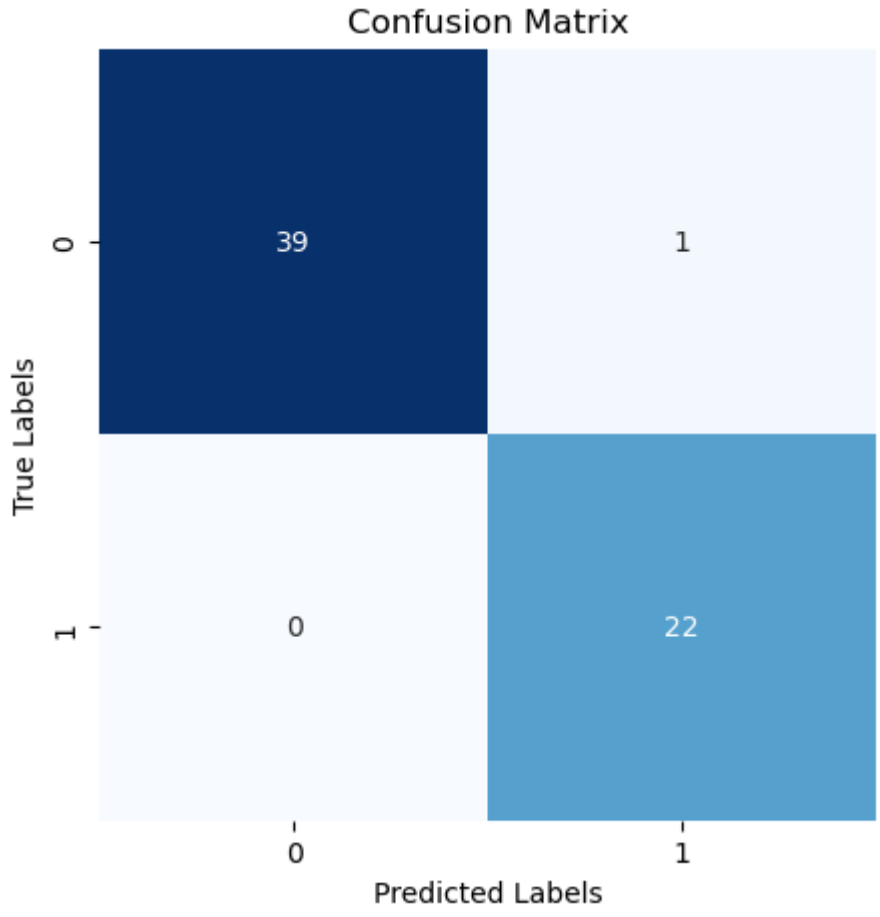
In [12]:
```
v_roc_auc_proba = roc_auc_score(y, pred_proba)
v_roc_auc = roc_auc_score(y, pred)
v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

#print(f"Roc AUC_proba: {v_roc_auc_proba}")
print(f"Roc AUC: {v_roc_auc}")
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

```
Roc AUC: 0.9874999999999999
Accuracy: 0.9838709677419355
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        40
           1       0.96      1.00      0.98        22

    accuracy                           0.98        62
   macro avg       0.98      0.99      0.98        62
weighted avg       0.98      0.98      0.98        62
```

REPORTING DEL MODELO

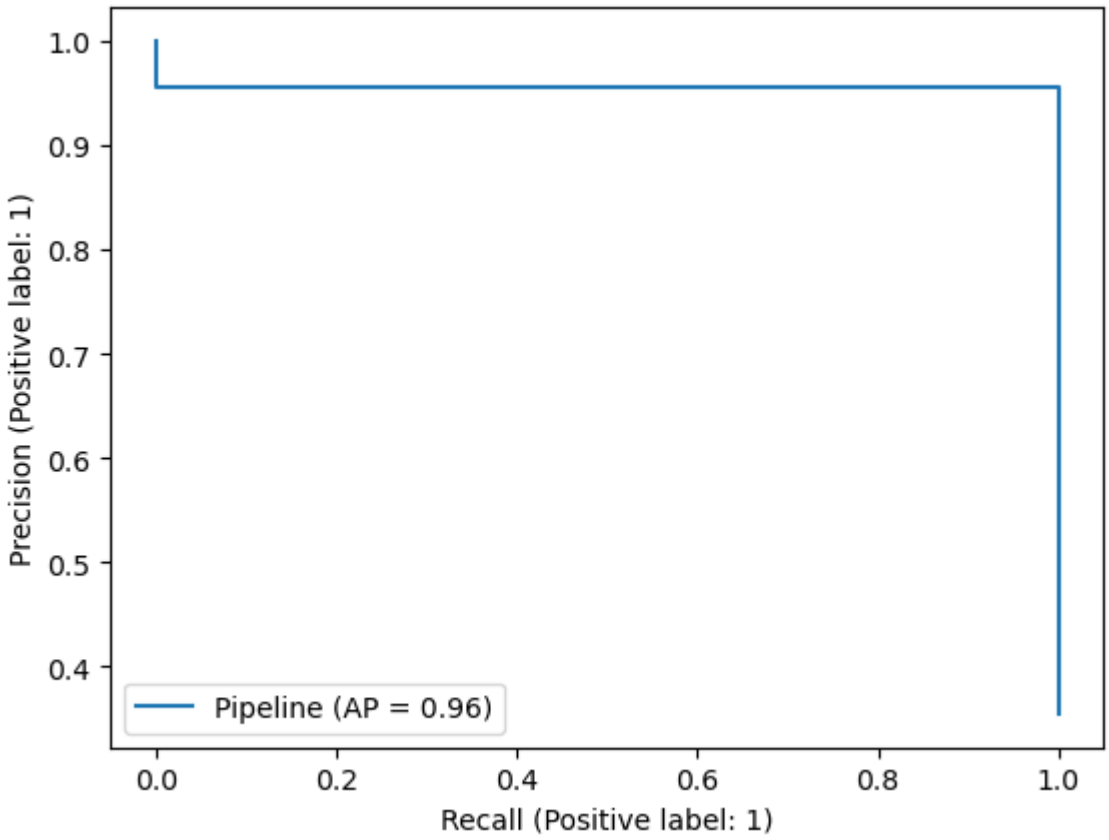
Matrix de Confusión MultiClass

In [13]:
```python
# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)


