PREDICT SEMINAL QUALITY OF AN INDIVISUAL

DISEÑO DEL PROYECTO

Este conjunto de datos de nivel principiante tiene 100 filas y 10 columnas. El conjunto de datos incluye muestras de semen de 100 voluntarios, analizados según los criterios de la OMS de 2010.

Este conjunto de datos se puede utilizar para determinar si es posible llegar a un diagnóstico sin un enfoque de laboratorio, que incluye pruebas costosas, a veces incómodas para los pacientes. La concentración espermática está relacionada con datos sociodemográficos, factores ambientales, estado de salud y hábitos de vida. Estos atributos se pueden tomar fácilmente usando un cuestionario.

OBJETIVO

Predecir la calidad del semen..

Este conjunto de datos se recomienda para aprender y practicar sus habilidades en técnicas de modelado de clasificación .

ENTIDADES Y DATOS

Posición de la columna	Nombre de atributo	Definición	Tipo de datos	Ejemplo	% relaciones nulas
1	Estación	Estación en la que se realizó el análisis (-1: invierno, -0,33: primavera, 0,33: verano, 1: otoño)	Cuantitativo	1, -1, -0,33	0
2	Edad	Edad en el momento del análisis. La edad está entre 18 y 36 y se escala de 0 a 1	Cuantitativo	0,64, 0,78, 1	0
3	Enfermedades Infantiles	Enfermedades infantiles, es decir, varicela, sarampión, paperas, polio (0: no, 1: sí)	Cuantitativo	1, 0	0
4	Accidente o trauma grave	Accidente o traumatismo grave (0: no, 1: sí)	Cuantitativo	1, 0	0
5	Intervención quirúrgica	Intervención quirúrgica (0: no, 1: sí)	Cuantitativo	1, 0	0
6	Fiebre alta en el último año	Fiebre alta en el último año (-1: hace menos de 3 meses, 0: hace más de 3 meses, 1: sin fiebre)	Cuantitativo	0, 1, -1	0
7	Frecuencia de consumo de alcohol	Frecuencia de consumo de alcohol en 5 categorías escaladas de 0 a 1. Las siguientes son las categorías en orden: 1) varias veces al día, 2) todos los días, 3) varias veces a la semana, 4) una vez a la semana, 5) casi nunca o nunca	Cuantitativo	0.2, 0.6, 1	0
8	El hábito de fumar	Tabaquismo (-1: nunca, 0: ocasional, 1: diario)	Cuantitativo	0, 1, -1	0
9	Número de horas que pasa sentado al día	Número de horas que pasa sentado al día. Entre 0 y 16, escalado de 0 a 1	Cuantitativo	0,32, 0,83, 1	0
10	Producción	Salida: Resultado del Diagnóstico (N: Normal, O: Alterado)	Cualitativo	NO	0

1- SET UP

IMPORTACIÓN DE PAQUETES

In [11]:
```
import pandas as pd
import numpy as np
```

CREAR LOS DATASETS INICIALES

CARGAR LOS DATOS

In [5]:
```
df = pd.read_csv('../../02_Datos/01_Originales/fertility_Diagnosis.txt', names= ['e
df.head()
```

Out[5]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
3	-0.33	0.75	0	1	1	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	

In [7]:
```
df.shape
```

Out[7]:
```
(100, 10)
```

EXTRAER Y RESERVAR EL DATASET DE PRUEBA

In [6]:
```
prueba = df.sample(frac = 0.3)
print(prueba.shape)
```

```
(30, 10)
```

In [8]:
```
# Guardamos el dataset de prueba

prueba.to_csv('../../02_Datos/02_Validacion/prueba.csv')
```

EXTRAER Y RESERVAR EL DATASET DE TRABAJO

In [18]:
```
trabajo = df[~df.index.isin(prueba.index)]
print(trabajo.shape)
```

```
(70, 10)
```

In [19]:
```
# Guardamos el dataset de trabajo
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

import seaborn as sns
```

IMPORTAR LOS DATOS

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

CORRECCIÓN DE NOMBRES DE VARIABLES

No se tiene que realizar corrección de variables porque se han incluido manualmente en el set up.

VISIÓN GENERAL

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

Out[4]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

TIPO DE DATOS

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

Out[5]:
```
estacion          float64
edad              float64
e_infantil          int64
acc_grave           int64
int_quirurgica      int64
fiebre_ult_any      int64
frec_alcohol      float64
fumar               int64
hr_sentado        float64
produccion         object
dtype: object
```

In [6]:
```
df.produccion.value_counts()
```

Out[6]:
```
N    61
O     9
Name: produccion, dtype: int64
```

CONCLUSIÓN: Los datos están bien estructurados y solo se tendría que transformar la target (produccion) a numérica

In [7]:
```
# Cambio los valores manualmente
df['produccion'] = df.produccion.replace({'N': 0, 'O':1 })
df.head()
```

Out[7]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

BALANCELO DE LA TARGET

In [8]:
```
df.produccion.value_counts(normalize= True) * 100
```

Out[8]:
```
0    87.142857
1    12.857143
Name: produccion, dtype: float64
```

Veremos cómo predice el dataset. Si vemos que no predice bien los casos alterados (1) tendremos que aumentar los casos mediante el balanceo del dataset.

VALORES ÚNICOS

In [9]:
```
df.nunique()
```

Out[9]:
```
estacion           4
edad              17
e_infantil         2
acc_grave          2
int_quirurgica     2
fiebre_ult_any     3
frec_alcohol       4
fumar              3
hr_sentado        11
produccion         2
dtype: int64
```

No hay valores únicos en el dataset

VALORES DUPLICADOS

In [10]:
```
df.duplicated().sum()
```

Out[10]:
```
0
```

No hay valores duplicados

VALORES NULOS

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

Out[11]:
```
estacion          0
edad              0
e_infantil        0
acc_grave         0
int_quirurgica    0
fiebre_ult_any    0
frec_alcohol      0
fumar             0
hr_sentado        0
produccion        0
dtype: int64
```

No hay valores nulos en el dataset

GESTIÓN DE NUMÉRICAS

Todas las variables del dataset son numéricas

ESTADÍSTICOS Y GRÁFICOS DE LAS VARIABLES NUMÉRICAS

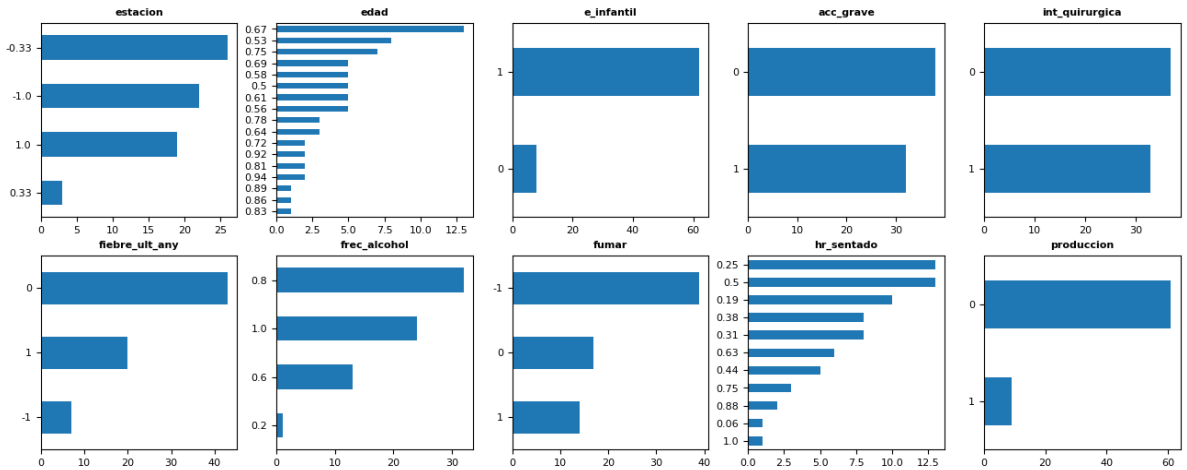
In [12]:
```
def estadisticos_cont(df_cat):
    #Calculamos describe
    estadisticos = df_cat.describe().T
    return(estadisticos)

estadisticos_cont(df)
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
estacion	70.0	-0.151286	0.785336	-1.00	-1.000	-0.33	1.00	1.00
edad	70.0	0.664571	0.116042	0.50	0.565	0.67	0.75	0.94
e_infantil	70.0	0.885714	0.320455	0.00	1.000	1.00	1.00	1.00
acc_grave	70.0	0.457143	0.501757	0.00	0.000	0.00	1.00	1.00
int_quirurgica	70.0	0.471429	0.502787	0.00	0.000	0.00	1.00	1.00
fiebre_ult_any	70.0	0.185714	0.596921	-1.00	0.000	0.00	1.00	1.00
frec_alcohol	70.0	0.822857	0.161668	0.20	0.800	0.80	1.00	1.00
fumar	70.0	-0.357143	0.799197	-1.00	-1.000	-1.00	0.00	1.00
hr_sentado	70.0	0.403143	0.192621	0.06	0.250	0.38	0.50	1.00
produccion	70.0	0.128571	0.337142	0.00	0.000	0.00	0.00	1.00

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] / 5)

    #Definimos el gráfico
    f, ax = plt.subplots(nrows = filas, ncols = 5, figsize = (16, filas * 3))

    #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)
```



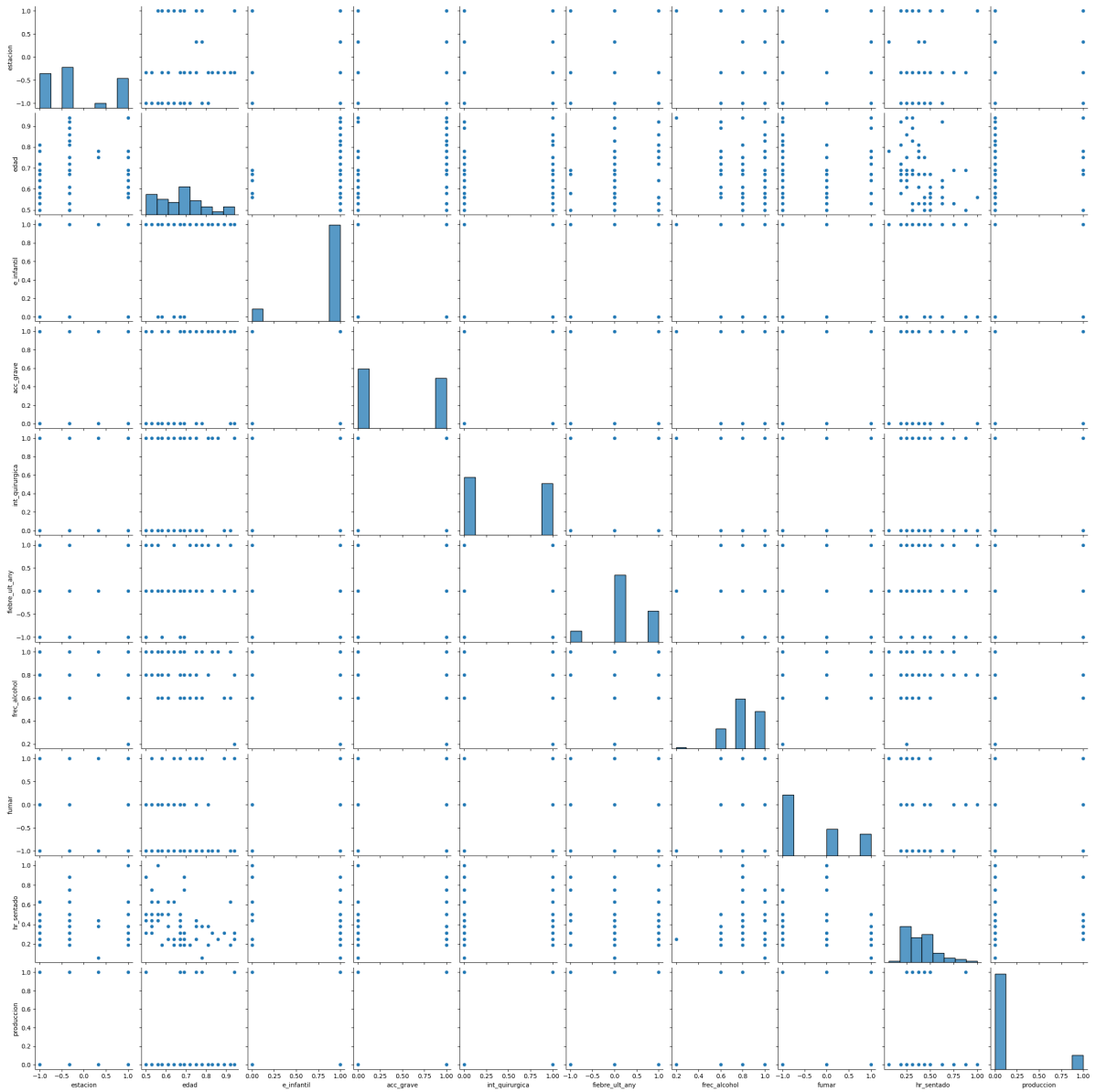
GESTIÓN DE ATÍPICOS

No vamos a gestionar atípicos

EDA

In [14]:
```
sns.pairplot(df);
```

Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x1c6a89c00d0>
```



In [15]:
```
df.corr()
```

Out[15]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcoh
estacion	1.000000	0.178577	-0.146864	-0.114341	-0.047626	-0.176012	-0.17007
edad	0.178577	1.000000	0.057124	0.289660	0.220863	0.102641	-0.28993
e_infantil	-0.146864	0.057124	1.000000	0.149366	-0.110510	0.112565	0.05115
acc_grave	-0.114341	0.289660	0.149366	1.000000	0.109972	0.051153	-0.20214
int_quirurgica	-0.047626	0.220863	-0.110510	0.109972	1.000000	-0.247654	-0.06316
fiebre_ult_any	-0.176012	0.102641	0.112565	0.051153	-0.247654	1.000000	0.01544
frec_alcohol	-0.170075	-0.289938	0.051153	-0.202144	-0.063168	0.015447	1.00000
fumar	0.027660	0.038175	0.121261	0.051630	0.028339	-0.101988	-0.16024
hr_sentado	-0.072354	-0.402714	-0.200712	-0.034575	-0.187612	-0.058089	0.14751
produccion	0.202614	0.132937	0.003833	-0.095464	0.064734	-0.120368	-0.21423

No se detecta correlaciones fuertes entre variables

GUARDAR DATASETS TRAS CALIDAD DE DATOS

In [71]:
```
df.to_pickle('../../02_Datos/03_Trabajo/df_tablon.pickle')
```

03 - PRESELECCIÓN MEJOR MODELO SEGÚN LAZYPREDICT

IMPORTACIÓN DE PAQUETES

In [15]:
```
import numpy as np
import pandas as pd

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

CARGAR DATOS

In [16]:
```
df = pd.read_pickle('../../02_Datos/03_Trabajo/df_tablon.pickle')
df.head()
```

Out[16]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.80	0	
1	-0.33	0.94	1	0	1	0	0.80	1	
2	-0.33	0.50	1	0	0	0	1.00	-1	
4	-0.33	0.67	1	1	0	0	0.80	-1	
5	-0.33	0.67	1	0	1	0	0.80	0	

SEPARAR PREDICTORAS Y TARGET

In [17]:
```
x = df.drop(columns='produccion').copy()
y = df.produccion.copy()
```

DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [18]:
```
train_x, val_x, train_y, val_y = train_test_split(x, y, train_size= 0.3, random_sta
```

MODELIZAR SEGÚN LAZYPREDICT

In [19]:
```
reg = LazyClassifier(verbose=0)
models, predictions = reg.fit(train_x, val_x, train_y, val_y)
models
```

```
  3%|▌          | 1/29 [00:00<00:03,  9.06it/s]c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The least populated class in y has only 3 members, which is less than n_splits=5.
  warnings.warn(
 59%|█████▊     | 17/29 [00:00<00:00, 34.16it/s]c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\discriminant_analysis.py:926: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
100%|██████████| 29/29 [00:00<00:00, 32.04it/s]
```

```
[LightGBM] [Warning] There are no meaningful features which satisfy the provided c
onfiguration. Decreasing Dataset parameters min_data_in_bin or min_data_in_leaf an
d re-constructing Dataset might resolve this warning.
[LightGBM] [Info] Number of positive: 3, number of negative: 18
[LightGBM] [Info] Total Bins 0
[LightGBM] [Info] Number of data points in the train set: 21, number of used featu
res: 0
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.142857 -> initscore=-1.791759
[LightGBM] [Info] Start training from score -1.791759
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
```

```
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
```

```
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
```

```
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
[LightGBM] [Warning] Stopped training because there are no more leaves that meet t
he split requirements
```

Out[19]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LabelPropagation	0.86	0.63	0.63	0.85	0.01
LabelSpreading	0.86	0.63	0.63	0.85	0.01
GaussianNB	0.82	0.61	0.61	0.82	0.01
AdaBoostClassifier	0.88	0.57	0.57	0.85	0.11
ExtraTreeClassifier	0.84	0.55	0.55	0.82	0.02
ExtraTreesClassifier	0.84	0.55	0.55	0.82	0.19
DecisionTreeClassifier	0.63	0.50	0.50	0.69	0.03
Perceptron	0.76	0.50	0.50	0.77	0.01
BaggingClassifier	0.88	0.50	0.50	0.82	0.03
SVC	0.88	0.50	0.50	0.82	0.01
RandomForestClassifier	0.88	0.50	0.50	0.82	0.19
QuadraticDiscriminantAnalysis	0.88	0.50	0.50	0.82	0.01
LGBMClassifier	0.88	0.50	0.50	0.82	0.01
KNeighborsClassifier	0.88	0.50	0.50	0.82	0.03
DummyClassifier	0.88	0.50	0.50	0.82	0.02
CalibratedClassifierCV	0.88	0.50	0.50	0.82	0.02
PassiveAggressiveClassifier	0.71	0.48	0.48	0.74	0.01
LogisticRegression	0.84	0.48	0.48	0.80	0.02
RidgeClassifier	0.84	0.48	0.48	0.80	0.02
RidgeClassifierCV	0.84	0.48	0.48	0.80	0.01
XGBClassifier	0.84	0.48	0.48	0.80	0.03
BernoulliNB	0.82	0.47	0.47	0.79	0.02
LinearSVC	0.82	0.47	0.47	0.79	0.01
LinearDiscriminantAnalysis	0.80	0.45	0.45	0.78	0.02
SGDClassifier	0.78	0.44	0.44	0.77	0.02
NearestCentroid	0.63	0.43	0.43	0.69	0.02

CONCLUSIÓN:

Probaremos los siguientes modelos propuestos por LazyPredict:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LabelPropagation	0.86	0.63	0.63	0.85	0.01
LabelSpreading	0.86	0.63	0.63	0.85	0.01
GaussianNB	0.82	0.61	0.61	0.82	0.01
AdaBoostClassifier	0.88	0.57	0.57	0.85	0.11
ExtraTreeClassifier	0.84	0.55	0.55	0.82	0.02

De primeras detectamos que el ROC AUC es muy bajo por lo que realizaremos los siguientes procesos.

1. Realizar predicción con modelos base con los modelos propuestos por LazyPredict
2. Preselección de variables y predecir con modelos base (incluiremos XGBClassifier y LGBMClassifier)
3. Revisaremos modelos vs métricas y evaluaremos el modelo definitivo con el GridSearch para evitar sobreajuste.

04 - THE BEST ESTIMATOR V1

MODELAR ALGORITMO DE CLASIFICACIÓN BASE

Los modelos que vamos a poner a competir con mediante el GridSearch:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LabelPropagation	0.86	0.63	0.63	0.85	0.01
LabelSpreading	0.86	0.63	0.63	0.85	0.01
GaussianNB	0.82	0.61	0.61	0.82	0.01
AdaBoostClassifier	0.88	0.57	0.57	0.85	0.11
ExtraTreeClassifier	0.84	0.55	0.55	0.82	0.02

IMPORTACIÓN DE PAQUETES

In [67]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#Crear datasets de entrenamiento y validación
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.semi_supervised import LabelPropagation
from sklearn.semi_supervised import LabelSpreading
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier

#Optimizar modelo
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

In [68]:
```
df_tablon = pd.read_pickle('../../02_Datos/03_Trabajo/df_tablon.pickle')
df_tablon.head()
```

Out[68]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

SEPARAR PREDICTORAS Y TARGET

In [69]:
```
x = df_tablon.drop(columns= 'produccion').copy()
y = df_tablon.produccion.copy()
```

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [70]:
```
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 [71]:
```
pipe = Pipeline([('algoritmo', ExtraTreesClassifier())])

grid =  [

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

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

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

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

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

```
        ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

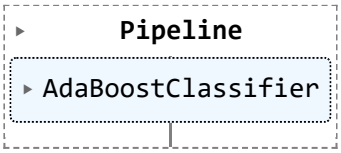
In [72]:
```
grid_search = GridSearchCV( estimator= pipe,
                            param_grid= grid,
                            cv = 5,
                            scoring= 'roc_auc',
                            verbose=0,
                            n_jobs= -1
                          )

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

Out[72]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	pa
4	0.120055	0.008775	0.014468	0.003235	AdaBoostClassifier()	{'algor AdaBoostClassi
2	0.005806	0.002979	0.004802	0.006401	LabelSpreading()	{'algor LabelSpreac
1	0.003972	0.004865	0.005890	0.003230	LabelPropagation()	{'algor LabelPropagat
3	0.222487	0.018532	0.016020	0.005070	ExtraTreesClassifier()	{'algor ExtraTreesClassi
0	0.002347	0.001617	0.002588	0.003851	GaussianNB()	{'algor Gaussiar

In [73]:
```
modelo.best_estimator_
```

Out[73]:



In [74]:
```
modelo.best_params_
```

Out[74]:
```
{'algoritmo': AdaBoostClassifier()}
```

In [75]:
```
modelo.best_score_
```

Out[75]:
```
0.7958333333333334
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [76]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [77]:
```
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

PREDECIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el Train

In [78]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [79]:
```
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        43
           1       1.00      1.00      1.00         6

    accuracy                           1.00        49
   macro avg       1.00      1.00      1.00        49
weighted avg       1.00      1.00      1.00        49
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

In [80]:
```
pred = modelo.best_estimator_.predict(val_x)
pred_proba = modelo.best_estimator_.predict_proba(val_x)[:,1]
```

Evaluar sobre la validación

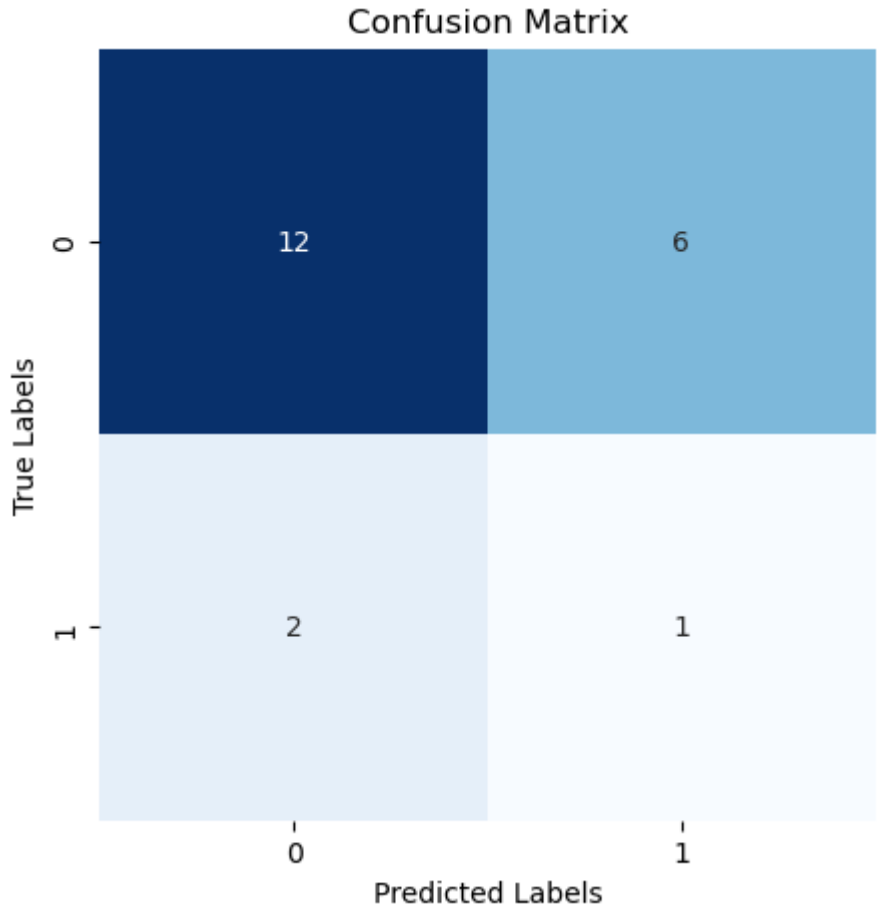
In [81]:
```
v_roc_auc_proba = roc_auc_score(val_y, pred_proba)
v_roc_auc = roc_auc_score(val_y, pred)
v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

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: 0.6666666666666667
Roc AUC: 0.5
Accuracy: 0.6190476190476191
Classification Report:              precision    recall  f1-score   support

           0       0.86      0.67      0.75        18
           1       0.14      0.33      0.20         3

    accuracy                           0.62        21
   macro avg       0.50      0.50      0.47        21
weighted avg       0.76      0.62      0.67        21
```

REPORTING DEL MODELO

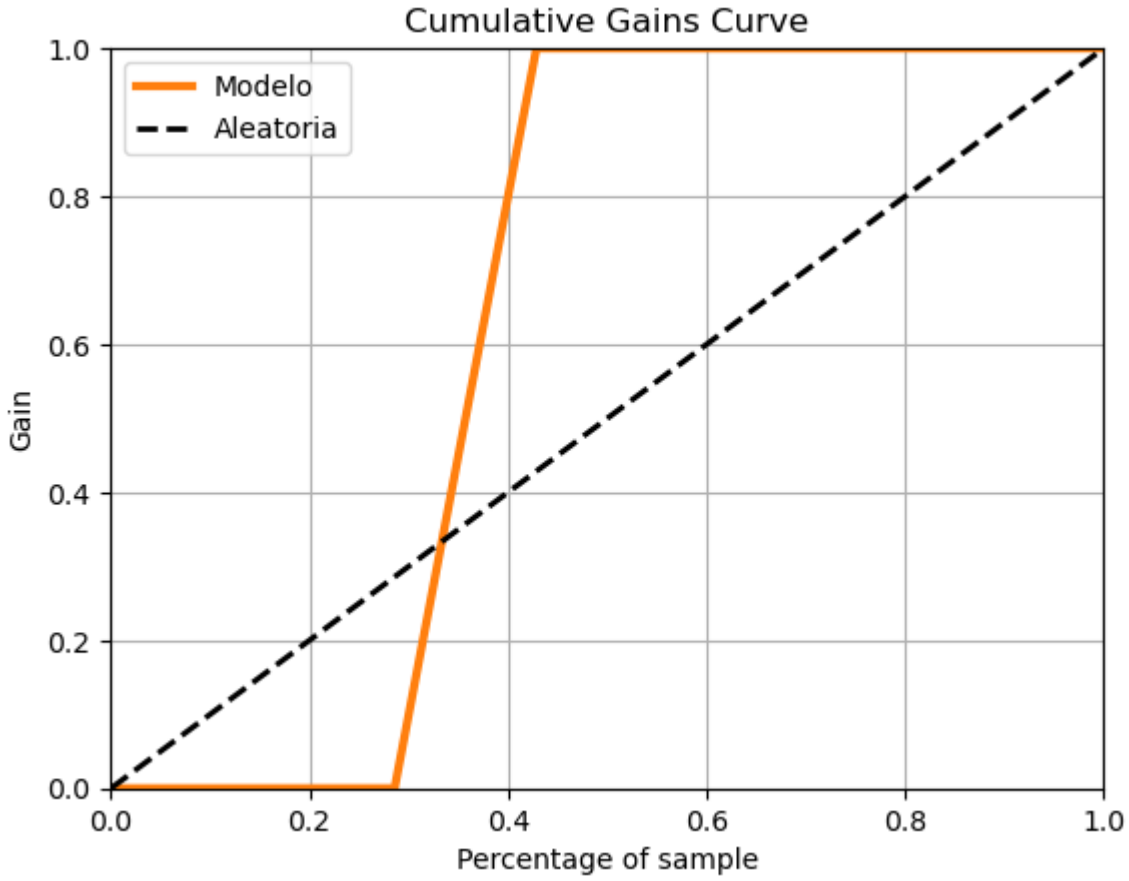
Matrix de Confusión MultiClass

In [82]:
```python
# 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");
```



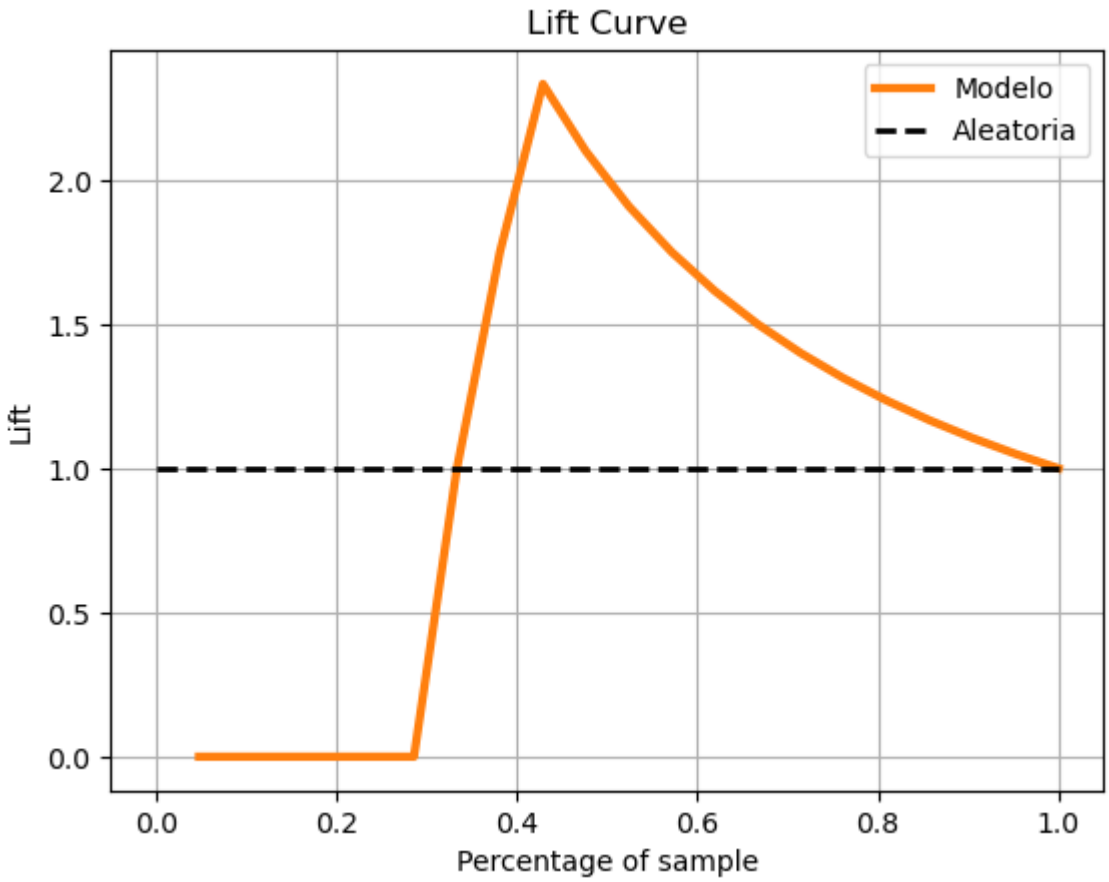
Gain Chart

In [83]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_cumulative_gain(val_y, modelo.best_estimator_.predict_proba(val_

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



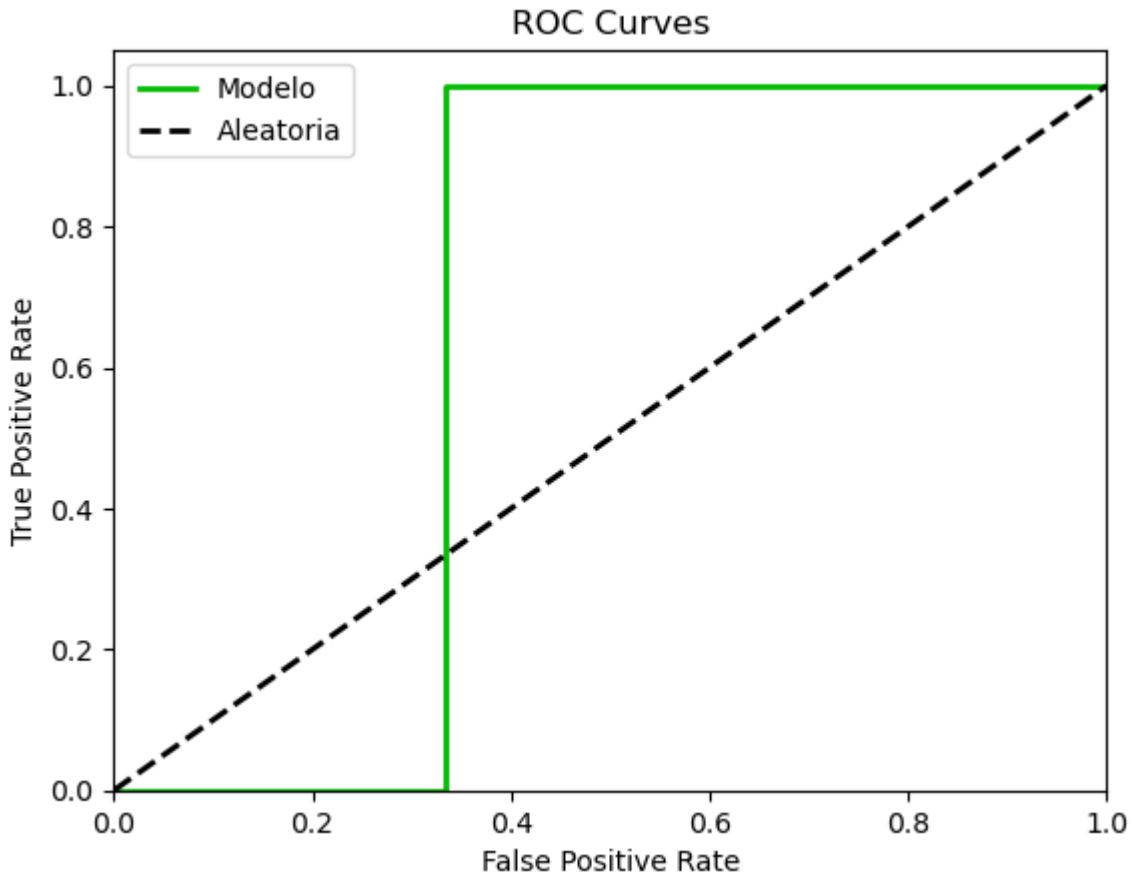
Lift Chart

In [84]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_lift_curve(val_y, modelo.best_estimator_.predict_proba(val_x), a

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



ROC Chart

In [85]:
```
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 [86]:
```
version_estimator = '_v01'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[86]:
```
'AdaBoostClassifier_v01.pickle'
```

In [87]:
```
m_best_estimator
```

Out[87]:
```
'AdaBoostClassifier'
```

In [88]:
```
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 [89]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo Base"
x_columns = list(x.columns)
y_target = y.name
```

In [90]:
```
#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[90]:

```
m_Best_estimator                                    AdaBoostClassifier
m_Best_paramans                   {'algoritmo': AdaBoostClassifier()}
m_Best_Score                                     0.7958333333333334
t_accuracy                                                      1.0
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                             0.666667
v_roc_auc                                                        0.5
v_accuracy                                                  0.619048
v_report                          precision    recall  f1-score   ...
comentarios                                                Modelo Base
predictoras_X       [estacion, edad, e_infantil, acc_grave, int_qu...
target_y                                                    produccion
Name: AdaBoostClassifier_v01.pickle, dtype: object
```

In [91]:

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

In [92]:

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

CONCLUSIÓN: El modelo tiene un AUC del 0.66 por lo que el modelo base no es lo suficientemente bueno para detectar todos los casos positivos.

Vamos a probar el mismo algoritmo parametrizándolo para ver si mejoramos mejoramos.

04 - THE BEST ESTIMATOR_V2

MODELAR ALGORITMO DE CLASIFICACIÓN

Vamos a parametrizar el algoritmo con mejor auc del entrenamiento y evaluación con modelos base.

AdaClassifier()

IMPORTACIÓN DE PAQUETES

In [39]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#Crear datasets de entrenamiento y validación
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.semi_supervised import LabelPropagation
from sklearn.semi_supervised import LabelSpreading
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier

#Optimizar modelo
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

In [40]:
```
df_tablon = pd.read_pickle('../../02_Datos/03_Trabajo/df_tablon.pickle')
df_tablon.head()
```

Out[40]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

SEPARAR PREDICTORAS Y TARGET

In [41]:
```
x = df_tablon.drop(columns= 'produccion').copy()
y = df_tablon.produccion.copy()
```

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [42]:
```
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 [43]:
```
pipe = Pipeline([('algoritmo', AdaBoostClassifier())])

grid =  [
            {   'algoritmo' : [AdaBoostClassifier()],
                'algoritmo__n_estimators': [50, 100, 200, 300],        # Número de e
                'algoritmo__learning_rate': [0.01, 0.1, 1.0, 2.0],    # Tasa de apre
                'algoritmo__algorithm': ['SAMME', 'SAMME.R'],          # Algoritmo
            }
        ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

In [44]:
```
grid_search = GridSearchCV( estimator= pipe,
                            param_grid= grid,
                            cv = 5,
                            scoring= 'roc_auc',
                            verbose=0,
                            n_jobs= -1
                          )

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

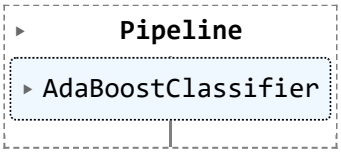
Out[44]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritm
26	0.667923	0.013898	0.076727	0.003749	AdaBoostClassifier(n_estimators=200
11	0.835379	0.127380	0.081897	0.012274	AdaBoostClassifier(n_estimators=200
23	1.013541	0.021972	0.119195	0.003539	AdaBoostClassifier(n_estimators=200
10	0.424889	0.029932	0.045440	0.003422	AdaBoostClassifier(n_estimators=200
9	0.191239	0.003343	0.017566	0.000791	AdaBoostClassifier(n_estimators=200
8	0.098871	0.001621	0.010304	0.000729	AdaBoostClassifier(n_estimators=200
22	0.675778	0.010716	0.076358	0.003010	AdaBoostClassifier(n_estimators=200
24	0.175583	0.007823	0.023754	0.001450	AdaBoostClassifier(n_estimators=200
27	1.014841	0.013604	0.113843	0.002652	AdaBoostClassifier(n_estimators=200
25	0.343522	0.008446	0.043129	0.002666	AdaBoostClassifier(n_estimators=200
21	0.333869	0.005011	0.040898	0.002378	AdaBoostClassifier(n_estimators=200
7	0.565551	0.011609	0.045404	0.002841	AdaBoostClassifier(n_estimators=200
20	0.204315	0.033444	0.025933	0.002273	AdaBoostClassifier(n_estimators=200
12	0.146346	0.007792	0.016758	0.001161	AdaBoostClassifier(n_estimators=200
13	0.313079	0.011333	0.031630	0.003247	AdaBoostClassifier(n_estimators=200
14	0.686774	0.046252	0.054171	0.002729	AdaBoostClassifier(n_estimators=200
18	0.705132	0.014369	0.088536	0.010594	AdaBoostClassifier(n_estimators=200
19	1.102746	0.027435	0.116033	0.002212	AdaBoostClassifier(n_estimators=200
15	1.074501	0.056175	0.085893	0.010046	AdaBoostClassifier(n_estimators=200
17	0.400715	0.030229	0.049979	0.017913	AdaBoostClassifier(n_estimators=200
6	0.395446	0.018057	0.030444	0.001293	AdaBoostClassifier(n_estimators=200
28	0.166504	0.002214	0.023672	0.000692	AdaBoostClassifier(n_estimators=200
29	0.329922	0.005252	0.041326	0.001630	AdaBoostClassifier(n_estimators=200

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritm
31	0.959997	0.078731	0.090462	0.031813	AdaBoostClassifier(n_estimators=200
30	0.717590	0.010365	0.075410	0.003385	AdaBoostClassifier(n_estimators=200
16	0.178420	0.002591	0.024335	0.001493	AdaBoostClassifier(n_estimators=200
5	0.189449	0.002820	0.016968	0.000703	AdaBoostClassifier(n_estimators=200
4	0.095989	0.001077	0.010375	0.000857	AdaBoostClassifier(n_estimators=200
2	0.375842	0.005649	0.031020	0.001196	AdaBoostClassifier(n_estimators=200
3	0.568737	0.017383	0.044310	0.001328	AdaBoostClassifier(n_estimators=200
0	0.109588	0.014387	0.010856	0.001592	AdaBoostClassifier(n_estimators=200
1	0.196331	0.010116	0.016658	0.000396	AdaBoostClassifier(n_estimators=200

In [45]:
```
modelo.best_estimator_
```

Out[45]:



In [46]:
```
modelo.best_params_
```

Out[46]:
```
{'algoritmo': AdaBoostClassifier(n_estimators=200),
 'algoritmo__algorithm': 'SAMME.R',
 'algoritmo__learning_rate': 1.0,
 'algoritmo__n_estimators': 200}
```

In [47]:
```
modelo.best_score_
```

Out[47]:
```
0.8083333333333333
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [48]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [49]:
```
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

PREDECIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el Train

In [50]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [51]:
```
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        43
           1       1.00      1.00      1.00         6

    accuracy                           1.00        49
   macro avg       1.00      1.00      1.00        49
weighted avg       1.00      1.00      1.00        49
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

In [52]:
```
pred = modelo.best_estimator_.predict(val_x)
pred_proba = modelo.best_estimator_.predict_proba(val_x)[:,1]
```

Evaluar sobre la validación

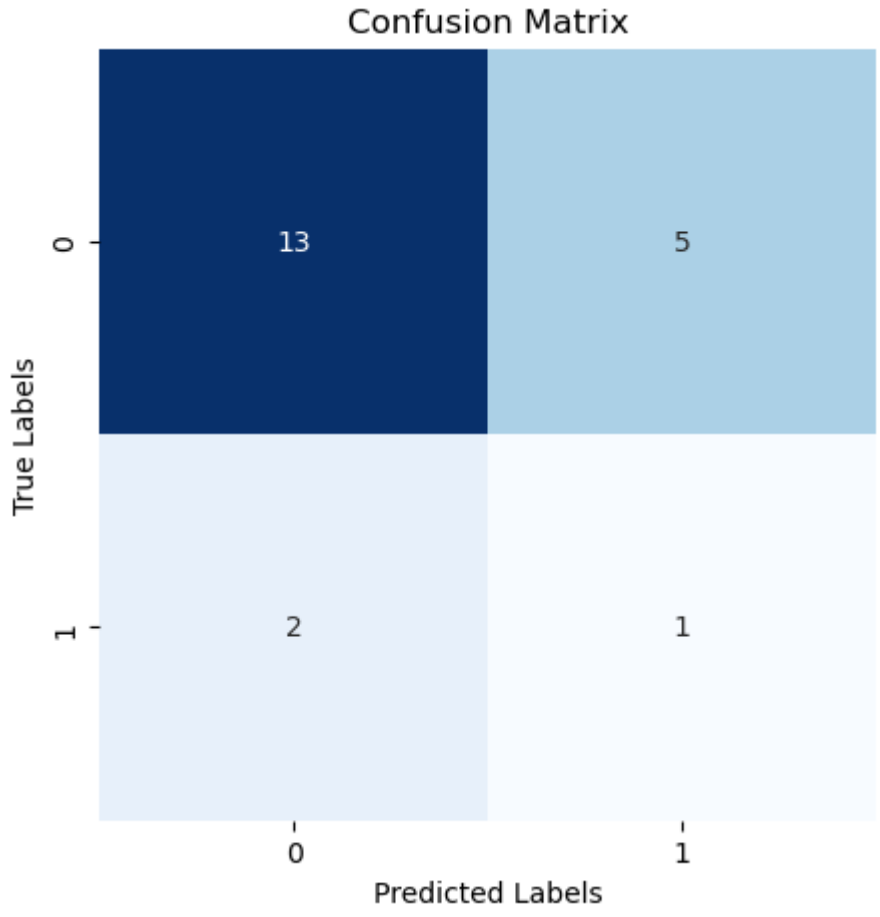
In [53]:
```
v_roc_auc_proba = roc_auc_score(val_y, pred_proba)
v_roc_auc = roc_auc_score(val_y, pred)
v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

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: 0.6666666666666667
Roc AUC: 0.5277777777777778
Accuracy: 0.6666666666666666
Classification Report:              precision    recall  f1-score   support

           0       0.87      0.72      0.79        18
           1       0.17      0.33      0.22         3

    accuracy                           0.67        21
   macro avg       0.52      0.53      0.51        21
weighted avg       0.77      0.67      0.71        21
```

REPORTING DEL MODELO

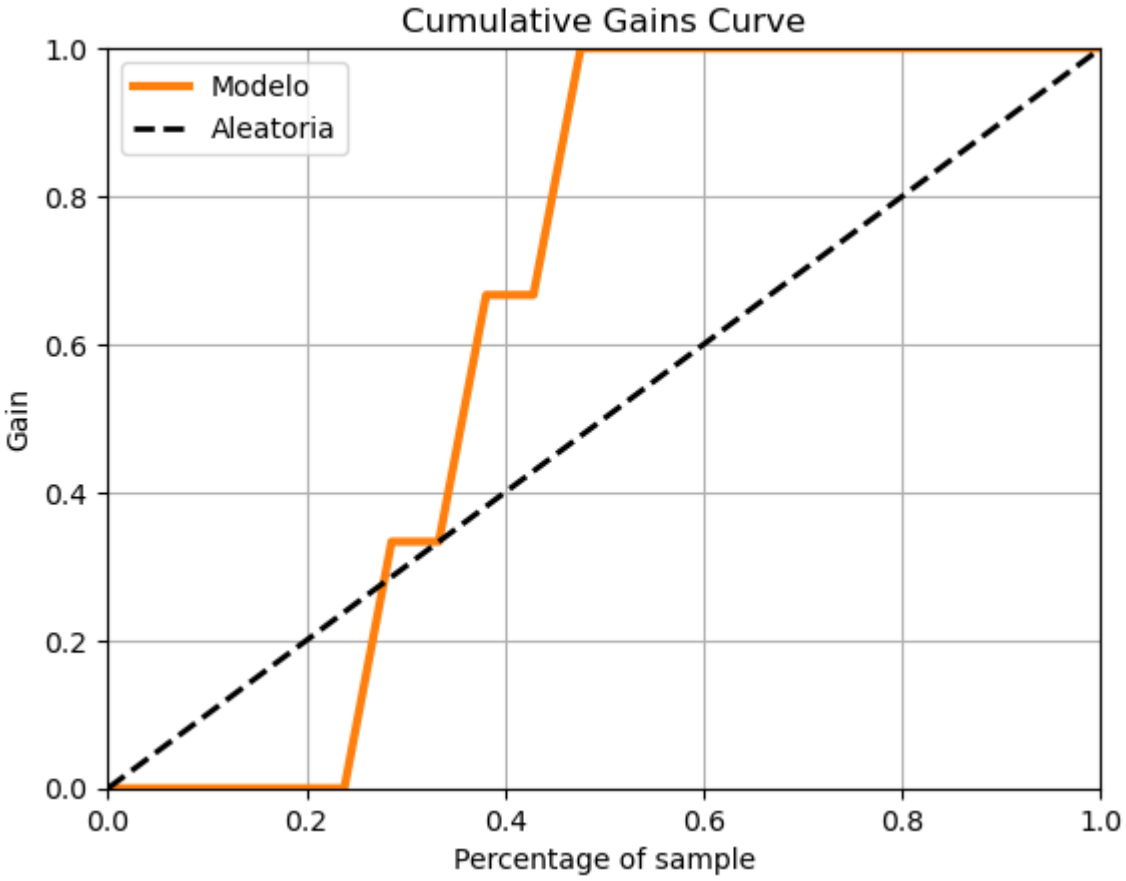
Matrix de Confusión MultiClass

In [54]:
```python
# 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");
```



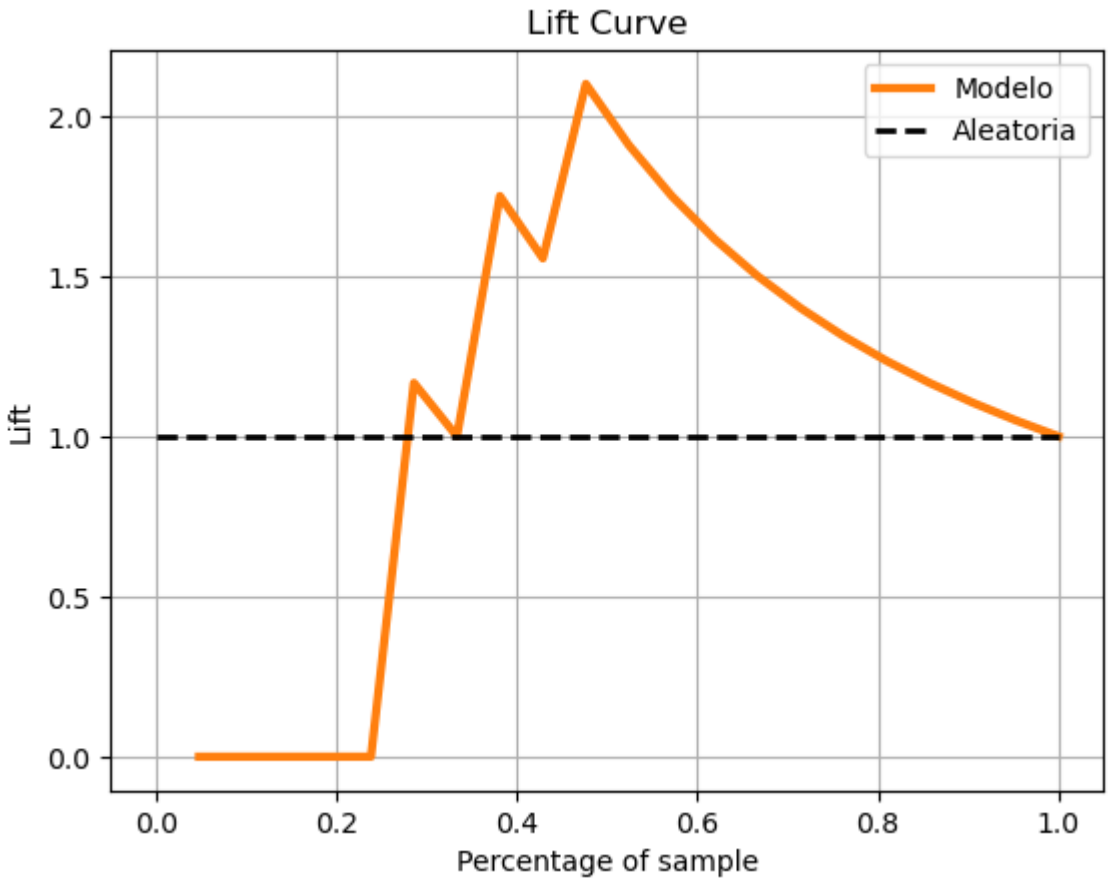
Gain Chart

In [55]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_cumulative_gain(val_y, modelo.best_estimator_.predict_proba(val_