# Crear un mapa de calor de la matriz de confusión
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False)

# Configurar etiquetas y título del gráfico
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



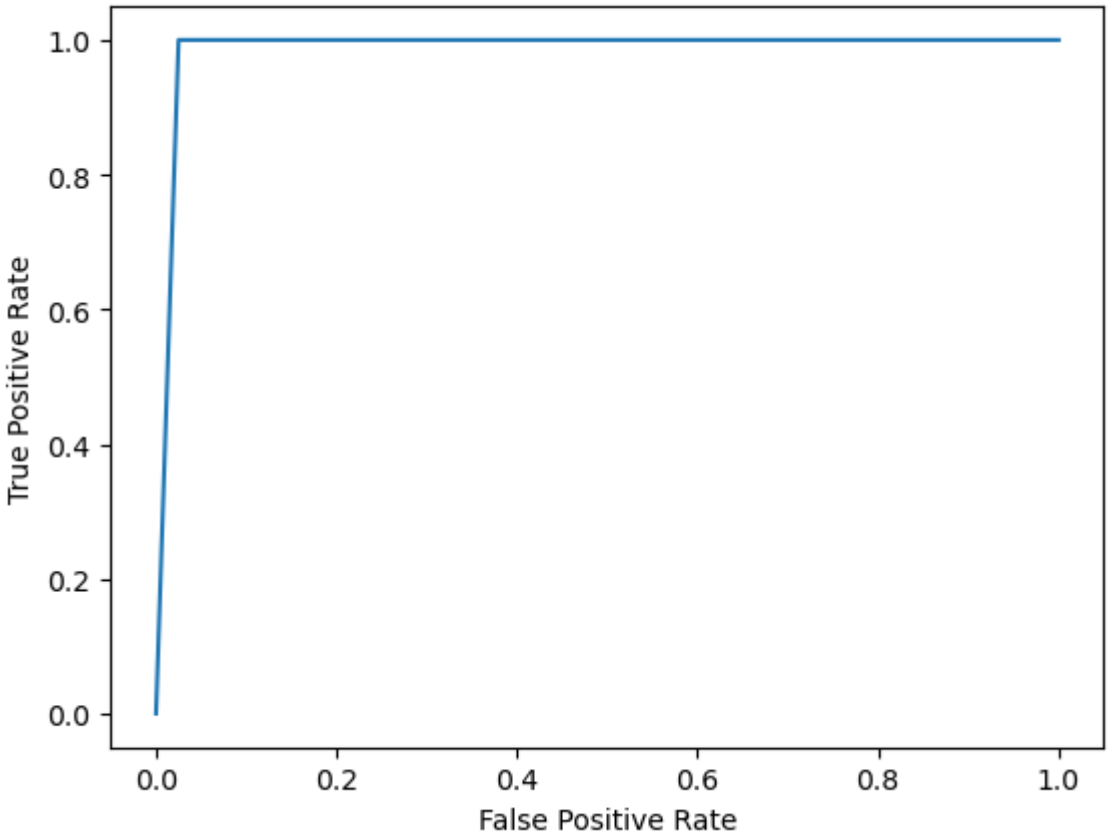
Precision-Recall

In [14]:
```
PrecisionRecallDisplay.from_estimator(modelo, x, y);
```



ROC Chart

In [17]:
```
y_score = modelo.predict_proba(x)[:,1]
fpr, tpr, _ = roc_curve(y, y_score, pos_label=modelo.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```



CONCLUSIÓN:

El modelo ha funcionado muy bien con los datos de prueba obteniendo un 100% roc_auc sobre los [1] y sin tener falsos positivos. El modelo ha funcionado ha dado mejor resultados con los datos de prueba que con los de validación por lo que entendemos que es un buen modelo. Además podemos interpretar los resultados con el gráfico del árbol.