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



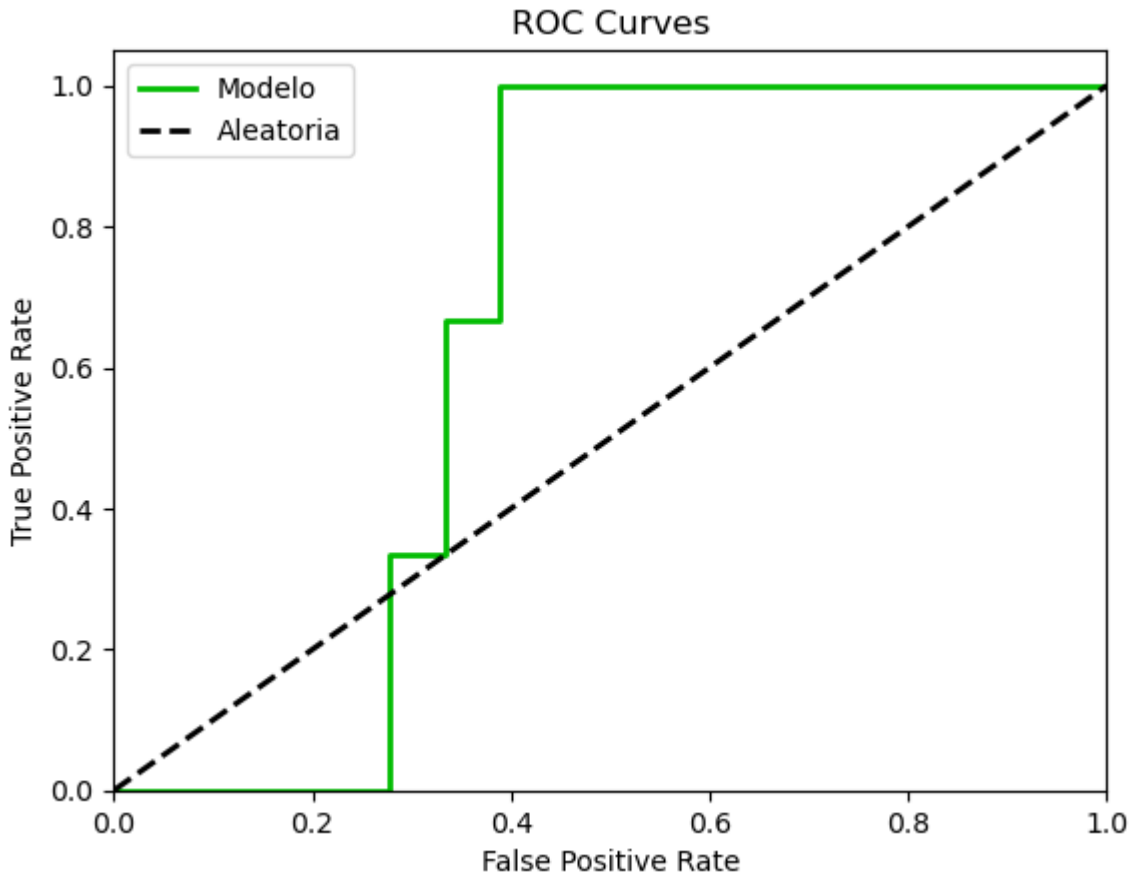
Lift Chart

In [56]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_lift_curve(val_y, modelo.best_estimator_.predict_proba(val_x), a

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



ROC Chart

In [57]:
```
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 [58]:
```
version_estimator = '_v02'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[58]:
```
'AdaBoostClassifier_v02.pickle'
```

In [59]:
```
m_best_estimator
```

Out[59]:
```
'AdaBoostClassifier'
```

In [60]:
```
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 [61]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo Base con parámetros para evitar el sobreajuste"
x_columns = list(x.columns)
y_target = y.name
```

In [62]:
```
#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[62]:

```
m_Best_estimator                                    AdaBoostClassifier
m_Best_paramans     {'algoritmo': AdaBoostClassifier(n_estimators=...
m_Best_Score                                        0.8083333333333333
t_accuracy                                                         1.0
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                                0.666667
v_roc_auc                                                      0.527778
v_accuracy                                                     0.666667
v_report                          precision    recall  f1-score   ...
comentarios         Modelo Base con parámetros para evitar el sobr...
predictoras_X       [estacion, edad, e_infantil, acc_grave, int_qu...
target_y                                                     produccion
Name: AdaBoostClassifier_v02.pickle, dtype: object
```

In [63]:

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

In [64]:

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

CONCLUSIÓN: Incluyendo para de sobreajuste se aprecia una pequeña mejora del modelo.

04 - THE BEST ESTIMATOR_V3

MODELAR ALGORITMO DE CLASIFICACIÓN

Vamos a parametrizar el algoritmo con mejor auc del entrenamiento y evaluación con modelos base.

AdaClassifier()

IMPORTACIÓN DE PAQUETES

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

#Crear datasets de entrenamiento y validación
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.semi_supervised import LabelPropagation
from sklearn.semi_supervised import LabelSpreading
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression

#Optimizar modelo
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

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

Out[2]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

SEPARAR PREDICTORAS Y TARGET

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

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [4]:
```
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 [5]:
```
b_estimator = LogisticRegression(random_state=1 ,class_weight="balanced")


pipe = Pipeline([('algoritmo', AdaBoostClassifier())])

grid =  [
            {   'algoritmo' : [AdaBoostClassifier()],
                'algoritmo__base_estimator' : [b_estimator],
                'algoritmo__n_estimators': [50, 100, 200, 300],        # Número de e
                'algoritmo__learning_rate': [0.01, 0.1, 1.0, 2.0],    # Tasa de apre
                'algoritmo__algorithm': ['SAMME', 'SAMME.R'],      # Algoritmo util
                'algoritmo__base_estimator__C': [0.01, 0.1, 1.0]
            }
        ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

In [6]:
```
grid_search = GridSearchCV( estimator= pipe,
                            param_grid= grid,
                            cv = 5,
                            scoring= 'roc_auc',
                            verbose=0,
                            n_jobs= -1
                          )

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

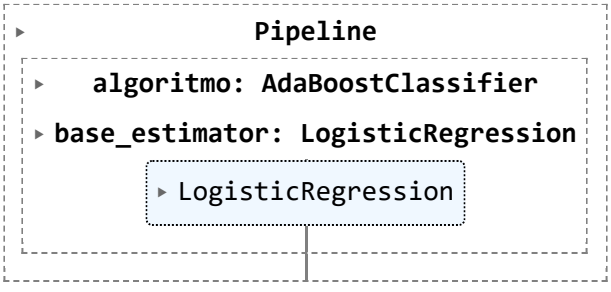
Out[6]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	parar
95	2.778895	0.244768	0.123909	0.017201	AdaBoostClassifier(base_estimator=L
93	1.259121	0.116310	0.067407	0.016628	AdaBoostClassifier(base_estimator=L
90	2.004140	0.075447	0.117490	0.009926	AdaBoostClassifier(base_estimator=L
94	2.323587	0.266636	0.122056	0.011561	AdaBoostClassifier(base_estimator=L
91	3.215461	0.206638	0.189759	0.017354	AdaBoostClassifier(base_estimator=L
...	...	...	...	...	
31	2.627707	1.291814	0.080848	0.039477	AdaBoostClassifier(base_estimator=L
26	0.027321	0.008426	0.011236	0.008153	AdaBoostClassifier(base_estimator=L
29	0.930358	0.463201	0.036338	0.024391	AdaBoostClassifier(base_estimator=L
33	0.944708	0.023970	0.033700	0.011944	AdaBoostClassifier(base_estimator=L
16	0.469497	0.112583	0.020863	0.013517	AdaBoostClassifier(base_estimator=L

96 rows × 19 columns

In [7]:
```
modelo.best_estimator_
```

Out[7]:



In [8]:
```
modelo.best_params_
```

Out[8]:
```
{'algoritmo': AdaBoostClassifier(base_estimator=LogisticRegression(class_weight='balanced',
                                                                   random_state=1),
                                 learning_rate=2.0, n_estimators=300),
 'algoritmo__algorithm': 'SAMME.R',
 'algoritmo__base_estimator': LogisticRegression(class_weight='balanced', random_state=1),
 'algoritmo__base_estimator__C': 1.0,
 'algoritmo__learning_rate': 2.0,
 'algoritmo__n_estimators': 300}
```

In [9]:
```
modelo.best_score_
```

Out[9]:
```
0.5847222222222223
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [10]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [11]:
```
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

PREDECIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el Train

In [12]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [13]:
```
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: 0.8468992248062016
Accuracy: 0.8571428571428571
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.86      0.91        43
           1       0.45      0.83      0.59         6

    accuracy                           0.86        49
   macro avg       0.71      0.85      0.75        49
weighted avg       0.91      0.86      0.87        49
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

In [14]:
```
pred = modelo.best_estimator_.predict(val_x)
pred_proba = modelo.best_estimator_.predict_proba(val_x)[:,1]
```

Evaluar sobre la validación

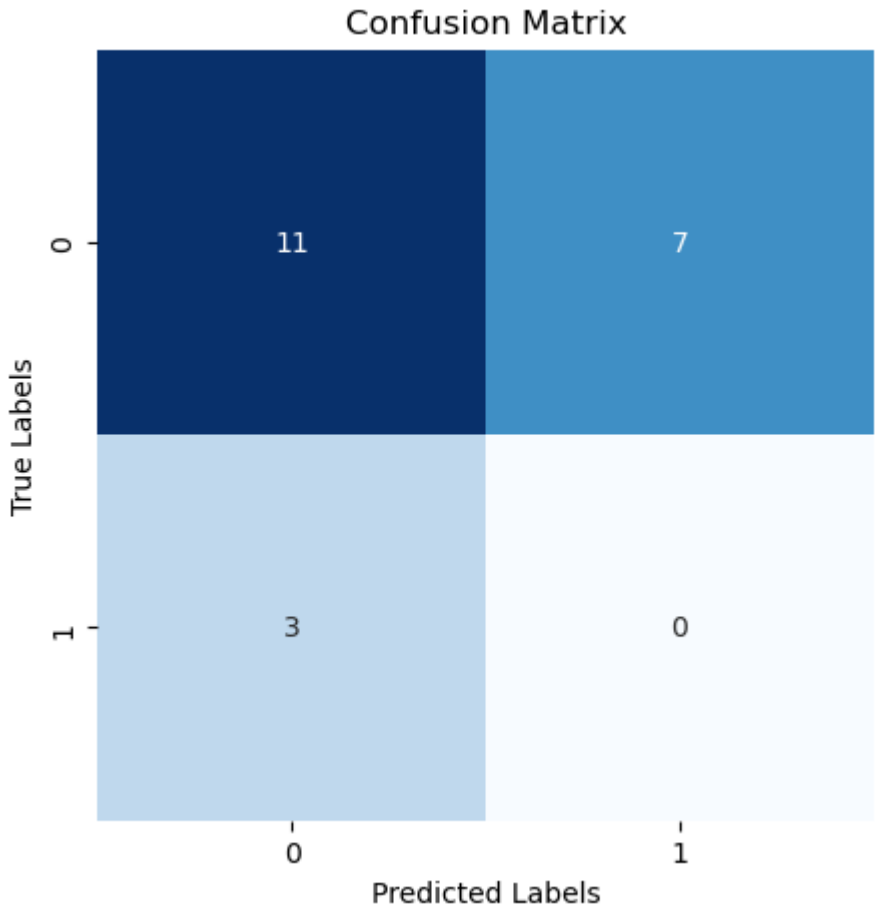
In [15]:
```
v_roc_auc_proba = roc_auc_score(val_y, pred_proba)
v_roc_auc = roc_auc_score(val_y, pred)
v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

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: 0.5
Roc AUC: 0.3055555555555556
Accuracy: 0.5238095238095238
Classification Report:              precision    recall  f1-score   support

           0       0.79      0.61      0.69        18
           1       0.00      0.00      0.00         3

    accuracy                           0.52        21
   macro avg       0.39      0.31      0.34        21
weighted avg       0.67      0.52      0.59        21
```

REPORTING DEL MODELO

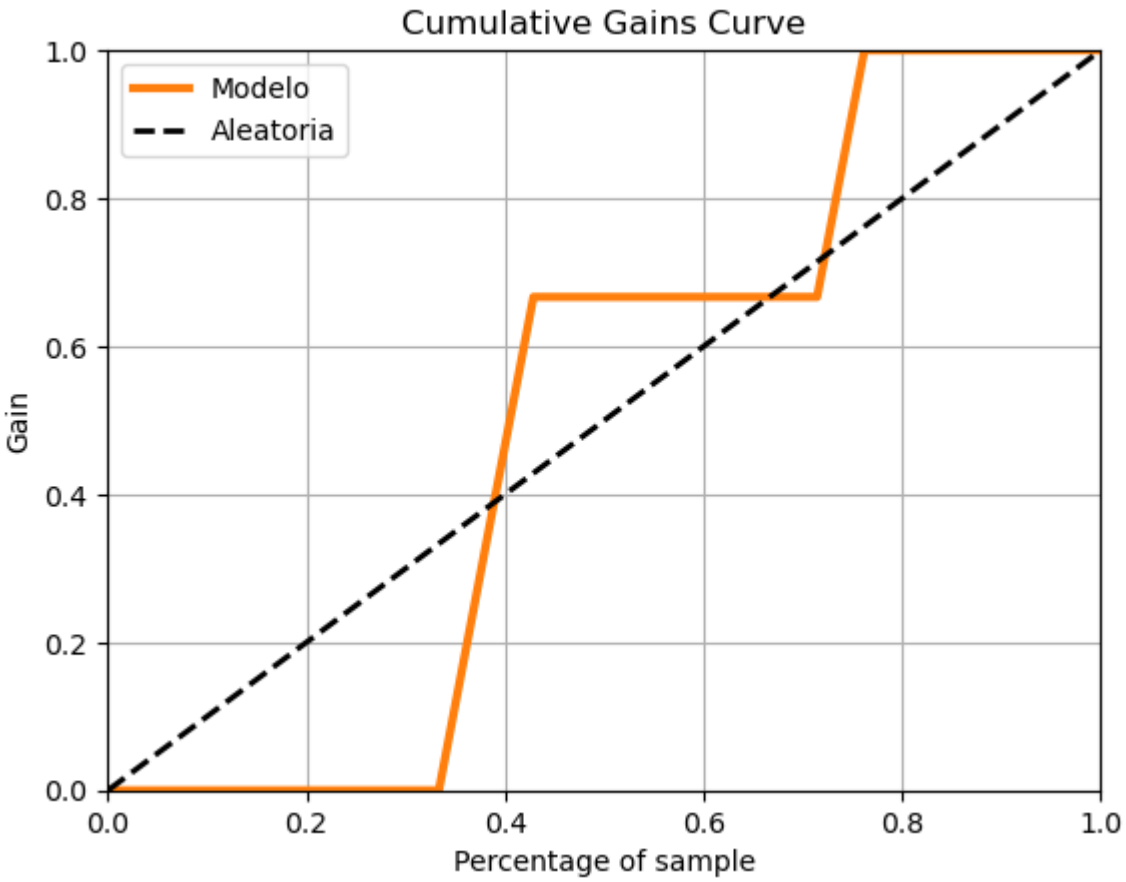
Matrix de Confusión MultiClass

In [16]:
```
# 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");
```



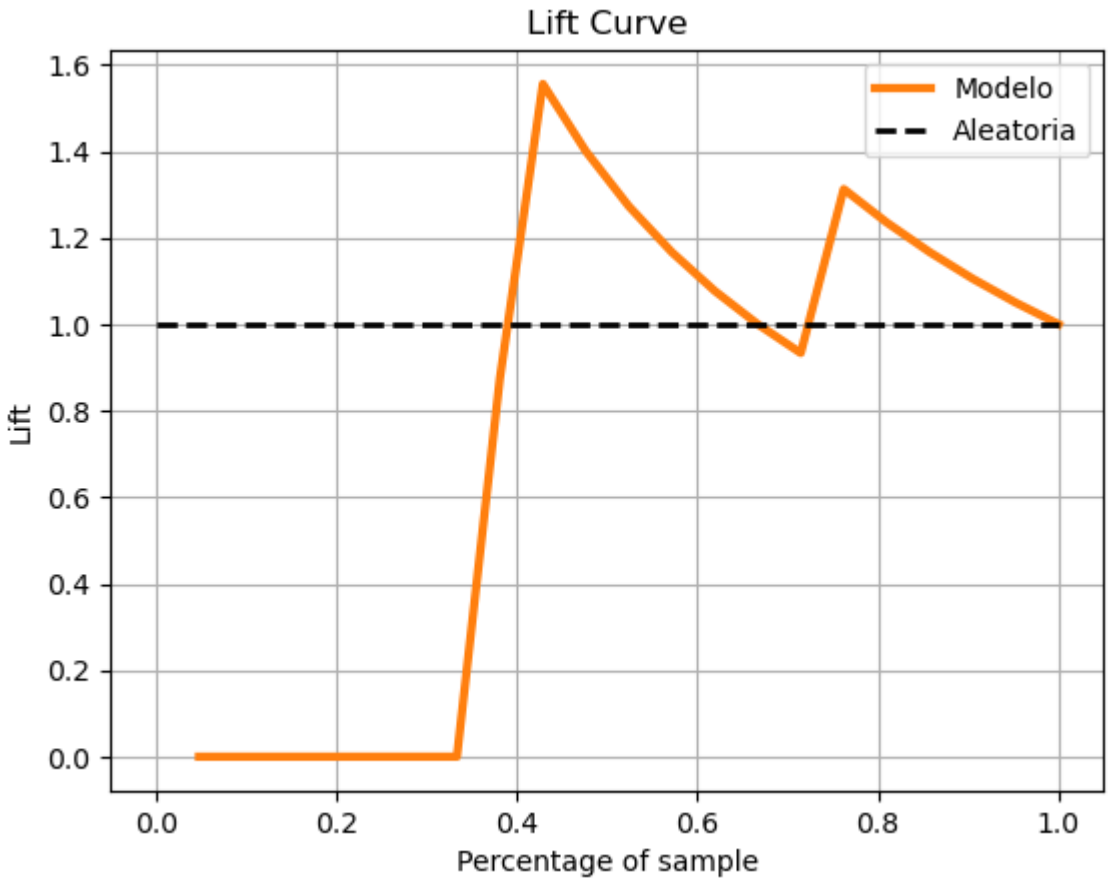
Gain Chart

In [17]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_cumulative_gain(val_y, modelo.best_estimator_.predict_proba(val_

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



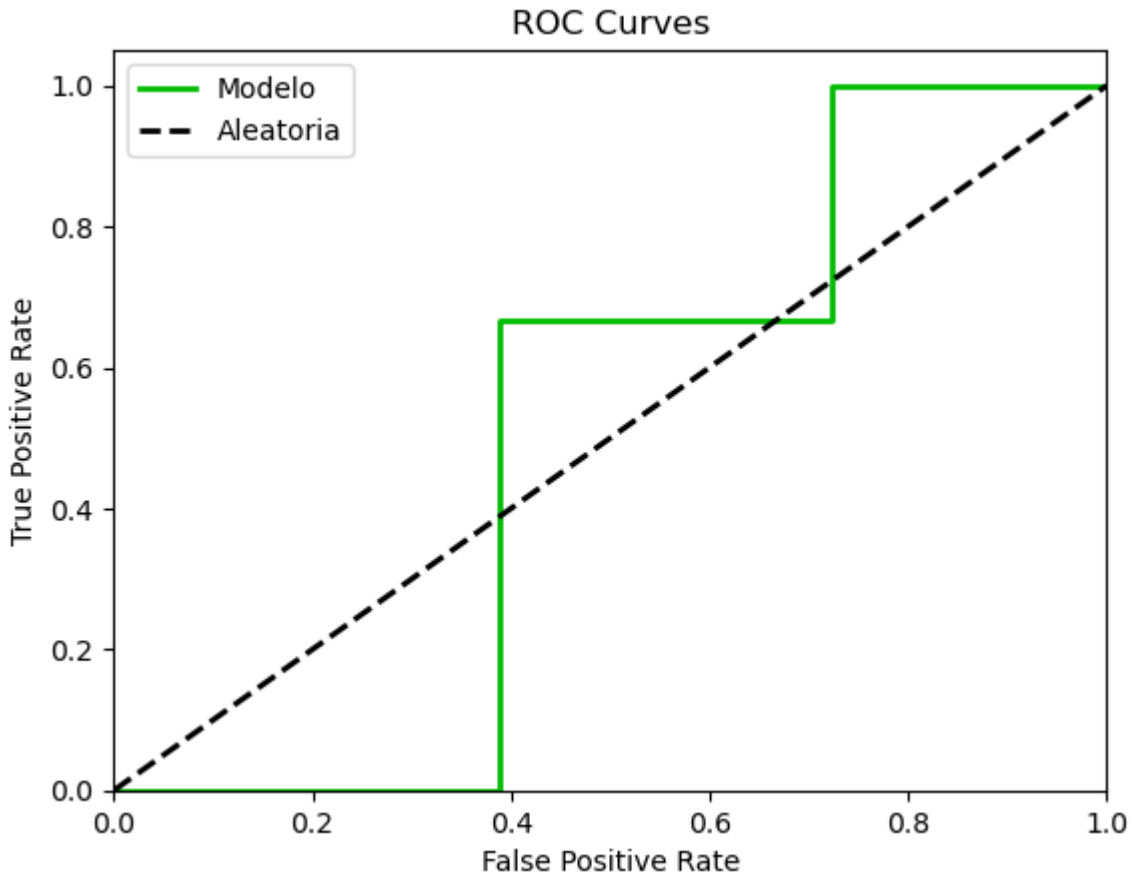
Lift Chart

In [18]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_lift_curve(val_y, modelo.best_estimator_.predict_proba(val_x), a

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



ROC Chart

In [19]:
```
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 [20]:
```
version_estimator = '_v02_2'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[20]:
```
'AdaBoostClassifier_v02_1.pickle'
```

In [21]:
```
m_best_estimator
```

Out[21]:
```
'AdaBoostClassifier'
```

In [22]:
```
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 [23]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo Base con parámetros para evitar el sobreajuste  y añadimos ba
x_columns = list(x.columns)
y_target = y.name
```

In [24]:
```
#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[24]:

```
m_Best_estimator                                    AdaBoostClassifier
m_Best_paramans     {'algoritmo': AdaBoostClassifier(base_estimato...
m_Best_Score                                        0.5847222222222223
t_accuracy                                                    0.857143
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                                    0.5
v_roc_auc                                                     0.305556
v_accuracy                                                     0.52381
v_report                          precision    recall  f1-score   ...
comentarios         Modelo Base con parámetros para evitar el sobr...
predictoras_X       [estacion, edad, e_infantil, acc_grave, int_qu...
target_y                                                     produccion
Name: AdaBoostClassifier_v02_1.pickle, dtype: object
```

In [25]:

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

In [26]:

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

CONCLUSIÓN: Incluyendo para de sobreajuste se aprecia una pequeña mejora del modelo.

05 - PRESELECCIÓN DE VARIABLES

Es este notebook vamos a realizar diferentes métodos para detectar las mejores variables predictivas y ayudar al modelo a reducir la dimensiones y mejorar las métricas predictivas.

IMPORTACIÓN DE PAQUETES

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

#Métodos supervisados
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import RFECV
from sklearn.inspection import permutation_importance

from xgboost import XGBClassifier
```

IMPORTAR LOS DATOS

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

Out[2]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

SEPARAR PREDICTORAS Y TARGET

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

In [4]:
```
#Revisamos predicoras
x.head()
```

Out[4]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

In [5]:
```
#revisamos las target
y.head()
```

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

MÉTODOS SUPERVISADOS

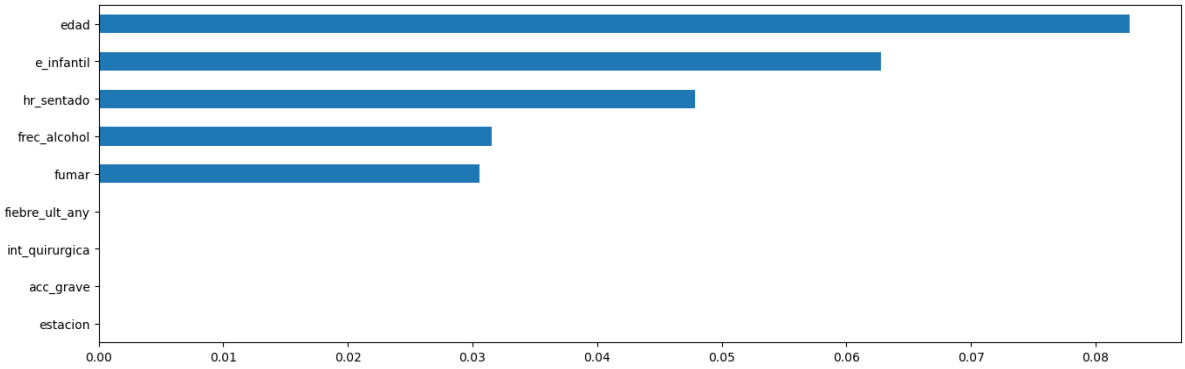
Mutual Information

Crear una función para mostrar el resultado

In [6]:
```
def ranking_mi(mutual_selector, modo = 'tabla'):
    #Maqueta el ranking
    ranking_mi = pd.DataFrame(mutual_selector, index = x.columns).reset_index()
    ranking_mi.columns = ['variable','importancia_mi']
    ranking_mi = ranking_mi.sort_values(by = 'importancia_mi', ascending = False)
    ranking_mi['ranking_mi'] = np.arange(0,ranking_mi.shape[0])
    #Muestra la salida
    if modo == 'tabla':
        return(ranking_mi)
    else:
        g = ranking_mi.importancia_mi.sort_values().plot.barh(figsize = (16,5))
        g.set_yticklabels(ranking_mi.sort_values(by = 'importancia_mi').variable)
        return(g)
```

Calcular y revisar

In [7]:
```
mutual_selector = mutual_info_classif(x,y)

rank_mi = ranking_mi(mutual_selector, modo = 'grafico')
```



Seleccionar las variables que pasan

Definir la posición de la última variable que va a entrar

In [8]:
```
posicion_variable_limite = 6
```

Extraer los nombres de las que entran

In [9]:
```
entran_mi = ranking_mi(mutual_selector).iloc[0:posicion_variable_limite].variable
```

Crear el dataframe con la selección

In [10]:
```
x_mi = x[entran_mi].copy()
```

Recursive Feature Elimination

Instanciar

In [11]:
```
rfe = RFECV(estimator = XGBClassifier(use_label_encoder=False, n_jobs = -1, eval_me
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

Entrenar

In [12]:
```
rfe.fit(x,y)
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
```

```
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
```

```
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

Out[12]:

RFECV

estimator: XGBClassifier

XGBClassifier

Extraer los nombres de las que entran

In [13]:

```
entran_rfe = x.columns[rfe.support_]
entran_rfe
```

Out[13]:

```
Index(['frec_alcohol'], dtype='object')
```

Crear el dataframe con la selección

In [14]:

```
x_rfe = x[entran_rfe].copy()
```

Permutation Importance

Crear una función para mostrar el resultado

In [15]:

```
def ranking_per(predictoras,permutacion):
    ranking_per = pd.DataFrame({'variable': predictoras.columns, 'importancia_per'
```

```
    ranking_per['ranking_per'] = np.arange(0,ranking_per.shape[0])
    return(ranking_per)
```

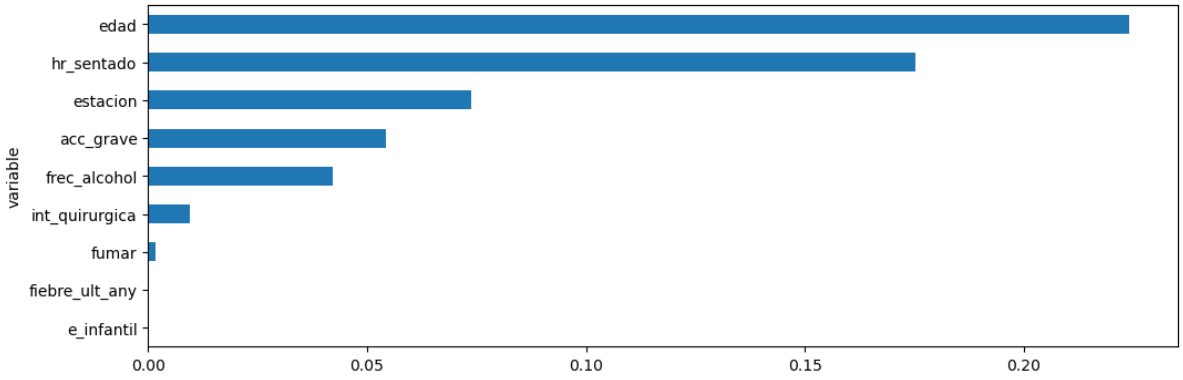
Instanciar y entrenar

In [16]:
```
import warnings
warnings.filterwarnings(action="ignore", message=r'.*Use subset.*of np.ndarray is r

xgb = XGBClassifier(use_label_encoder=False, n_jobs = -1, eval_metric='auc')

xgb.fit(x,y)

permutacion = permutation_importance(xgb,
                                     x, y,
                                     scoring = 'roc_auc',
                                     n_repeats=5, n_jobs = -1)
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

Revisar la salida

In [17]:
```
rank_per = ranking_per(x,permutacion)

rank_per.set_index('variable').importancia_per.sort_values().plot.barh(figsize = (1
```



Seleccionar las variables que pasan

Definir la posición de la última variable que va a entrar

In [18]:
```
posicion_variable_limite = 6
```

Extraer los nombres de las que entran

In [19]:
```
entran_per = rank_per.iloc[0:posicion_variable_limite].variable
```

Crear el dataframe con la selección

In [20]:
```
x_per = x[entran_per].copy()
```

SELECCIONAR EL MÉTODO FINAL

Descomentar el método de preselección elegido y dejar comentados el resto.

In [21]:
```
x_preseleccionado = x_mi
# x_preseleccionado = x_rfe
# x_preseleccionado = x_per
```

MÉTODOS NO SUPERVISADOS

Correlación

Crear una función para mostrar el resultado

In [22]:
```
def correlaciones_fuertes(df, lim_inf = 0.3, lim_sup = 1,drop_dupli=True):
    #Calcula la matriz de correlación
    c = df.corr().abs()
    #Lo pasa todo a filas
    c= c.unstack()
    #Pasa el índice a columnas y le pone nombres
    c = pd.DataFrame(c).reset_index()
    c.columns = ['var1','var2','corr']
    #A dataframe, filtra limites y ordena en descendiente
    c = c.loc[(c['corr'] > lim_inf) &  (c['corr'] < lim_sup),:].sort_values(by = '
    #Desduplica las correlaciones (o no si drop_dupli es False)
    c = c if drop_dupli == False else c.drop_duplicates(subset = ['corr'])
    #Devuelve la salida
    return(c)
```

Calcular y revisar

Calcular

In [23]:
```
cor_finales = correlaciones_fuertes(x_preseleccionado)
```

Revisar agregado

In [24]:
```
cor_finales.var1.value_counts()
```

Out[24]:
```
edad    1
Name: var1, dtype: int64
```

Revisar detalle

In [25]:
```
cor_finales.head(50)
```

Out[25]:

	var1	var2	corr
2	edad	hr_sentado	0.402714

In [26]:
```
x_preseleccionado.columns.to_list()
```

Out[26]:
```
['edad', 'e_infantil', 'hr_sentado', 'frec_alcohol', 'fumar', 'estacion']
```

CONCLUSIÓN: La correlación no es muy fuerte entre edad y hr. sentado, pero hr.sentado no es una variable importante en el mutual information por lo que la eliminaremos del dataset definitivo.

Los próximos notebooks trabajaremos los mejores modelos anteriores con la preselección de variables.

- v2 --> ExtraTreesClassifer con parámetros
- v3 --> XGBClassifier con parámetros

GUARDAR DATASETS TRAS PRESELECCION DE VARIABLES

In [27]:
```
#Definir los nombres de los archivos
nombre_x_preseleccionado = '../../02_Datos/03_Trabajo/' + 'x_preseleccionado.pickle
nombre_y_preseleccionado = '../../02_Datos/03_Trabajo/' + 'y_preseleccionado.pickle
```

In [28]:
```
#Guardar los archivos
x_preseleccionado.to_pickle(nombre_x_preseleccionado)

y_preseleccionado = y.copy()
y_preseleccionado.to_pickle(nombre_y_preseleccionado)
```

06_01 THE BEST ESTIMATOR_V3

MODELAR ALGORITMO DE CLASIFICACIÓN

Vamos a parametrizar el algoritmo con mejor roc_auc del entrenamiento y evaluación con modelos base y preselección de variables.

ExtraTreesClassifier()

IMPORTACIÓN DE PAQUETES

In [34]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#Crear datasets de entrenamiento y validación
from sklearn.model_selection import train_test_split

#Modelos de clasificación
from sklearn.ensemble import AdaBoostClassifier


#Optimizar modelo
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import Pipeline


import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

In [35]:
```
x = pd.read_pickle('../../02_Datos/03_Trabajo/x_preseleccionado.pickle')
y = pd.read_pickle('../../02_Datos/03_Trabajo/y_preseleccionado.pickle')
```

In [36]:
```
#Verificamos predictoras

print(x.shape)
x.head()
```

(70, 6)

Out[36]:

	edad	e_infantil	hr_sentado	frec_alcohol	fumar	estacion
0	0.69	0	0.88	0.8	0	-0.33
1	0.94	1	0.31	0.8	1	-0.33
2	0.50	1	0.50	1.0	-1	-0.33
4	0.67	1	0.50	0.8	-1	-0.33
5	0.67	1	0.50	0.8	0	-0.33

In [37]:
```
#Verificamos target
print(y.shape)
y.head()
```

Out[37]:
```
(70,)
0    0
1    1
2    0
4    1
5    0
Name: produccion, dtype: int64
```

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [38]:
```
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 [39]:
```
pipe = Pipeline([('algoritmo', AdaBoostClassifier())])

grid =  [
            {   'algoritmo' : [AdaBoostClassifier()],
                'algoritmo__n_estimators': [50, 100, 200, 300],        # Número de e
                'algoritmo__learning_rate': [0.01, 0.1, 1.0, 2.0],    # Tasa de apre
                'algoritmo__algorithm': ['SAMME', 'SAMME.R'],          # Algoritmo
            }
        ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

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

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

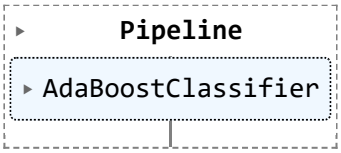
Out[40]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritm
27	1.057990	0.020337	0.121624	0.011655	AdaBoostClassifier(n_estimators=300
8	0.095393	0.005164	0.009888	0.000473	AdaBoostClassifier(n_estimators=300
26	0.688964	0.018538	0.078027	0.002573	AdaBoostClassifier(n_estimators=300
25	0.352953	0.017156	0.044461	0.001986	AdaBoostClassifier(n_estimators=300
9	0.203824	0.004359	0.018064	0.002584	AdaBoostClassifier(n_estimators=300
10	0.565253	0.107229	0.053490	0.016583	AdaBoostClassifier(n_estimators=300
23	1.076932	0.046423	0.122175	0.011066	AdaBoostClassifier(n_estimators=300
24	0.187330	0.022576	0.027969	0.005581	AdaBoostClassifier(n_estimators=300
11	0.667154	0.074423	0.045878	0.001343	AdaBoostClassifier(n_estimators=300
7	0.677776	0.133265	0.052352	0.005815	AdaBoostClassifier(n_estimators=300
21	0.250655	0.047865	0.035657	0.004907	AdaBoostClassifier(n_estimators=300
22	0.692910	0.060558	0.079524	0.003943	AdaBoostClassifier(n_estimators=300
31	0.792906	0.079115	0.061496	0.006196	AdaBoostClassifier(n_estimators=300
28	0.168582	0.002858	0.023634	0.001527	AdaBoostClassifier(n_estimators=300
29	0.357290	0.010430	0.050734	0.007765	AdaBoostClassifier(n_estimators=300
19	0.858923	0.193875	0.117608	0.017694	AdaBoostClassifier(n_estimators=300
18	0.470404	0.034821	0.049644	0.003560	AdaBoostClassifier(n_estimators=300
30	0.696173	0.015255	0.080442	0.010323	AdaBoostClassifier(n_estimators=300
15	0.658635	0.057223	0.048545	0.005176	AdaBoostClassifier(n_estimators=300
14	0.398887	0.020582	0.031788	0.002459	AdaBoostClassifier(n_estimators=300
13	0.226317	0.019268	0.017053	0.000658	AdaBoostClassifier(n_estimators=300
12	0.162487	0.044966	0.021172	0.023156	AdaBoostClassifier(n_estimators=300
20	0.111200	0.010360	0.013663	0.000421	AdaBoostClassifier(n_estimators=300

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritm
6	0.401846	0.035868	0.031956	0.001485	AdaBoostClassifier(n_estimators=30(
17	0.245434	0.014846	0.025858	0.001465	AdaBoostClassifier(n_estimators=30(
5	0.246853	0.022648	0.019411	0.003445	AdaBoostClassifier(n_estimators=30(
4	0.102690	0.008704	0.010760	0.001571	AdaBoostClassifier(n_estimators=30(
16	0.116102	0.013338	0.016770	0.003116	AdaBoostClassifier(n_estimators=30(
2	0.458286	0.056520	0.040139	0.005517	AdaBoostClassifier(n_estimators=30(
3	0.624140	0.045691	0.044104	0.001046	AdaBoostClassifier(n_estimators=30(
0	0.108761	0.011911	0.013372	0.006618	AdaBoostClassifier(n_estimators=30(
1	0.201648	0.018528	0.016885	0.000664	AdaBoostClassifier(n_estimators=30(

In [41]:
```
modelo.best_estimator_
```

Out[41]:



In [42]:
```
modelo.best_params_
```

Out[42]:
```
{'algoritmo': AdaBoostClassifier(n_estimators=300),
 'algoritmo__algorithm': 'SAMME.R',
 'algoritmo__learning_rate': 1.0,
 'algoritmo__n_estimators': 300}
```

In [43]:
```
modelo.best_score_
```

Out[43]:
```
0.7458333333333333
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [44]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [45]:
```
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

PREDECIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el Train

In [46]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [47]:
```
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        43
           1       1.00      1.00      1.00         6

    accuracy                           1.00        49
   macro avg       1.00      1.00      1.00        49
weighted avg       1.00      1.00      1.00        49
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

In [48]:
```
pred = modelo.best_estimator_.predict(val_x)
pred_proba = modelo.best_estimator_.predict_proba(val_x)[:,1]
```

Evaluar sobre la validación

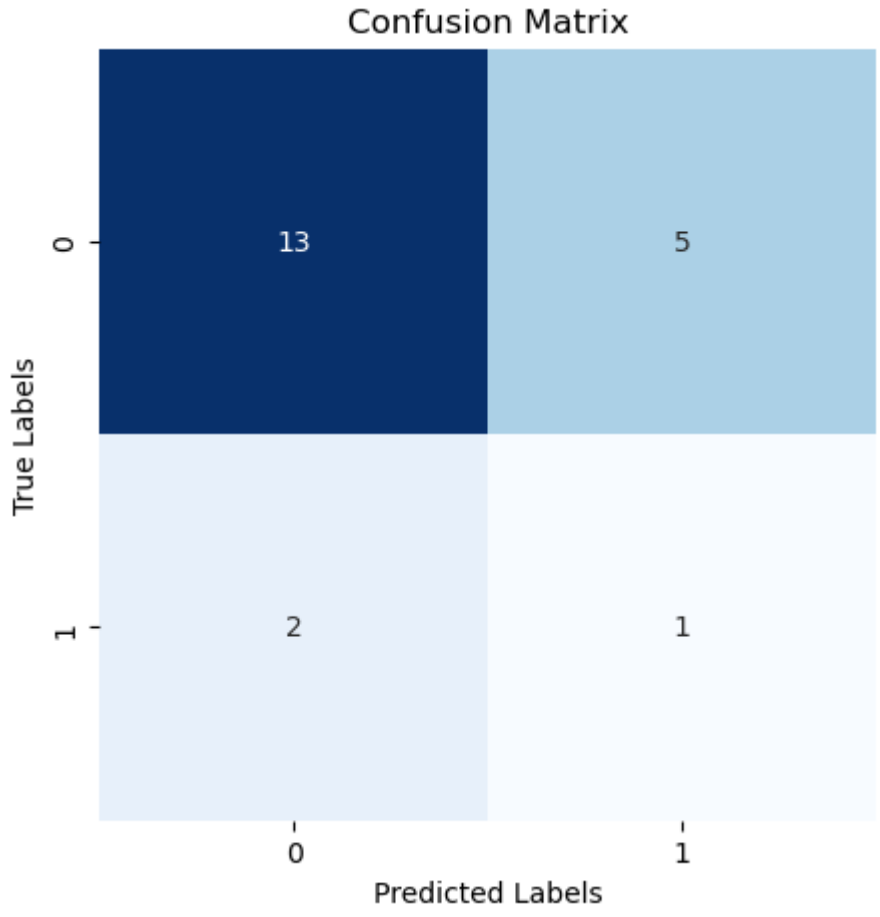
In [49]:
```
v_roc_auc_proba = roc_auc_score(val_y, pred_proba)
v_roc_auc = roc_auc_score(val_y, pred)
v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

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: 0.574074074074074
Roc AUC: 0.5277777777777778
Accuracy: 0.6666666666666666
Classification Report:              precision    recall  f1-score   support

           0       0.87      0.72      0.79        18
           1       0.17      0.33      0.22         3

    accuracy                           0.67        21
   macro avg       0.52      0.53      0.51        21
weighted avg       0.77      0.67      0.71        21
```

REPORTING DEL MODELO

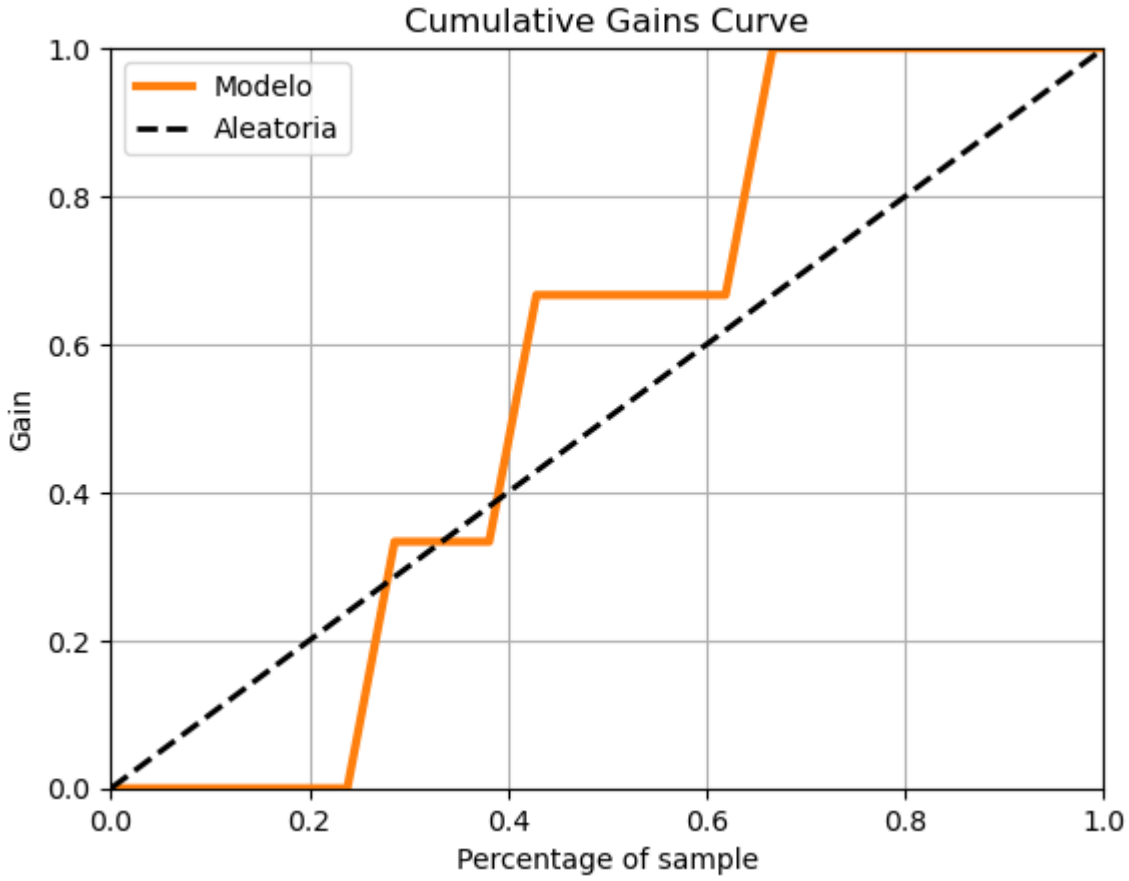
Matrix de Confusión MultiClass

In [50]:
```python
# 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");
```



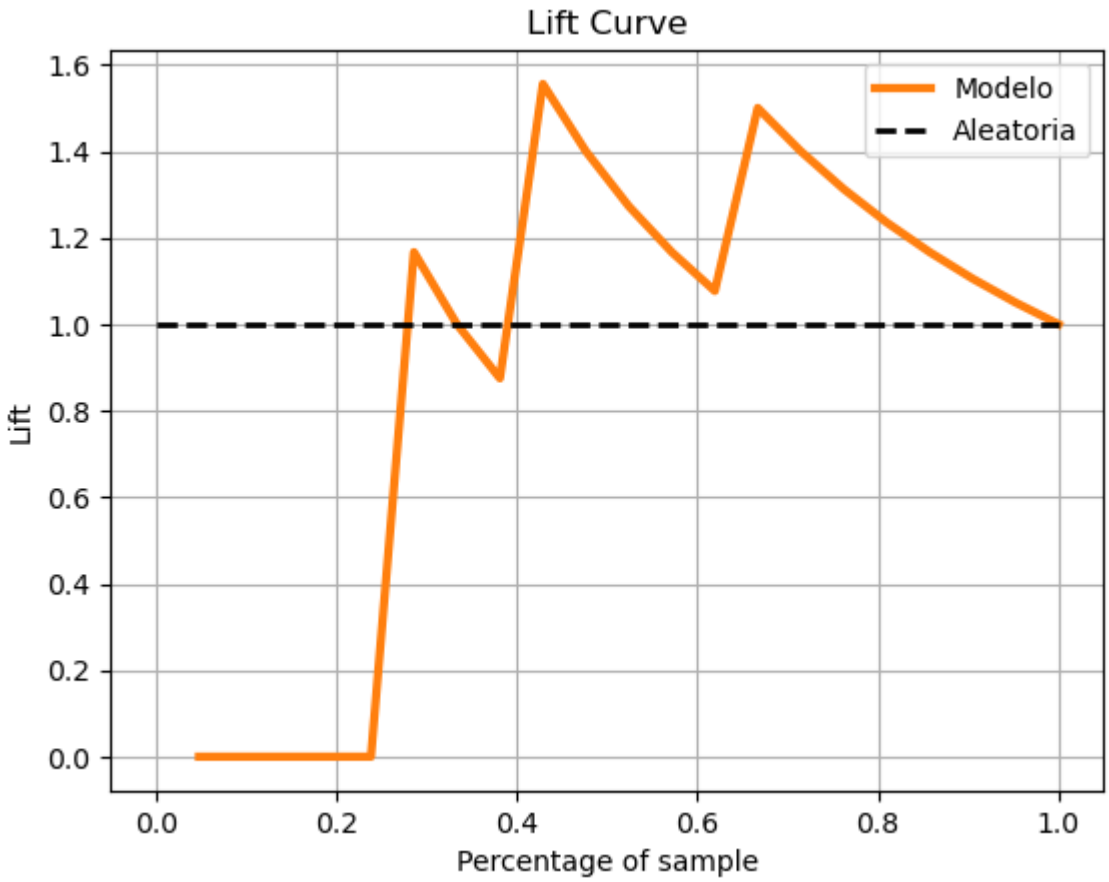
Gain Chart

In [51]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_cumulative_gain(val_y, modelo.best_estimator_.predict_proba(val_

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



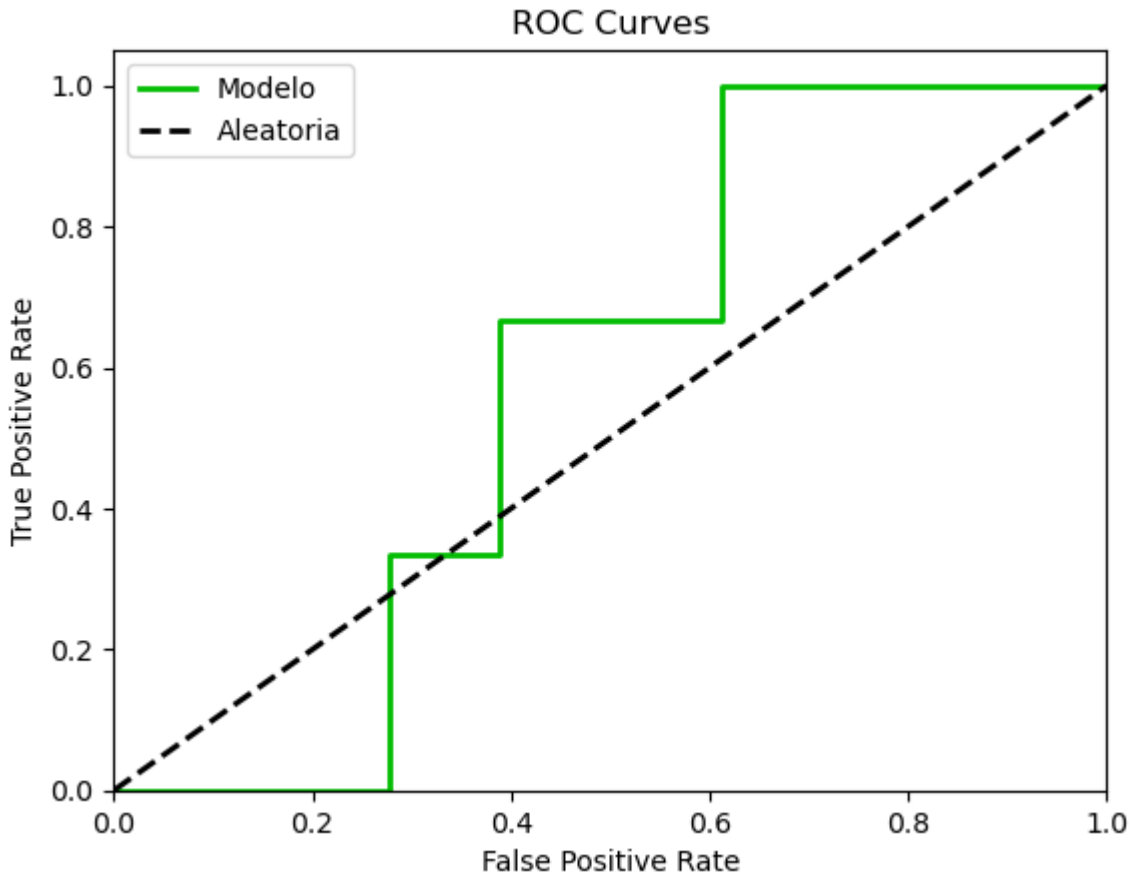
Lift Chart

In [52]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_lift_curve(val_y, modelo.best_estimator_.predict_proba(val_x), a

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



ROC Chart

In [53]:
```
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 [54]:
```
version_estimator = '_v03'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[54]:
```
'AdaBoostClassifier_v03.pickle'
```

In [55]:
```
m_best_estimator
```

Out[55]:
```
'AdaBoostClassifier'
```

In [56]:
```
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 [57]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo AdaClassifier con parámetros para evitar el sobreajuste y pre
x_columns = list(x.columns)
y_target = y.name
```

In [58]:
```
#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[58]:

```
m_Best_estimator                                    AdaBoostClassifier
m_Best_paramans     {'algoritmo': AdaBoostClassifier(n_estimators=...
m_Best_Score                                        0.7458333333333333
t_accuracy                                                         1.0
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                                0.574074
v_roc_auc                                                      0.527778
v_accuracy                                                     0.666667
v_report                          precision    recall  f1-score   ...
comentarios         Modelo AdaClassifier con parámetros para evita...
predictoras_X       [edad, e_infantil, hr_sentado, frec_alcohol, f...
target_y                                                     produccion
Name: AdaBoostClassifier_v03.pickle, dtype: object
```

In [59]:

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

In [60]:

```
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 la preselección de variables no ha ayudado a mejorar la predicción del modelo.

06_02 THE BEST ESTIMATOR_V4

MODELAR ALGORITMO DE CLASIFICACIÓN

Vamos a parametrizar el algoritmo con mejor roc_auc del entrenamiento y evaluación con XGBClassifier con preselección de variables

XGBClassifier()

IMPORTACIÓN DE PAQUETES

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

#Crear datasets de entrenamiento y validación
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier


#Optimizar modelo
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

In [2]:
```
x = pd.read_pickle('../../02_Datos/03_Trabajo/x_preseleccionado.pickle')
y = pd.read_pickle('../../02_Datos/03_Trabajo/y_preseleccionado.pickle')
```

In [3]:
```
#Verificamos predictoras

print(x.shape)
x.head()
```

(70, 6)

Out[3]:

	edad	e_infantil	hr_sentado	frec_alcohol	fumar	estacion
0	0.69	0	0.88	0.8	0	-0.33
1	0.94	1	0.31	0.8	1	-0.33
2	0.50	1	0.50	1.0	-1	-0.33
4	0.67	1	0.50	0.8	-1	-0.33
5	0.67	1	0.50	0.8	0	-0.33

In [4]:
```
#Verificamos target
print(y.shape)
y.head()
```

Out[4]:
```
(70,)
0    0
1    1
2    0
4    1
5    0
Name: produccion, dtype: int64
```

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [5]:
```
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 [6]:
```
pipe = Pipeline([('algoritmo', XGBClassifier())])

grid = [
        {'algoritmo': [XGBClassifier()],
         'algoritmo__n_jobs': [-1],
         'algoritmo__verbosity': [0],#para que no salgan warnings
         'algoritmo__learning_rate': [0.01,0.025,0.05,0.1],
         'algoritmo__max_depth': [5,10,20],
         'algoritmo__reg_alpha': [0,0.1,0.5,1],
         'algoritmo__reg_lambda': [0.01,0.1,1],
         'algoritmo__n_estimators': [100,500,1000]
         }
       ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

In [7]:
```
grid_search = GridSearchCV( estimator= pipe,
                            param_grid= grid,
                            cv = 5,
                            scoring= 'roc_auc',
                            verbose=0,
                            n_jobs= -1
                          )
```

```
modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

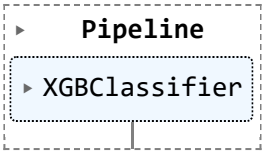
Out[7]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	p
300	0.167594	0.003926	0.008083	0.000207	XGBClassifier(base_score=None, booster=None, c...	
399	0.043036	0.003322	0.008976	0.003989	XGBClassifier(base_score=None, booster=None, c...	
243	0.348920	0.021141	0.007284	0.000866	XGBClassifier(base_score=None, booster=None, c...	
301	0.173299	0.005458	0.007382	0.000492	XGBClassifier(base_score=None, booster=None, c...	
268	0.201251	0.039013	0.023190	0.018829	XGBClassifier(base_score=None, booster=None, c...	
...	...	...	...	...	...	
10	0.046886	0.006373	0.011570	0.004618	XGBClassifier(base_score=None, booster=None, c...	
46	0.042886	0.006246	0.007484	0.001408	XGBClassifier(base_score=None, booster=None, c...	
11	0.052641	0.006459	0.006685	0.000601	XGBClassifier(base_score=None, booster=None, c...	
83	0.045824	0.004612	0.010572	0.004910	XGBClassifier(base_score=None, booster=None, c...	
47	0.050187	0.009385	0.007779	0.000399	XGBClassifier(base_score=None, booster=None, c...	

432 rows × 21 columns

In [8]:
```
modelo.best_estimator_
```

Out[8]:



In [9]:
```
modelo.best_params_
```

Out[9]:
```
{'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                            colsample_bylevel=None, colsample_bynode=None,
                            colsample_bytree=None, early_stopping_rounds=None,
                            enable_categorical=False, eval_metric=None, feature_types=None,
                            gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                            interaction_constraints=None, learning_rate=0.01, max_bin=None,
                            max_cat_threshold=None, max_cat_to_onehot=None,
                            max_delta_step=None, max_depth=5, max_leaves=None,
                            min_child_weight=None, missing=nan, monotone_constraints=None,
                            n_estimators=1000, n_jobs=-1, num_parallel_tree=None,
                            predictor=None, random_state=None, ...),
 'algoritmo__learning_rate': 0.01,
 'algoritmo__max_depth': 5,
 'algoritmo__n_estimators': 1000,
 'algoritmo__n_jobs': -1,
 'algoritmo__reg_alpha': 0,
 'algoritmo__reg_lambda': 0.01,
 'algoritmo__verbosity': 0}
```

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

Out[10]:
```
0.7458333333333333
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [11]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [12]:
```
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

PREDECIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el Train

In [13]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [14]:
```
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: 0.8333333333333333
Accuracy: 0.9591836734693877
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        43
           1       1.00      0.67      0.80         6

    accuracy                           0.96        49
   macro avg       0.98      0.83      0.89        49
weighted avg       0.96      0.96      0.96        49
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

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

Evaluar sobre la validación

In [16]:
```
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}")
```

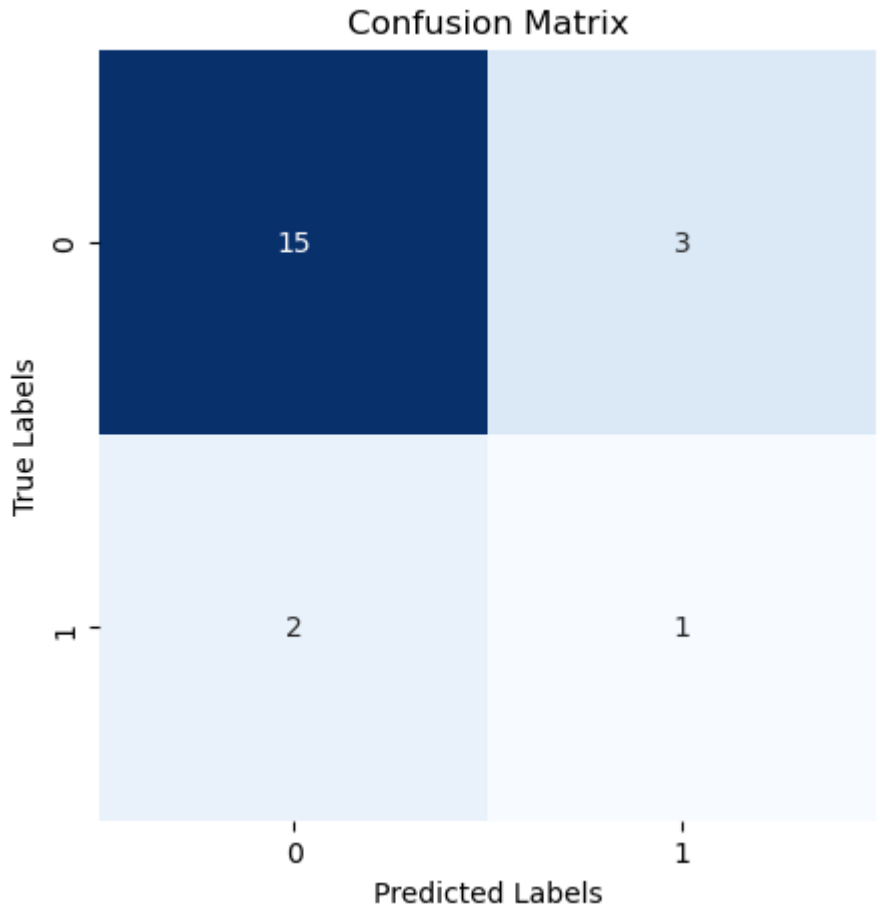
```
Roc AUC_proba: 0.6481481481481481
Roc AUC: 0.5833333333333334
Accuracy: 0.7619047619047619
Classification Report:              precision    recall  f1-score   support

           0       0.88      0.83      0.86        18
           1       0.25      0.33      0.29         3

    accuracy                           0.76        21
   macro avg       0.57      0.58      0.57        21
weighted avg       0.79      0.76      0.78        21
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass

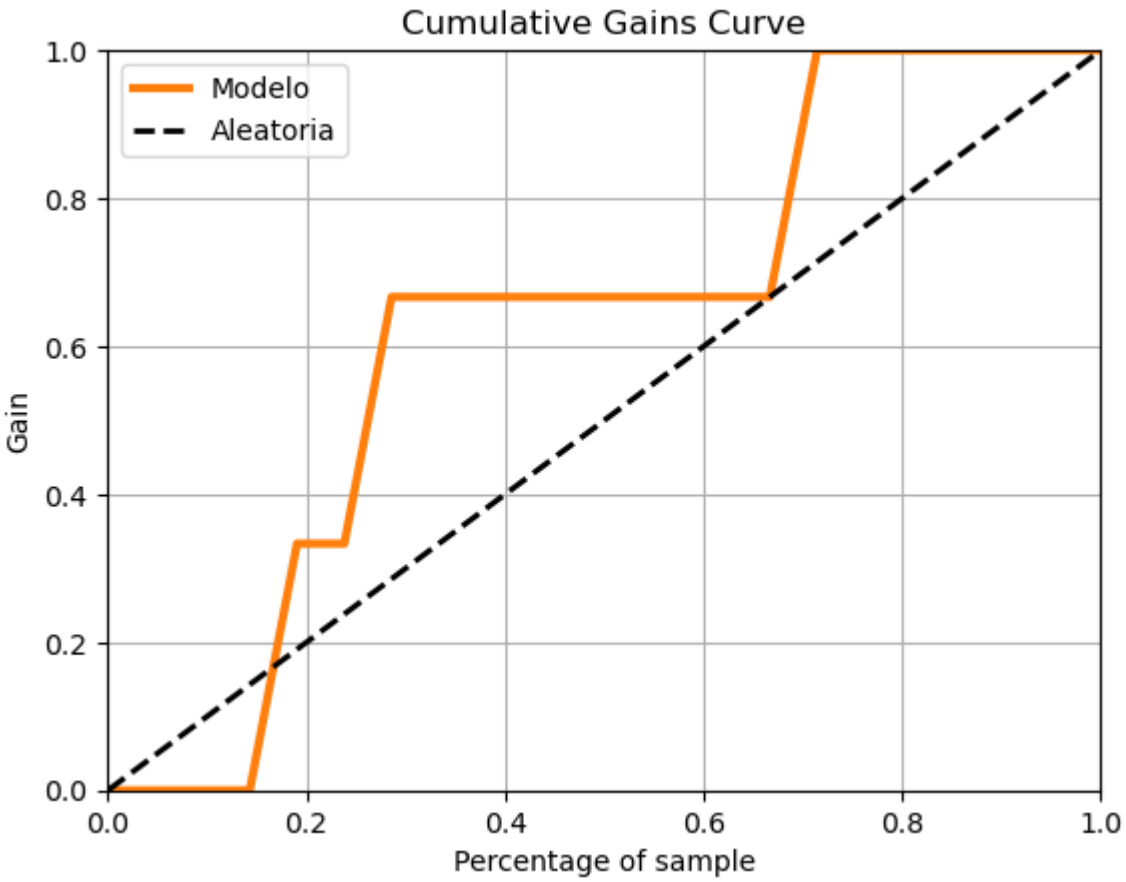
In [17]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)


# 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");
```



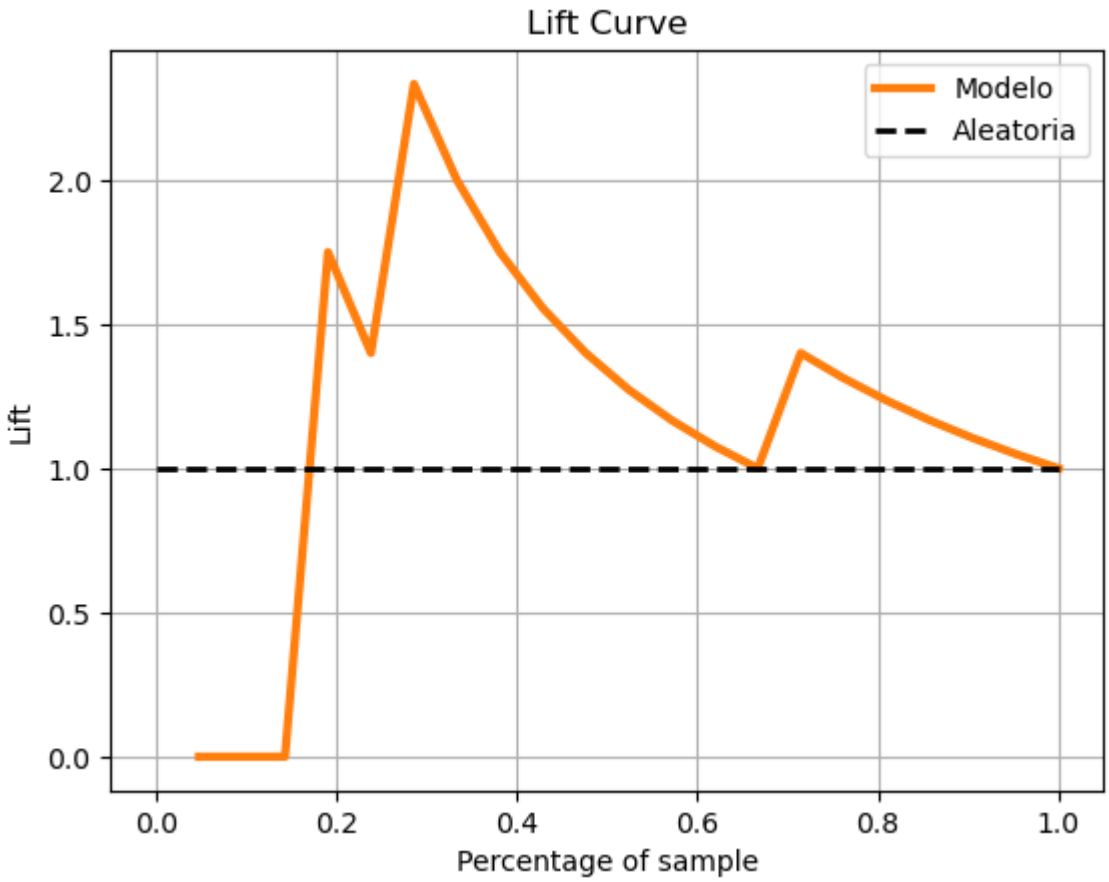
Gain Chart

In [18]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_cumulative_gain(val_y, modelo.best_estimator_.predict_proba(val_

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



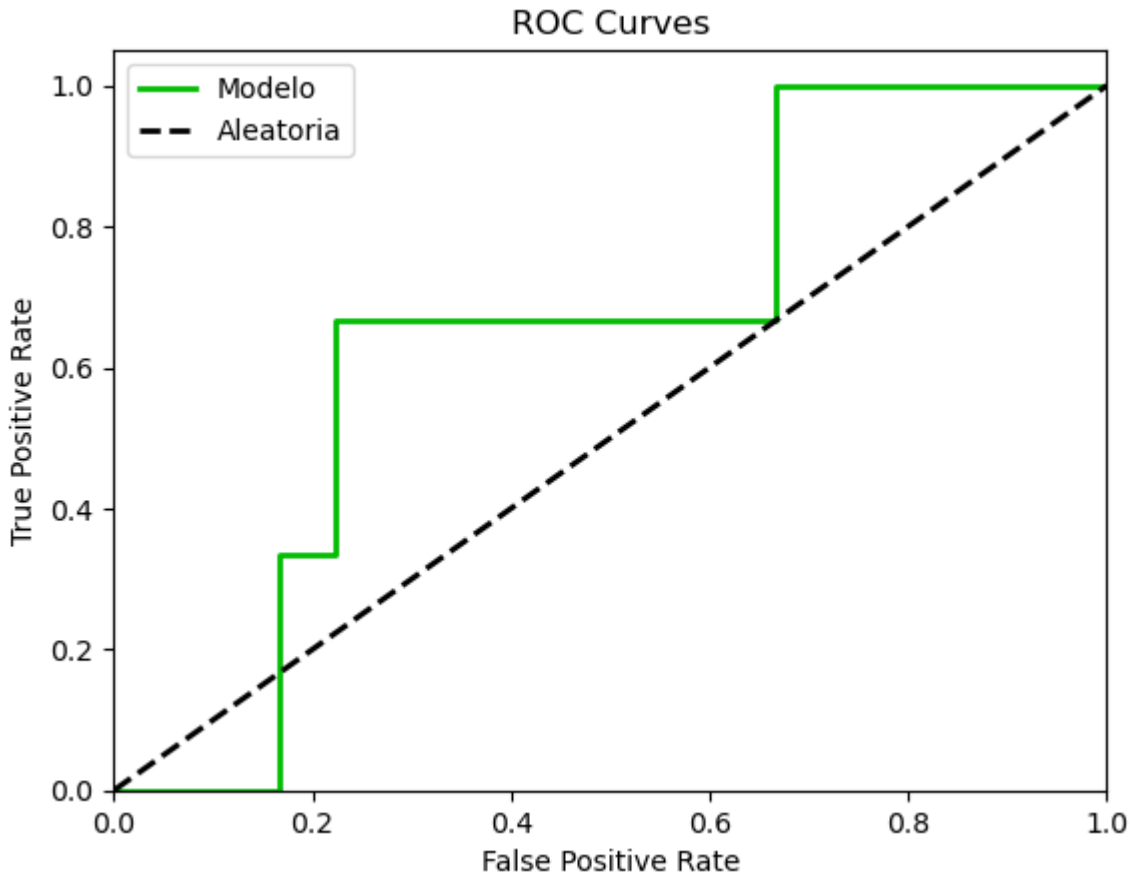
Lift Chart

In [19]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_lift_curve(val_y, modelo.best_estimator_.predict_proba(val_x), a

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



ROC Chart

In [20]:
```
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 [21]:
```
version_estimator = '_v04'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[21]:
```
'XGBClassifier_v04.pickle'
```

In [22]:
```
m_best_estimator
```

Out[22]:
```
'XGBClassifier'
```

In [23]:
```
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 [24]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo XGBClassifier con parámetros para evitar el sobreajuste y pre
x_columns = list(x.columns)
y_target = y.name
```

In [25]:
```
#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[25]:

```
m_Best_estimator                                        XGBClassifier
m_Best_paramans     {'algoritmo': XGBClassifier(base_score=None, b...
m_Best_Score                                      0.7458333333333333
t_accuracy                                                  0.959184
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                             0.648148
v_roc_auc                                                   0.583333
v_accuracy                                                  0.761905
v_report                          precision    recall  f1-score   ...
comentarios         Modelo XGBClassifier con parámetros para evita...
predictoras_X       [edad, e_infantil, hr_sentado, frec_alcohol, f...
target_y                                                    produccion
Name: XGBClassifier_v04.pickle, dtype: object
```

In [26]:

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

In [27]:

```
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 la preselección de variables no ha ayudado a mejorar la predicción del modelo.

06_03 THE BEST ESTIMATOR_V5

MODELAR ALGORITMO DE CLASIFICACIÓN

Vamos a parametrizar el algoritmo con mejor roc_auc del entrenamiento y evaluación con XGBClassifier sin preselección de variables

XGBClassifier()

IMPORTACIÓN DE PAQUETES

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

#Crear datasets de entrenamiento y validación
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier


#Optimizar modelo
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

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

Out[2]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

SEPARAR PREDICTORAS Y TARGET

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

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [4]:
```
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 [5]:
```
pipe = Pipeline([('algoritmo', XGBClassifier())])

grid = [
        {'algoritmo': [XGBClassifier()],
         'algoritmo__n_jobs': [-1],
         'algoritmo__verbosity': [0],#para que no salgan warnings
         'algoritmo__learning_rate': [0.01,0.025,0.05,0.1],
         'algoritmo__max_depth': [5,10,20],
         'algoritmo__reg_alpha': [0,0.1,0.5,1],
         'algoritmo__reg_lambda': [0.01,0.1,1],
         'algoritmo__n_estimators': [100,500,1000]
         }
       ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

In [6]:
```
grid_search = GridSearchCV( estimator= pipe,
                            param_grid= grid,
                            cv = 5,
                            scoring= 'roc_auc',
                            verbose=0,
                            n_jobs= -1
                          )

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

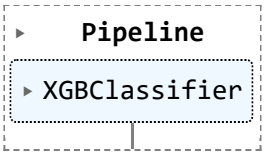
Out[6]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	p
120	0.237701	0.013760	0.011197	0.003916	XGBClassifier(base_score=None, booster=None, c...	
156	0.261889	0.018517	0.007105	0.001355	XGBClassifier(base_score=None, booster=None, c...	
192	0.259459	0.023970	0.017600	0.009332	XGBClassifier(base_score=None, booster=None, c...	
27	0.586696	0.038745	0.018806	0.011289	XGBClassifier(base_score=None, booster=None, c...	
63	0.507730	0.047621	0.012813	0.006408	XGBClassifier(base_score=None, booster=None, c...	
...	...	...	...	...	...	
46	0.051198	0.006045	0.010472	0.003424	XGBClassifier(base_score=None, booster=None, c...	
82	0.049633	0.012800	0.009600	0.003194	XGBClassifier(base_score=None, booster=None, c...	
83	0.052011	0.007164	0.008000	0.000001	XGBClassifier(base_score=None, booster=None, c...	
11	0.056540	0.010962	0.013750	0.007092	XGBClassifier(base_score=None, booster=None, c...	
47	0.050098	0.016609	0.012007	0.007481	XGBClassifier(base_score=None, booster=None, c...	

432 rows × 21 columns

In [7]:
```
modelo.best_estimator_
```

Out[7]:



In [8]:
```
modelo.best_params_
```

Out[8]:
```
{'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                            colsample_bylevel=None, colsample_bynode=None,
                            colsample_bytree=None, early_stopping_rounds=None,
                            enable_categorical=False, eval_metric=None, feature_types=None,
                            gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                            interaction_constraints=None, learning_rate=0.025, max_bin=None,
                            max_cat_threshold=None, max_cat_to_onehot=None,
                            max_delta_step=None, max_depth=5, max_leaves=None,
                            min_child_weight=None, missing=nan, monotone_constraints=None,
                            n_estimators=500, n_jobs=-1, num_parallel_tree=None,
                            predictor=None, random_state=None, ...),
 'algoritmo__learning_rate': 0.025,
 'algoritmo__max_depth': 5,
 'algoritmo__n_estimators': 500,
 'algoritmo__n_jobs': -1,
 'algoritmo__reg_alpha': 0,
 'algoritmo__reg_lambda': 0.01,
 'algoritmo__verbosity': 0}
```

In [9]:
```
modelo.best_score_
```

Out[9]:
```
0.825
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [10]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [11]:
```
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

PREDECIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el Train

In [12]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [13]:
```
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: 0.8333333333333333
Accuracy: 0.9591836734693877
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        43
           1       1.00      0.67      0.80         6

    accuracy                           0.96        49
   macro avg       0.98      0.83      0.89        49
weighted avg       0.96      0.96      0.96        49
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

In [14]:
```
pred = modelo.best_estimator_.predict(val_x)
pred_proba = modelo.best_estimator_.predict_proba(val_x)[:,1]
```

Evaluar sobre la validación

In [15]:
```
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}")
```

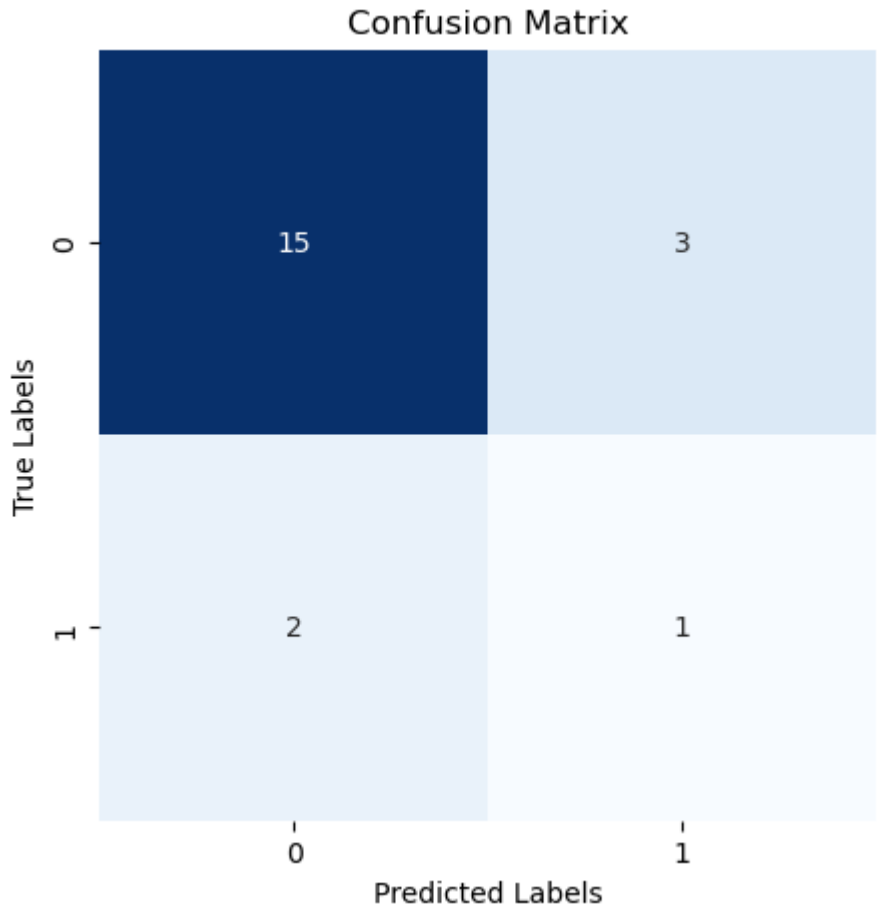
```
Roc AUC_proba: 0.7407407407407408
Roc AUC: 0.5833333333333334
Accuracy: 0.7619047619047619
Classification Report:              precision    recall  f1-score   support

           0       0.88      0.83      0.86        18
           1       0.25      0.33      0.29         3

    accuracy                           0.76        21
   macro avg       0.57      0.58      0.57        21
weighted avg       0.79      0.76      0.78        21
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass

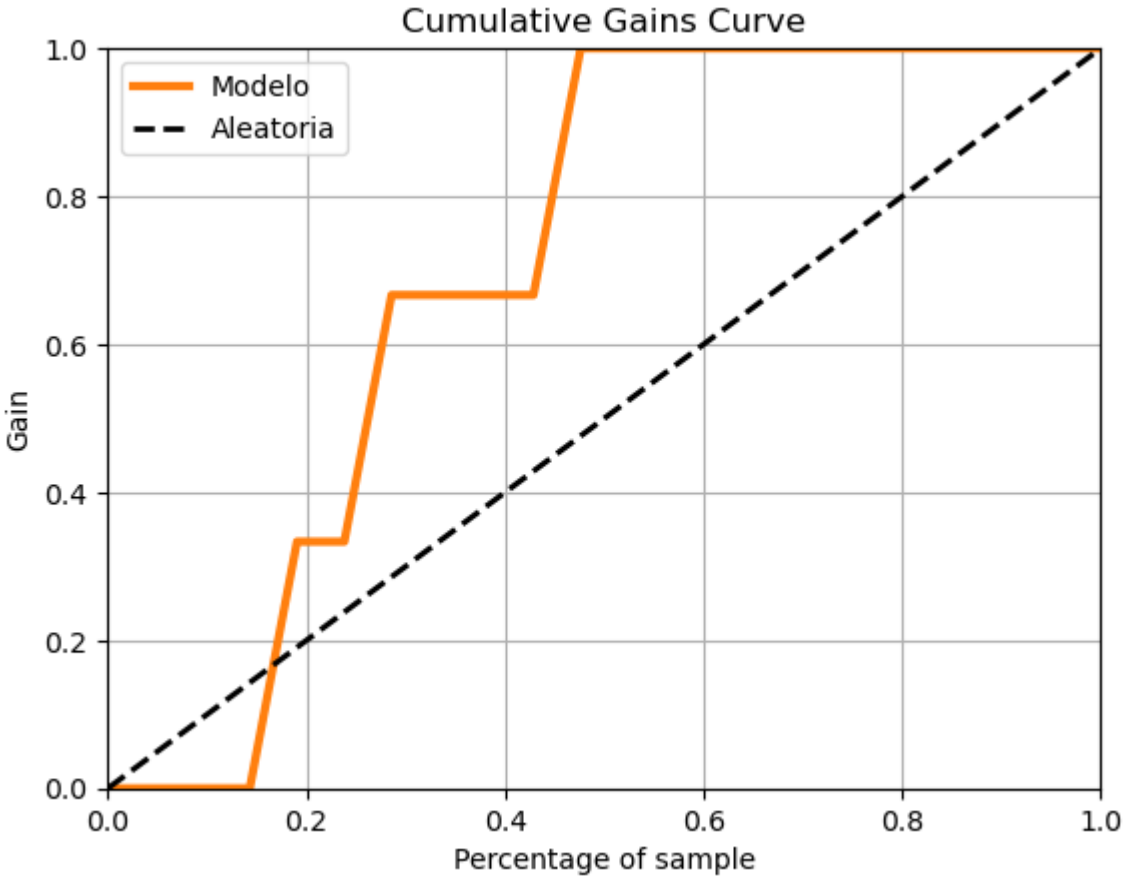
In [16]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)


# 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");
```



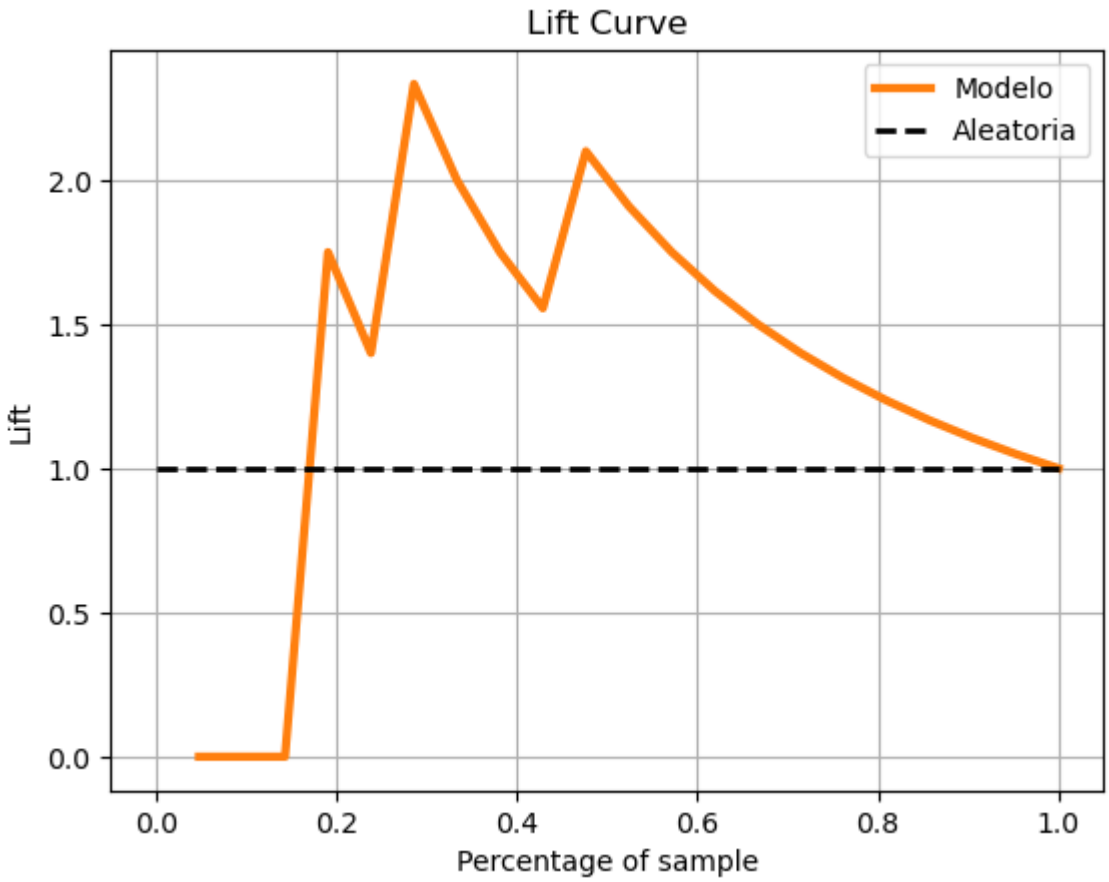
Gain Chart

In [17]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_cumulative_gain(val_y, modelo.best_estimator_.predict_proba(val_

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



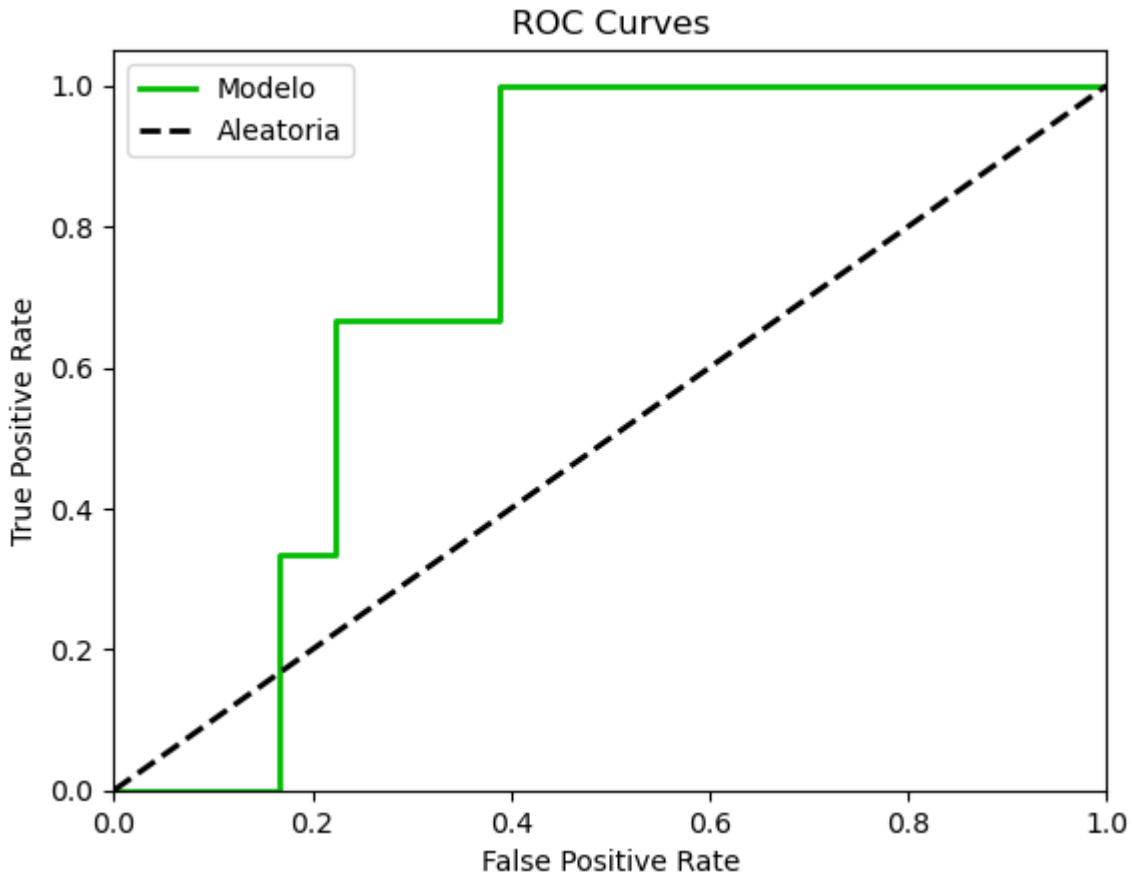
Lift Chart

In [18]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_lift_curve(val_y, modelo.best_estimator_.predict_proba(val_x), a

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



ROC Chart

In [19]:
```
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 [20]:
```
version_estimator = '_v05'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[20]:
```
'XGBClassifier_v05.pickle'
```

In [21]:
```
m_best_estimator
```

Out[21]:
```
'XGBClassifier'
```

In [22]:
```
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 [23]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo XGBClassifier con parámetros para evitar el sobreajuste sin p
x_columns = list(x.columns)ht
y_target = y.name
```

In [24]:
```
#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[24]:

```
m_Best_estimator                                          XGBClassifier
m_Best_paramans     {'algoritmo': XGBClassifier(base_score=None, b...
m_Best_Score                                                     0.825
t_accuracy                                                    0.959184
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                               0.740741
v_roc_auc                                                     0.583333
v_accuracy                                                    0.761905
v_report                          precision    recall  f1-score   ...
comentarios         Modelo XGBClassifier con parámetros para evita...
predictoras_X       [estacion, edad, e_infantil, acc_grave, int_qu...
target_y                                                     produccion
Name: XGBClassifier_v05.pickle, dtype: object
```

In [25]:

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

In [26]:

```
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 la preselección de variables no ha ayudado a mejorar la predicción del modelo.

06_04 THE BEST ESTIMATOR_V5_1

MODELAR ALGORITMO DE CLASIFICACIÓN

Vamos a parametrizar el algoritmo con mejor roc_auc del entrenamiento y evaluación con XGBClassifier sin preselección de variables. Añadimos el parámetro **scale_pos_weight**

XGBClassifier()

IMPORTACIÓN DE PAQUETES

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

#Crear datasets de entrenamiento y validación
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier


#Optimizar modelo
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
import scikitplot as skplt

#Crear Pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

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

Out[2]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

SEPARAR PREDICTORAS Y TARGET

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

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [4]:
```
train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_stat
```

DETERMINAR EL PESO DE LA CLASE DESBALANCEADA.

In [5]:
```
# Supongamos que tienes X_train y y_train como datos de entrenamiento
# Calcula el valor de scale_pos_weight en base a la proporción de clases
proporcion_clase_positiva = sum(y == 1) / len(y)
scale_pos_weight = 1 / proporcion_clase_positiva
```

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

In [6]:
```
pipe = Pipeline([('algoritmo', XGBClassifier())])

grid = [
        {'algoritmo': [XGBClassifier(scale_pos_weight = scale_pos_weight)],
         'algoritmo__n_jobs': [-1],
         'algoritmo__verbosity': [0],#para que no salgan warnings
         'algoritmo__learning_rate': [0.01,0.025,0.05,0.1],
         'algoritmo__max_depth': [5,10,20],
         'algoritmo__reg_alpha': [0,0.1,0.5,1],
         'algoritmo__reg_lambda': [0.01,0.1,1],
         'algoritmo__n_estimators': [100,500,1000]
         }
       ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

In [7]:
```
grid_search = GridSearchCV( estimator= pipe,
                            param_grid= grid,
                            cv = 5,
                            scoring= 'roc_auc',
```

```
                           verbose=0,
                           n_jobs= -1
                          )

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

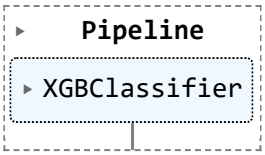
Out[7]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	p
385	0.373514	0.021536	0.004851	3.961762e-03	XGBClassifier(base_score=None, booster=None, c...	
349	0.325141	0.009660	0.006401	3.200269e-03	XGBClassifier(base_score=None, booster=None, c...	
421	0.333445	0.004019	0.006401	3.200723e-03	XGBClassifier(base_score=None, booster=None, c...	
36	0.049828	0.003250	0.008036	6.721787e-05	XGBClassifier(base_score=None, booster=None, c...	
72	0.044809	0.003932	0.006399	3.199459e-03	XGBClassifier(base_score=None, booster=None, c...	
...	...	...	...	...	...	
152	0.045463	0.003753	0.004862	3.482591e-03	XGBClassifier(base_score=None, booster=None, c...	
81	0.048012	0.000015	0.009604	3.198125e-03	XGBClassifier(base_score=None, booster=None, c...	
146	0.048199	0.005065	0.008003	4.016769e-06	XGBClassifier(base_score=None, booster=None, c...	
116	0.046414	0.003208	0.008000	8.259062e-07	XGBClassifier(base_score=None, booster=None, c...	
110	0.044084	0.006102	0.009619	1.631343e-03	XGBClassifier(base_score=None, booster=None, c...	

432 rows × 21 columns

In [8]:
```
modelo.best_estimator_
```

Out[8]:



In [9]:
```
modelo.best_params_
```

Out[9]:
```
{'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                            colsample_bylevel=None, colsample_bynode=None,
                            colsample_bytree=None, early_stopping_rounds=None,
                            enable_categorical=False, eval_metric=None, feature_types=None,
                            gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                            interaction_constraints=None, learning_rate=0.1, max_bin=None,
                            max_cat_threshold=None, max_cat_to_onehot=None,
                            max_delta_step=None, max_depth=5, max_leaves=None,
                            min_child_weight=None, missing=nan, monotone_constraints=None,
                            n_estimators=1000, n_jobs=-1, num_parallel_tree=None,
                            predictor=None, random_state=None, ...),
 'algoritmo__learning_rate': 0.1,
 'algoritmo__max_depth': 5,
 'algoritmo__n_estimators': 1000,
 'algoritmo__n_jobs': -1,
 'algoritmo__reg_alpha': 0,
 'algoritmo__reg_lambda': 0.1,
 'algoritmo__verbosity': 0}
```

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

Out[10]:
```
0.7555555555555555
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [11]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [12]:
```
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

PREDECIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el Train

In [13]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [14]:
```
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        43
           1       1.00      1.00      1.00         6

    accuracy                           1.00        49
   macro avg       1.00      1.00      1.00        49
weighted avg       1.00      1.00      1.00        49
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

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

Evaluar sobre la validación

In [16]:
```
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}")
```

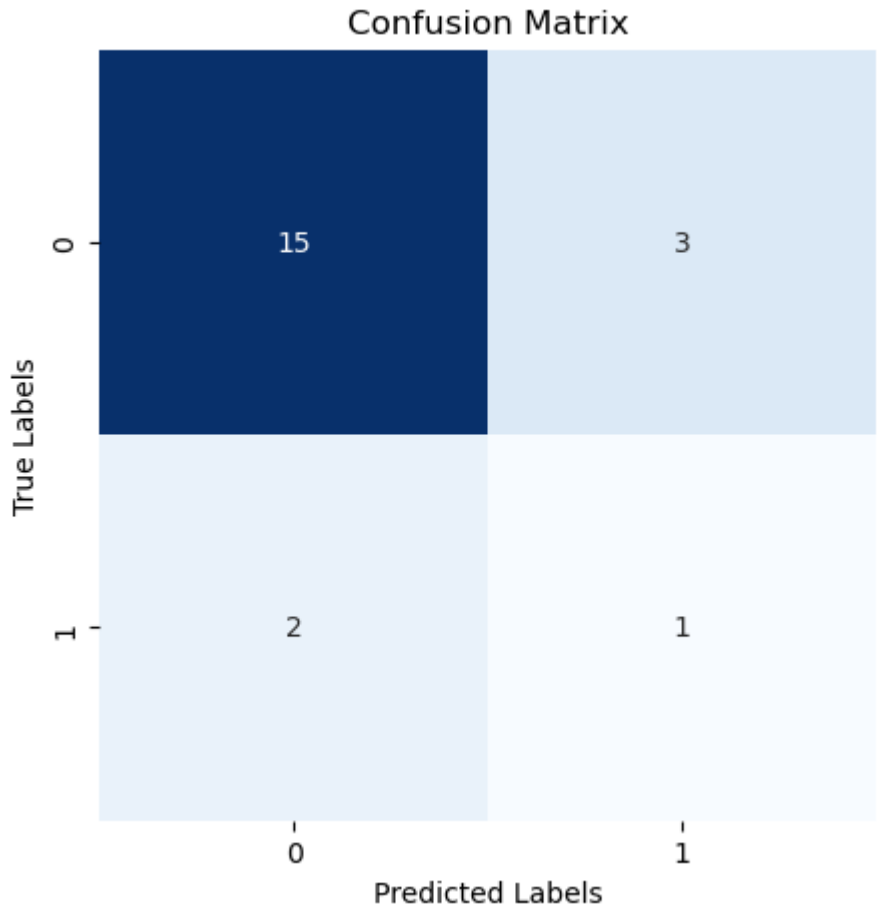
```
Roc AUC_proba: 0.6851851851851852
Roc AUC: 0.5833333333333334
Accuracy: 0.7619047619047619
Classification Report:              precision    recall  f1-score   support

           0       0.88      0.83      0.86        18
           1       0.25      0.33      0.29         3

    accuracy                           0.76        21
   macro avg       0.57      0.58      0.57        21
weighted avg       0.79      0.76      0.78        21
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass

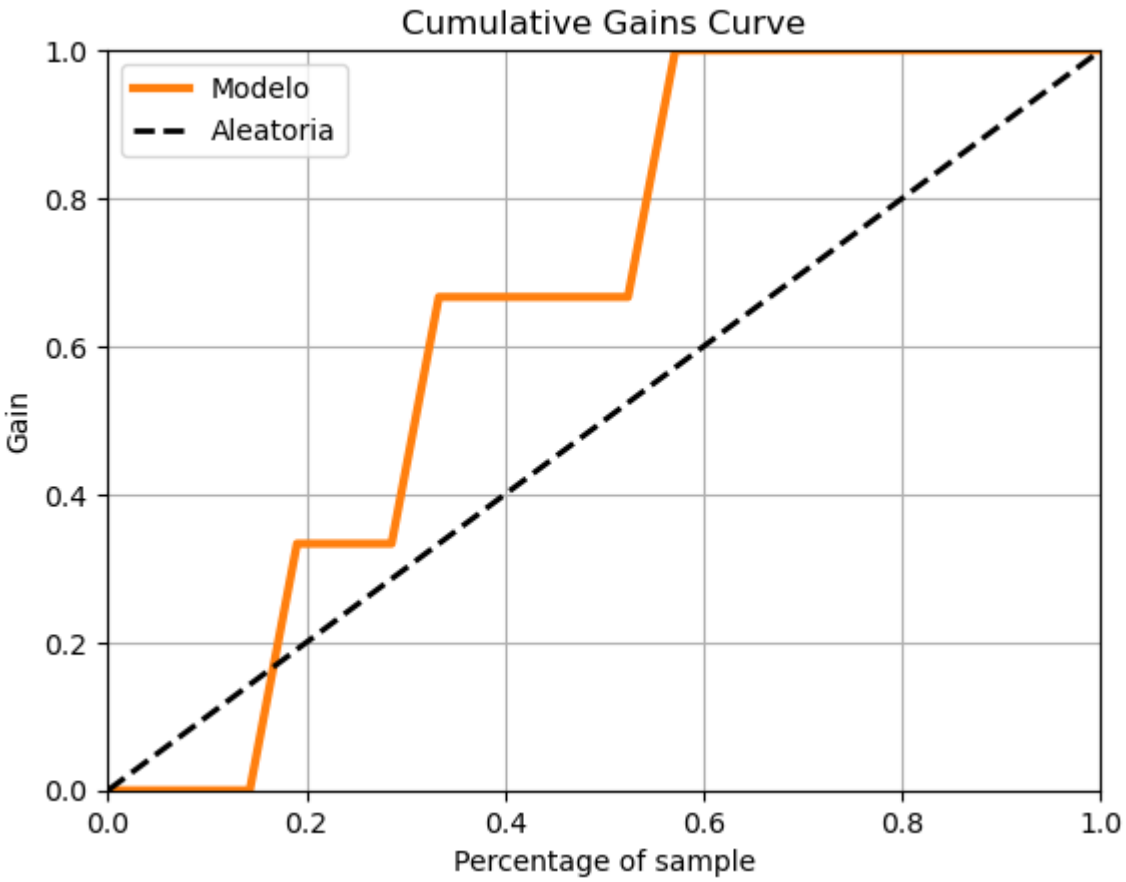
In [17]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)


# 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");
```



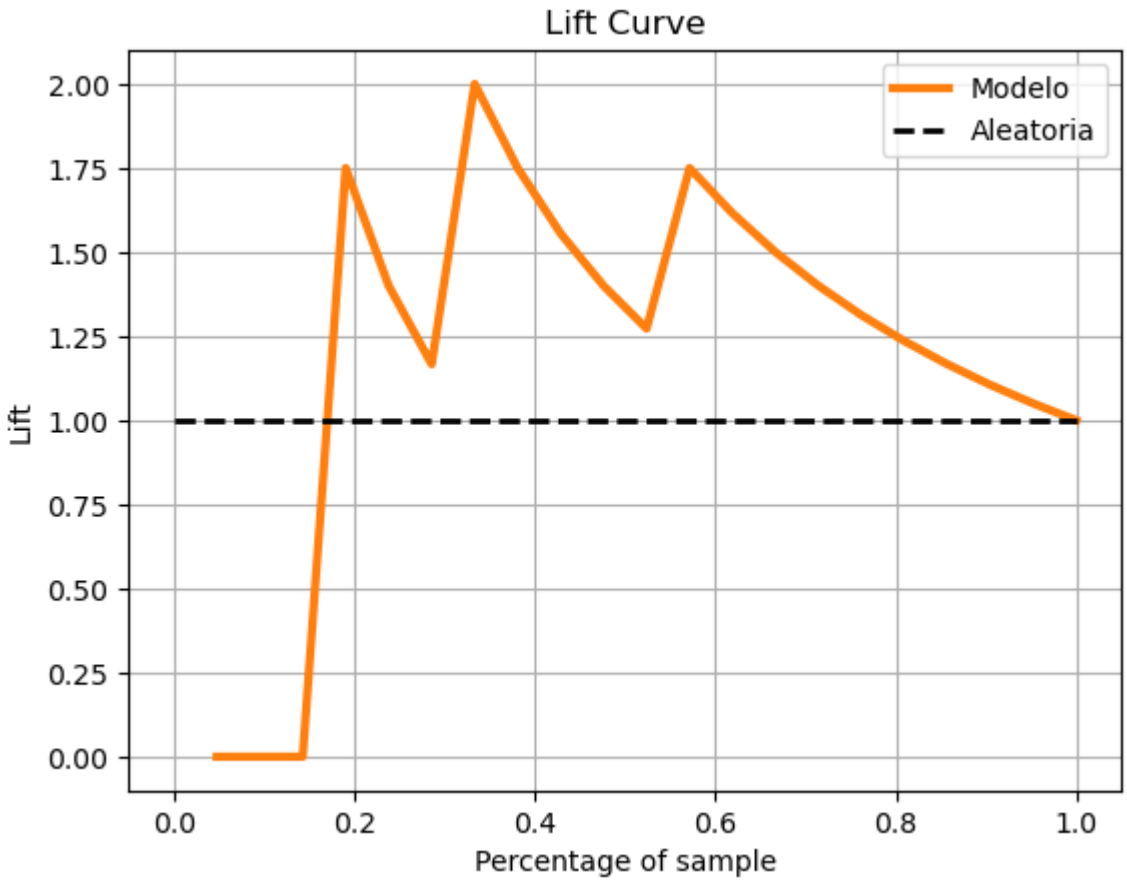
Gain Chart

In [18]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_cumulative_gain(val_y, modelo.best_estimator_.predict_proba(val_

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



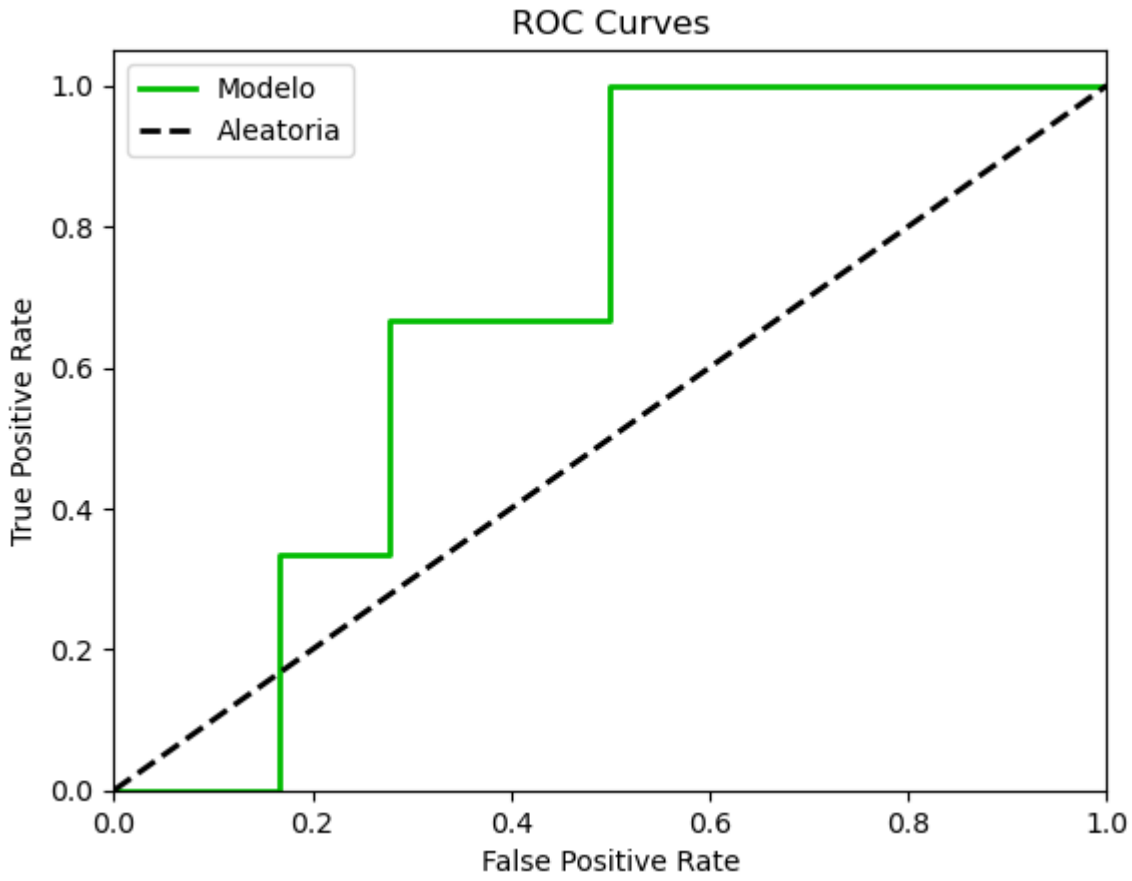
Lift Chart

In [19]:
```
fig, ax = plt.subplots()

skplt.metrics.plot_lift_curve(val_y, modelo.best_estimator_.predict_proba(val_x), a

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



ROC Chart

In [20]:
```
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 [21]:
```
version_estimator = '_v05_1'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[21]:
```
'XGBClassifier_v05_1.pickle'
```

In [22]:
```
m_best_estimator
```

Out[22]:
```
'XGBClassifier'
```

In [23]:
```
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 [24]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo XGBClassifier con parámetros para evitar el sobreajuste sin p
x_columns = list(x.columns)
y_target = y.name
```

In [25]:
```
#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[25]:

```
m_Best_estimator                                        XGBClassifier
m_Best_paramans     {'algoritmo': XGBClassifier(base_score=None, b...
m_Best_Score                                        0.7555555555555555
t_accuracy                                                         1.0
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                                0.685185
v_roc_auc                                                      0.583333
v_accuracy                                                     0.761905
v_report                          precision    recall  f1-score   ...
comentarios         Modelo XGBClassifier con parámetros para evita...
predictoras_X       [estacion, edad, e_infantil, acc_grave, int_qu...
target_y                                                     produccion
Name: XGBClassifier_v05_1.pickle, dtype: object
```

In [26]:

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

In [27]:

```
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 la scale-class-weight no ha ayudado a mejorar la predicción del modelo.

PRÓXIMOS PASOS: Trabajaremos con el The Best_Estimator V5 (XGBClassifier) sin preselección de variables con mejores valores con mejores resultados con AUC[1] del (74%). Con un 74% sería un modelo útil. Intentaremos mejorar el modelo realizando un balanceo de datos con los siguientes métodos:

- Sin balanceo
- Undersampling
- Oversampling
- Smote-tomek

07 - BALANCEO DE DATOS

Vamos a realizar los siguientes métodos:

- Sin balanceo
- Undersampling
- Oversampling
- Smote-tomek

IMPORTAR PAQUETES

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

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import TomekLinks
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTETomek

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

IMPORTAR LOS DATOS

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

Out[2]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

Cargar los datos

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

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

Out[4]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_sen
0	-0.33	0.69	0	1	1	0	0.8	0	
1	-0.33	0.94	1	0	1	0	0.8	1	
2	-0.33	0.50	1	0	0	0	1.0	-1	
4	-0.33	0.67	1	1	0	0	0.8	-1	
5	-0.33	0.67	1	0	1	0	0.8	0	

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

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

BALANCEO

SIN BALANCEO

Crear train y test

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

Instanciar modelo

In [7]:
```
rl_sin_balanceo = LogisticRegression(n_jobs= -1)
```

Entrenar

In [8]:
```
rl_sin_balanceo.fit(train_x, train_y)
```

Out[8]:
```
LogisticRegression
LogisticRegression(n_jobs=-1)
```

Aplicar

In [9]:
```
pred_rl_sin_balanceo = rl_sin_balanceo.predict_proba(val_x)[:,1]
pred = rl_sin_balanceo.predict(val_x)
```

Evaluar

In [10]:
```
roc1_rl_sin_balanceo = roc_auc_score(val_y, pred_rl_sin_balanceo)
roc_rl_sin_balanceo = roc_auc_score(val_y, pred)
roc_rl_sin_balanceo_CR = (classification_report(val_y, pred))
roc_rl_sin_balanceo_CF = (confusion_matrix(val_y, pred))

print(f'AUC [1]: {roc1_rl_sin_balanceo}')
print(f'AUC: {roc_rl_sin_balanceo}')
```

```
print(roc_rl_sin_balanceo_CR)
print(roc_rl_sin_balanceo_CF)
```

```
AUC [1]: 0.48148148148148145
AUC: 0.5
              precision    recall  f1-score   support

           0       0.86      1.00      0.92        18
           1       0.00      0.00      0.00         3

    accuracy                           0.86        21
   macro avg       0.43      0.50      0.46        21
weighted avg       0.73      0.86      0.79        21

[[18  0]
 [ 3  0]]
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classif
ication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classif
ication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classif
ication.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and
being set to 0.0 in labels with no predicted samples. Use `zero_division` paramete
r to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

UNDERSAMPLING

Instanciar el undersampler

In [11]:
```
rus = RandomUnderSampler(sampling_strategy= 1)
```

Entrenar y aplicar el undersampler

In [12]:
```
x_rus, y_rus = rus.fit_resample(x,y)
```

Crear train y test

In [13]:
```
train_x_rus, val_x_rus, train_y_rus, val_y_rus = train_test_split(x_rus, y_rus, te
```

Instanciar el modelo

In [14]:
```
rl_rus = LogisticRegression(n_jobs=-1)
```

Entrenar

In [15]:
```
rl_rus.fit(train_x_rus,train_y_rus)
```

Out[15]:
```
LogisticRegression
LogisticRegression(n_jobs=-1)
```

Aplicar

In [16]:
```
pred_rl_rus = rl_rus.predict_proba(val_x_rus)[:,1]
pred_rl_rusCR = rl_rus.predict(val_x_rus)
```

Evaluar

In [17]:
```
roc1_rl_rus = roc_auc_score(val_y_rus,pred_rl_rus)
roc_rl_rus = roc_auc_score(val_y_rus, pred_rl_rusCR)
roc_rl_rus_CR = classification_report(val_y_rus, pred_rl_rusCR)
roc_rl_rus_CF = confusion_matrix(val_y_rus, pred_rl_rusCR)

print(f'AUC [1]: {roc1_rl_rus}')
print(f'AUC: {roc_rl_rus}')
print(roc_rl_rus_CR)
print(roc_rl_rus_CF)
```

```
AUC [1]: 0.8
AUC: 0.7
              precision    recall  f1-score   support

           0       1.00      0.40      0.57         5
           1       0.25      1.00      0.40         1

    accuracy                           0.50         6
   macro avg       0.62      0.70      0.49         6
weighted avg       0.88      0.50      0.54         6

[[2 3]
 [0 1]]
```

OVERSAMPLING

Instanciar el Oversampler

In [18]:
```
ros = RandomOverSampler(sampling_strategy=1)
```

Entrenar y aplicar el Oversampler

In [19]:
```
x_ros, y_ros = ros.fit_resample(x,y)
```

Crear train y test

In [20]:
```
train_x_ros, val_x_ros, train_y_ros, val_y_ros = train_test_split(x_ros, y_ros, tes
```

Instanciar el modelo

In [21]:
```
rl_ros = LogisticRegression(n_jobs= -1)
```

Entrenar

In [22]:
```
rl_ros.fit(train_x_ros, train_y_ros)
```

Out[22]:
```
LogisticRegression
LogisticRegression(n_jobs=-1)
```

Aplicar

In [23]:
```
pred_rl_ros = rl_ros.predict_proba(val_x_ros)[:,1]
pred_rl_rosCR = rl_ros.predict(val_x_ros)
```

Evaluar

In [24]:
```
roc1_rl_ros = roc_auc_score(val_y_ros,pred_rl_ros)
roc_rl_ros = roc_auc_score(val_y_ros, pred_rl_rosCR)
roc_rl_ros_CR = classification_report(val_y_ros, pred_rl_rosCR)
roc_rl_ros_CF = confusion_matrix(val_y_ros, pred_rl_rosCR)

print(f'AUC [1]: {roc1_rl_ros}')
print(f'AUC: {roc_rl_ros}')
print(roc_rl_ros_CR)
print(roc_rl_ros_CF)
```

```
AUC [1]: 0.7690058479532164
AUC: 0.6198830409356725
              precision    recall  f1-score   support

           0       0.62      0.56      0.59        18
           1       0.62      0.68      0.65        19

    accuracy                           0.62        37
   macro avg       0.62      0.62      0.62        37
weighted avg       0.62      0.62      0.62        37

[[10  8]
 [ 6 13]]
```

SMOTE-TOMEK

Instanciar un Tomek y un SMOTE

In [25]:
```
tom = TomekLinks(n_jobs= -1)
smo = SMOTE(sampling_strategy= 1, n_jobs= -1)
```

Instanciar el SMOTE-Tomek

In [26]:
```
sto = SMOTETomek(sampling_strategy= 1,
                 smote= smo,
                 tomek= tom,
                 n_jobs= -1)
```

Entrenar y aplicar el SMOTE-Tomek

In [27]:
```
x_sto, y_sto = sto.fit_resample(x, y)
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\imblearn\over_sampling\_smote\base.py:336: FutureWarning: The parameter `n_jobs` has been deprecated in 0.10 and will be removed in 0.12. You can pass an nearest neighbors estimator where `n_jobs` is already set instead.
  warnings.warn(
```

Crear train y test el SMOTE-Tomek

In [28]:
```
train_x_sto, val_x_sto, train_y_sto, val_y_sto = train_test_split(x_sto, y_sto, tes
```

Instanciar el modelo

In [29]:
```
rl_sto = LogisticRegression(n_jobs=-1)
```

Entrenar

In [30]:
```
rl_sto.fit(train_x_sto, train_y_sto)
```

Out[30]:
```
LogisticRegression
LogisticRegression(n_jobs=-1)
```

Aplicar

In [31]:
```
pred_rl_sto = rl_sto.predict_proba(val_x_sto)[:,1]
pred_rl_stoCR = rl_sto.predict(val_x_sto)
```

Evaluar

In [32]:
```
roc1_rl_sto = roc_auc_score(val_y_sto,pred_rl_sto)
roc_rl_sto = roc_auc_score(val_y_sto, pred_rl_stoCR)
roc_rl_sto_CR = classification_report(val_y_sto, pred_rl_stoCR)
roc_rl_sto_CF = confusion_matrix(val_y_sto, pred_rl_stoCR)

print(f'AUC [1]: {roc1_rl_sto}')
print(f'AUC: {roc_rl_sto}')
print(roc_rl_sto_CR)
print(roc_rl_sto_CF)
```

```
AUC [1]: 0.8450292397660819
AUC: 0.8377192982456141
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        18
           1       0.84      0.84      0.84        19

    accuracy                           0.84        37
   macro avg       0.84      0.84      0.84        37
weighted avg       0.84      0.84      0.84        37

[[15  3]
 [ 3 16]]
```

EVALUAR RESULTADOS

In [33]:
```
print('***Sin Balanceo***\n')
print(f'Sin Balanceo - AUC[1]: {roc1_rl_sin_balanceo}')
```

```
print(f'Sin Balanceo - AUC: {roc_rl_sin_balanceo}')
print(roc_rl_sin_balanceo_CR)
print('Matrix Confusión:\n' ,roc_rl_sin_balanceo_CF)

print('\n***Undersampling***\n')
print(f'Undersampling - AUC[1]: {roc1_rl_rus}')
print(f'Undersampling - AUC: {roc_rl_rus}')
print(roc_rl_rus_CR)
print('Matrix Confusión:\n' ,roc_rl_rus_CF)

print('\n***Oversampling***\n')
print(f'Oversampling - AUC[1]: {roc1_rl_ros}')
print(f'Oversampling - AUC: {roc_rl_ros}')
print(roc_rl_ros_CR)
print('Matrix Confusión:\n' ,roc_rl_ros_CF)

print('\n***Smote-Tomek***\n')
print(f'Smote-Tomek - AUC[1]: {roc1_rl_sto}')
print(f'Smote-Tomek - AUC: {roc_rl_sto}')
print(roc_rl_sto_CR)
print('Matrix Confusión:\n' ,roc_rl_sto_CF)
```

```
***Sin Balanceo***

Sin Balanceo - AUC[1]: 0.48148148148148145
Sin Balanceo - AUC: 0.5
              precision    recall  f1-score   support

           0       0.86      1.00      0.92        18
           1       0.00      0.00      0.00         3

    accuracy                           0.86        21
   macro avg       0.43      0.50      0.46        21
weighted avg       0.73      0.86      0.79        21

Matrix Confusión:
 [[18  0]
 [ 3  0]]

***Undersampling***

Undersampling - AUC[1]: 0.8
Undersampling - AUC: 0.7
              precision    recall  f1-score   support

           0       1.00      0.40      0.57         5
           1       0.25      1.00      0.40         1

    accuracy                           0.50         6
   macro avg       0.62      0.70      0.49         6
weighted avg       0.88      0.50      0.54         6

Matrix Confusión:
 [[2 3]
 [0 1]]

***Oversampling***

Oversampling - AUC[1]: 0.7690058479532164
Oversampling - AUC: 0.6198830409356725
              precision    recall  f1-score   support

           0       0.62      0.56      0.59        18
           1       0.62      0.68      0.65        19

    accuracy                           0.62        37
   macro avg       0.62      0.62      0.62        37
weighted avg       0.62      0.62      0.62        37

Matrix Confusión:
 [[10  8]
 [ 6 13]]

***Smote-Tomek***

Smote-Tomek - AUC[1]: 0.8450292397660819
Smote-Tomek - AUC: 0.8377192982456141
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        18
           1       0.84      0.84      0.84        19

    accuracy                           0.84        37
   macro avg       0.84      0.84      0.84        37
weighted avg       0.84      0.84      0.84        37
```

```
Matrix Confusión:
 [[15  3]
 [ 3 16]]
```

CONCLUSIÓN: Podemos establecer que el modelo trabaja mejor aplicando un balanceo. El que mejor resultados a obtenido ha sido el Smote-Tomek. Por lo que aplicaremos el dataset con el Smote-Tomek al XGBClassifier

GUARDAR DATASET TRAS PRESELECCIÓN

In [34]:
```
# Balanceo seleccionado

x_final = x_sto
y_final = y_sto
# Definir los nombres de los archivos

x_final.to_pickle('../../02_Datos/03_Trabajo/x_balanceo.pickle')
y_final.to_pickle('../../02_Datos/03_Trabajo/y_balanceo.pickle')
```

08_01 THE BEST ESTIMATOR_V6

MODELAR ALGORITMO DE CLASIFICACIÓN

Vamos a parametrizar el algoritmo con mejor roc_auc del entrenamiento y evaluación con XGBClassifier sin preselección de variables y con balanceo de datos.

XGBClassifier()

IMPORTACIÓN DE PAQUETES

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

#Crear datasets de entrenamiento y validación
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier


#Optimizar modelo
from sklearn.model_selection import GridSearchCV

#Métricas de evaluación
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
from sklearn.metrics import PrecisionRecallDisplay, precision_recall_curve
from yellowbrick.classifier import discrimination_threshold, DiscriminationThreshol
import scikitplot as skplt



#Crear Pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

In [2]:
```
x = pd.read_pickle('../../02_Datos/03_Trabajo/x_balanceo.pickle')
y = pd.read_pickle('../../02_Datos/03_Trabajo/y_balanceo.pickle')
```

MODELIZAR

RESERVAR LOS DATASET DE ENTRENAMIENTO Y VALIDACIÓN

In [3]:
```
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 [4]:
```
pipe = Pipeline([('algoritmo', XGBClassifier())])

grid = [
        {'algoritmo': [XGBClassifier()],
         'algoritmo__n_jobs': [-1],
         'algoritmo__verbosity': [0],#para que no salgan warnings
         'algoritmo__learning_rate': [0.01,0.025,0.05,0.1],
         'algoritmo__max_depth': [5,10,20],
         'algoritmo__reg_alpha': [0,0.1,0.5,1],
         'algoritmo__reg_lambda': [0.01,0.1,1],
         'algoritmo__n_estimators': [100,500,1000]
         }
       ]
```

OPTIMIZAR LOS HIPERPARÁMETROS

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

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

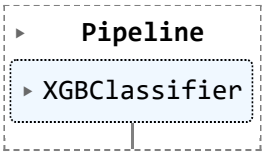
Out[5]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	p
413	0.262402	0.015423	0.013541	0.005506	XGBClassifier(base_score=None, booster=None, c...	
389	0.505076	0.057981	0.008377	0.001492	XGBClassifier(base_score=None, booster=None, c...	
341	0.240600	0.021598	0.008379	0.001198	XGBClassifier(base_score=None, booster=None, c...	
374	0.247004	0.034991	0.008269	0.001448	XGBClassifier(base_score=None, booster=None, c...	
386	0.408500	0.024842	0.007581	0.000490	XGBClassifier(base_score=None, booster=None, c...	
...	...	...	...	...	...	
4	0.032965	0.002403	0.004890	0.000659	XGBClassifier(base_score=None, booster=None, c...	
3	0.037539	0.009166	0.005784	0.000976	XGBClassifier(base_score=None, booster=None, c...	
1	0.036348	0.003689	0.006986	0.003524	XGBClassifier(base_score=None, booster=None, c...	
40	0.072529	0.010924	0.009104	0.001779	XGBClassifier(base_score=None, booster=None, c...	
76	0.079564	0.006454	0.007753	0.000706	XGBClassifier(base_score=None, booster=None, c...	

432 rows × 21 columns

In [6]:
```
modelo.best_estimator_
```

Out[6]:



In [7]:
```
modelo.best_params_
```

Out[7]:
```
{'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                            colsample_bylevel=None, colsample_bynode=None,
                            colsample_bytree=None, early_stopping_rounds=None,
                            enable_categorical=False, eval_metric=None, feature_types=None,
                            gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                            interaction_constraints=None, learning_rate=0.05, max_bin=None,
                            max_cat_threshold=None, max_cat_to_onehot=None,
                            max_delta_step=None, max_depth=5, max_leaves=None,
                            min_child_weight=None, missing=nan, monotone_constraints=None,
                            n_estimators=500, n_jobs=-1, num_parallel_tree=None,
                            predictor=None, random_state=None, ...),
 'algoritmo__learning_rate': 0.05,
 'algoritmo__max_depth': 5,
 'algoritmo__n_estimators': 500,
 'algoritmo__n_jobs': -1,
 'algoritmo__reg_alpha': 0.1,
 'algoritmo__reg_lambda': 0.1,
 'algoritmo__verbosity': 0}
```

In [8]:
```
modelo.best_score_
```

Out[8]:
```
0.9847222222222222
```

GUARDAR MODELO.BEST_ESTIMATOR Y PARÁMETROS

In [9]:
```
modelo_best_estimator = modelo
```

Guardar modelo, parámetros y score

In [10]:
```
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

PREDECIR Y EVALUAR SOBRE EL TRAIN

Predecir sobre el Train

In [11]:
```
pred = modelo.best_estimator_.predict(train_x)
```

Evaluar sobre el Train

In [12]:
```
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: 0.9883720930232558
Accuracy: 0.9882352941176471
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        43
           1       0.98      1.00      0.99        42

    accuracy                           0.99        85
   macro avg       0.99      0.99      0.99        85
weighted avg       0.99      0.99      0.99        85
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

Predecir sobre la validación

In [13]:
```
pred = modelo.best_estimator_.predict(val_x)
pred_proba = modelo.best_estimator_.predict_proba(val_x)[:,1]
```

Evaluar sobre la validación

In [14]:
```
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}")
```

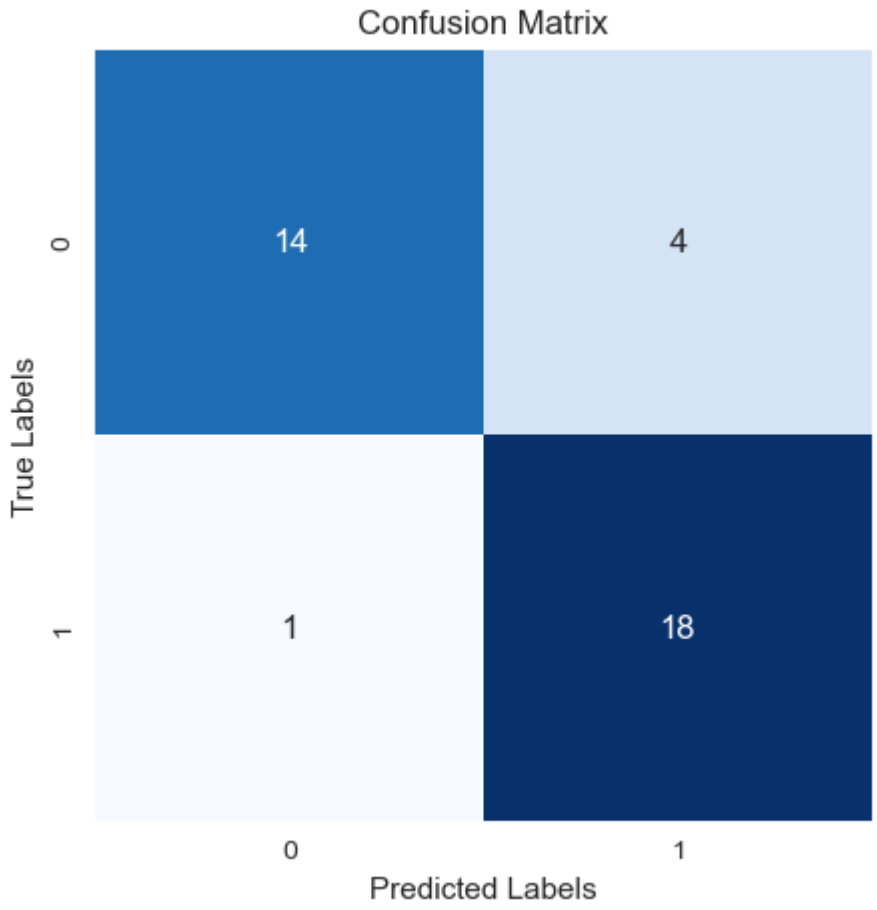
```
Roc AUC_proba: 0.9181286549707602
Roc AUC: 0.8625730994152047
Accuracy: 0.8648648648648649
Classification Report:               precision    recall  f1-score   support

           0       0.93      0.78      0.85        18
           1       0.82      0.95      0.88        19

    accuracy                           0.86        37
   macro avg       0.88      0.86      0.86        37
weighted avg       0.87      0.86      0.86        37
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass

In [15]:
```
# 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")
```

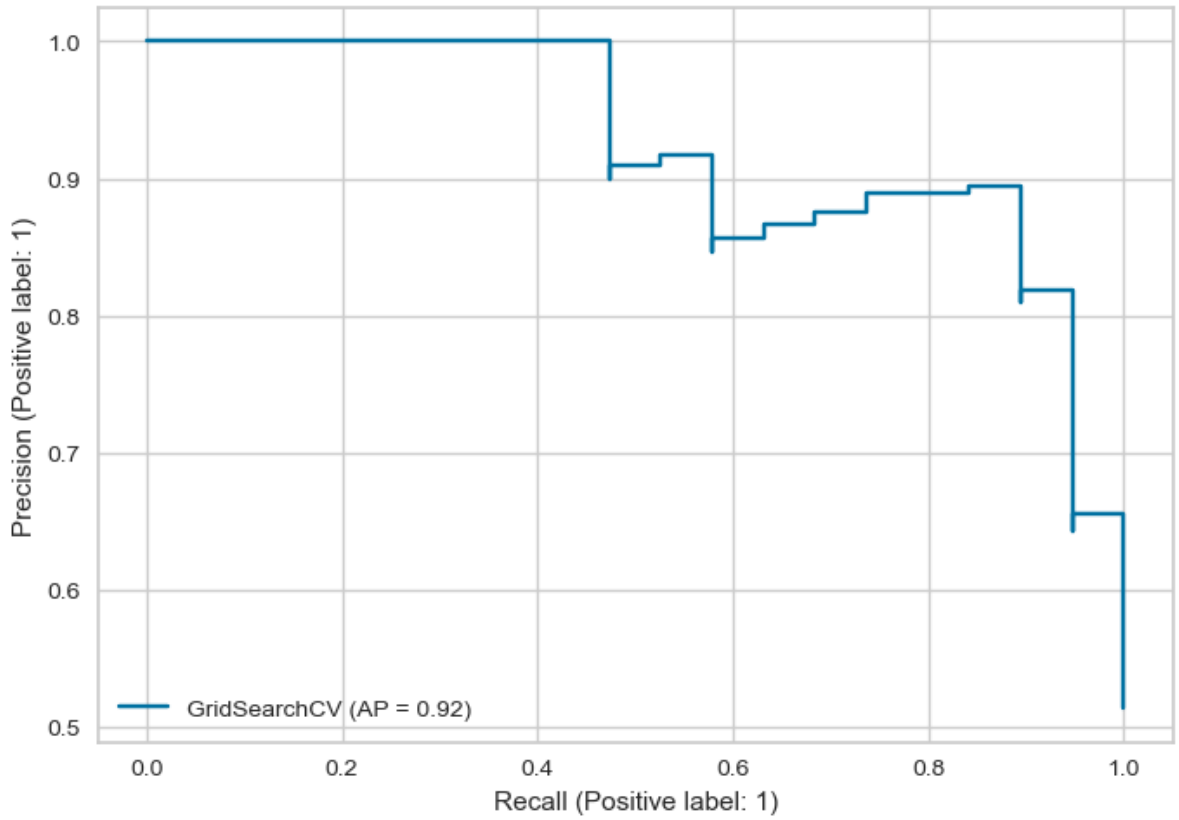
```
plt.ylabel("True Labels")
plt.title("Confusion Matrix");
```



Precision-Recall

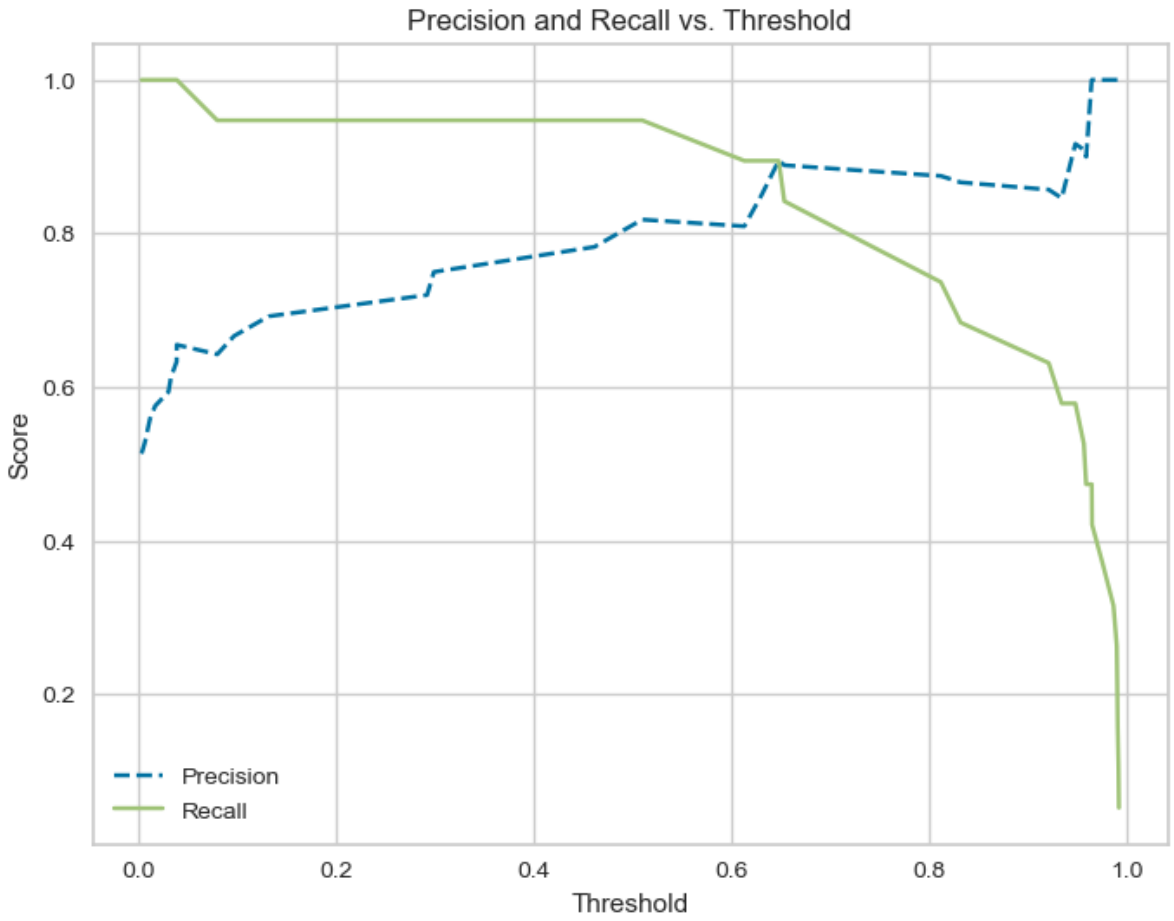
In [16]:
```
PrecisionRecallDisplay.from_estimator(modelo_best_estimator, val_x, val_y)
```

Out[16]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x27c8cd1cad0>
```



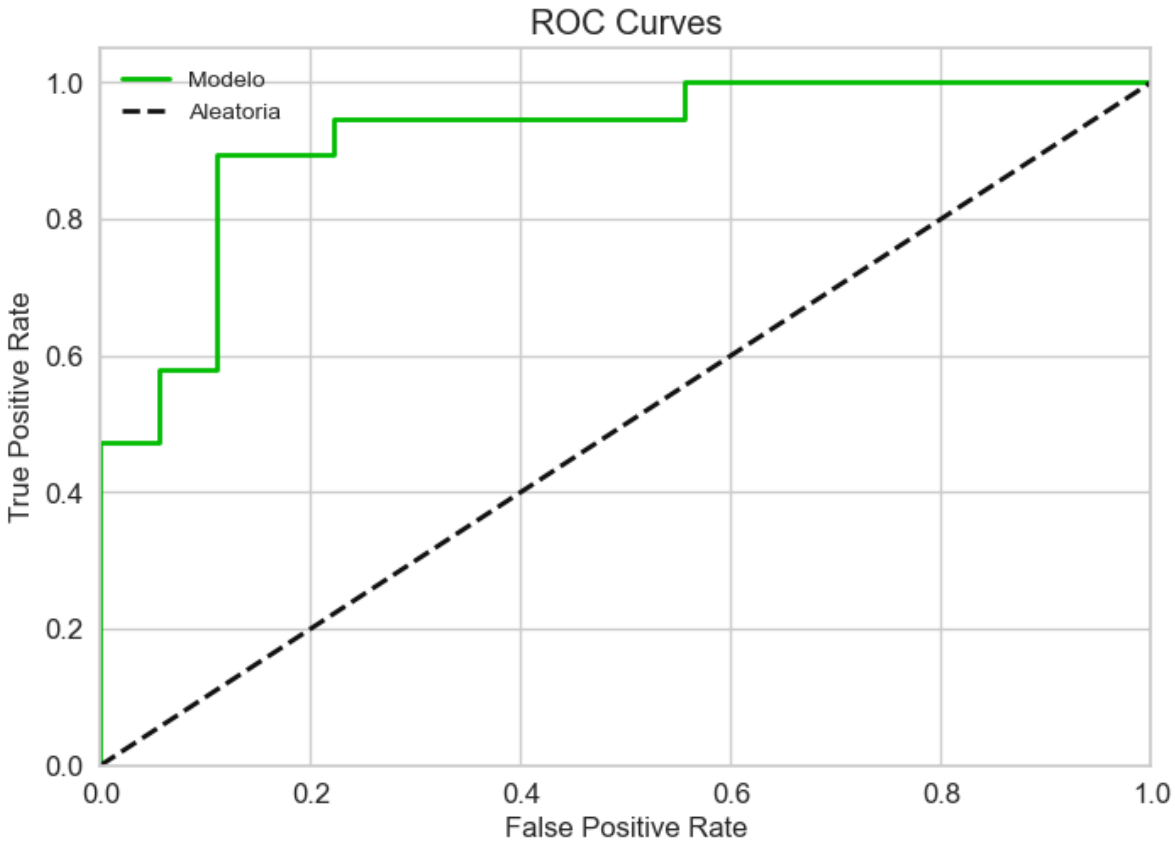
In [17]:
```
# Calcula la curva de precisión y recall para diferentes umbrales de corte
precision, recall, thresholds = precision_recall_curve(val_y, pred_proba)

# 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: 0.64831114
```



ROC Chart

In [30]:
```
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']);
```



ESTABLECER UMBRAL DE CORTE

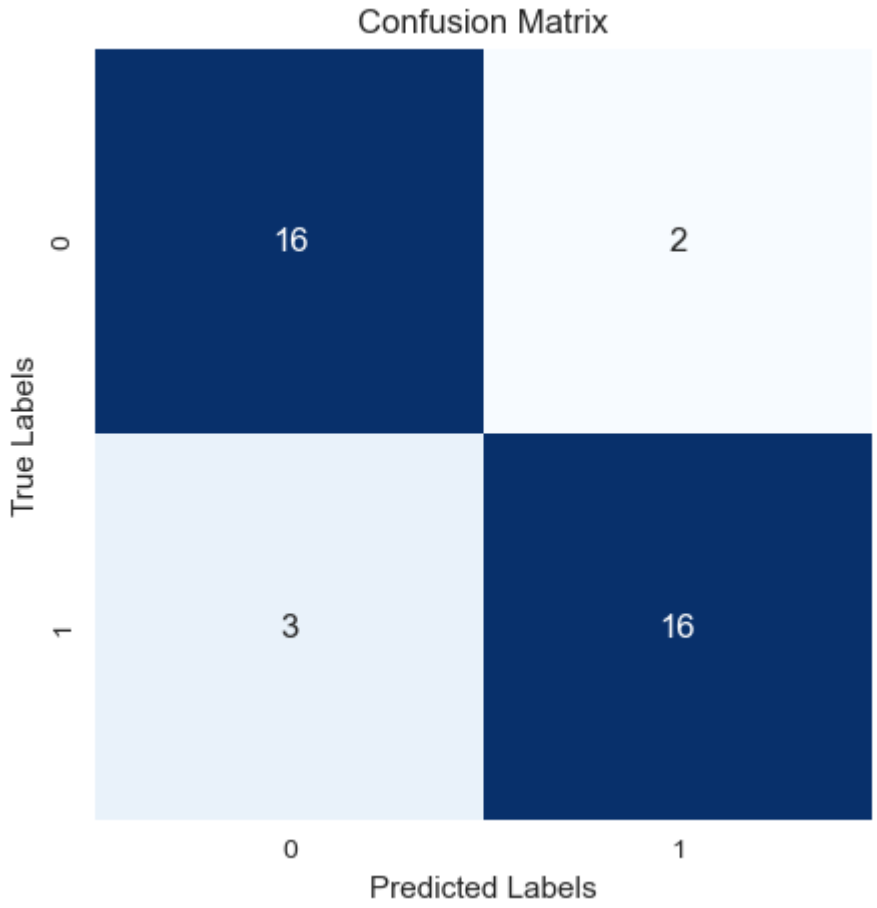
En este caso, al laboratorio o especialista le interesa tener una alta precisión en los casos y tener menos casos de falsos positivos. Por ese motivo vamos a aplicar un umbral de corte equilibrado entre precision y recall para tener el mayor caso de positivos reales y evitar falsos positivos.

In [26]:
```
def calcular_metricas(real, scoring, umbral):

    #CALCULAR LA DECISION SEGUN EL UMBRAL
    predicho = np.where(scoring > umbral,1,0)

    #CALCULAR TODAS LAS MÉTRICAS
    conf = confusion_matrix(real,predicho)

    tn, fp, fn, tp = conf.ravel()

    total_casos = y.shape[0]

    accuracy = (tn + tp) / total_casos
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    F1 = 2 * (precision * recall) / (precision + recall)

    #IMPRIMIR RESULTADOS
    # Crear un mapa de calor de la matriz de confusión
    plt.figure(figsize=(5, 5))
    sns.heatmap(conf, 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");
    print('\nUmbral de corte:', umbral)
```

```
    print('accuracy:',round(accuracy,3))
    print('precision:',round(precision,3))
    print('recall:',round(recall,3))
    print('F1:',round(F1,3))
```

In [29]:
```
calcular_metricas(val_y, pred_proba, best_threshold)
```

```
Umbral de corte: 0.64831114
accuracy: 0.262
precision: 0.889
recall: 0.842
F1: 0.865
```



GUARDAR BEST_ESTIMATOR, PARÁMETROS Y RESULTADOS DEL TEST Y LA VALIDACIÓN

Guardar el mejor estimador

In [39]:
```
version_estimator = '_v06'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[39]:
```
'XGBClassifier_v06.pickle'
```

In [40]:
```
m_best_estimator
```

Out[40]:
```
'XGBClassifier'
```

In [41]:
```
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 [42]:
```
# Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo XGBClassifier con parámetros para evitar el sobreajuste sin p
x_columns = list(x.columns)
y_target = y.name
```

In [43]:
```
#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[43]:
```
m_Best_estimator                                        XGBClassifier
m_Best_paramans     {'algoritmo': XGBClassifier(base_score=None, b...
m_Best_Score                                        0.9847222222222222
t_accuracy                                                    0.988235
t_report                          precision    recall  f1-score   ...
v_roc_auc_proba                                               0.918129
v_roc_auc                                                     0.862573
v_accuracy                                                    0.864865
v_report                          precision    recall  f1-score   ...
comentarios         Modelo XGBClassifier con parámetros para evita...
predictoras_X       [estacion, edad, e_infantil, acc_grave, int_qu...
target_y                                                     produccion
Name: XGBClassifier_v06.pickle, dtype: object
```

In [44]:
```
df_best = pd.read_excel('../../04_Modelos/Best_estimator/Best_estimator.xlsx',index
```

In [45]:
```
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 con los datos de valuación tiene un AUC [1] del 91% debido al Smote-Tomek. Hemos obtenido tanto un precisión como un recall muy buenos con los datos de validación y hemos aplicado el umbral de corte para ajustar entre estas dos variables.

PRÓXIMOS PASOS:

Realizaremos el modelo definitivo con este modelo. Pero antes veremos si podemos ajustar el precisión y recall para ajustar los falsos positivos.

09 - MODELO CLASIFICACIÓN TRAS THE BEST ESTIMATOR Y PRESELECCIÓN DE VARIABLES

Hemos comparado varios modelos de clasificación con parámetros para evitar el sobreajuste, preselección de variables y balanceo de datos. El mejor modelo ha sido el XGBClassifier (v6) sin preselección de variables con Smote-Tomek en el dataset.

En este notebook entrenaremos el modelo definitivo (v6) con todos los datos de trabajo con el Smote-Tomek y crearemos el modelo de ejecución para predecir con los datos de prueba en el próximo notebook.

XGBClassifier:

- 'algoritmo__learning_rate': 0.05,
- 'algoritmo__max_depth': 5,
- 'algoritmo__n_estimators': 500,
- 'algoritmo__n_jobs': -1,
- 'algoritmo__reg_alpha': 0.1,
- 'algoritmo__reg_lambda': 0.1,
- 'algoritmo__verbosity': 0

IMPORTACIÓN DE PAQUETES

In [16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scikitplot as skplt

from xgboost import XGBClassifier

#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

Utilizaremos los datos balanceados.

In [2]:
```
x = pd.read_pickle('../../02_Datos/03_Trabajo/x_balanceo.pickle')
y = pd.read_pickle('../../02_Datos/03_Trabajo/y_balanceo.pickle')
```

MODELIZAR

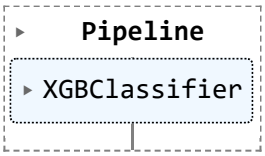
CARGAR EL MEJOR MODELO CON EL ALGORITMO, PARÁMETROS Y VALORES

In [3]:
```
modelo = pd.read_pickle('../../04_Modelos/Best_Estimator/XGBClassifier_v06.pickle')
```

EXAMINAR MODELO

In [4]:
```
modelo.best_estimator_
```

Out[4]:



In [5]:
```
modelo.best_params_
```

Out[5]:
```
{'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                            colsample_bylevel=None, colsample_bynode=None,
                            colsample_bytree=None, early_stopping_rounds=None,
                            enable_categorical=False, eval_metric=None, feature_types=None,
                            gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                            interaction_constraints=None, learning_rate=0.05, max_bin=None,
                            max_cat_threshold=None, max_cat_to_onehot=None,
                            max_delta_step=None, max_depth=5, max_leaves=None,
                            min_child_weight=None, missing=nan, monotone_constraints=None,
                            n_estimators=500, n_jobs=-1, num_parallel_tree=None,
                            predictor=None, random_state=None, ...),
 'algoritmo__learning_rate': 0.05,
 'algoritmo__max_depth': 5,
 'algoritmo__n_estimators': 500,
 'algoritmo__n_jobs': -1,
 'algoritmo__reg_alpha': 0.1,
 'algoritmo__reg_lambda': 0.1,
 'algoritmo__verbosity': 0}
```

PREDECIR SOBRE LA VALIDACIÓN

In [7]:
```
pred = modelo.best_estimator_.predict(x)
pred_proba = modelo.best_estimator_.predict_proba(x)[:,1]
```

EVALUAR SOBRE LA VALIDACIÓN

In [8]:
```
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}")
```

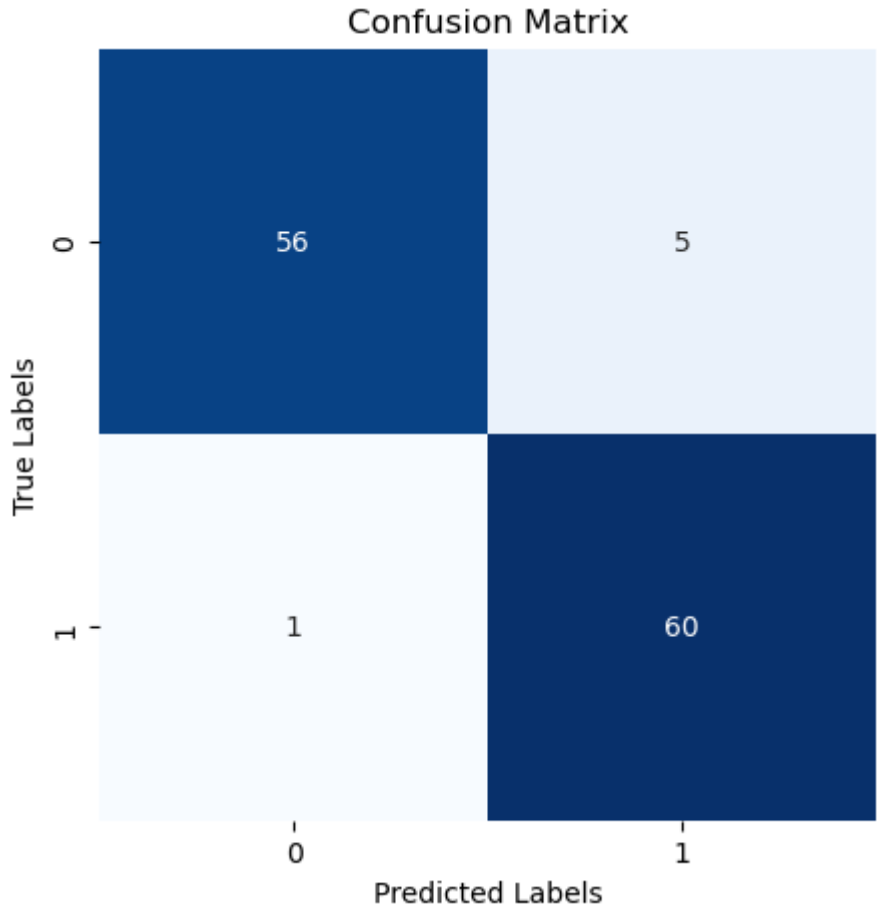
```
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:{v_report}")
```

```
Roc AUC_proba: 0.9822628325718893
Roc AUC: 0.9508196721311476
Accuracy: 0.9508196721311475
Classification Report:              precision    recall  f1-score   support

           0       0.98      0.92      0.95        61
           1       0.92      0.98      0.95        61

    accuracy                           0.95       122
   macro avg       0.95      0.95      0.95       122
weighted avg       0.95      0.95      0.95       122
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass

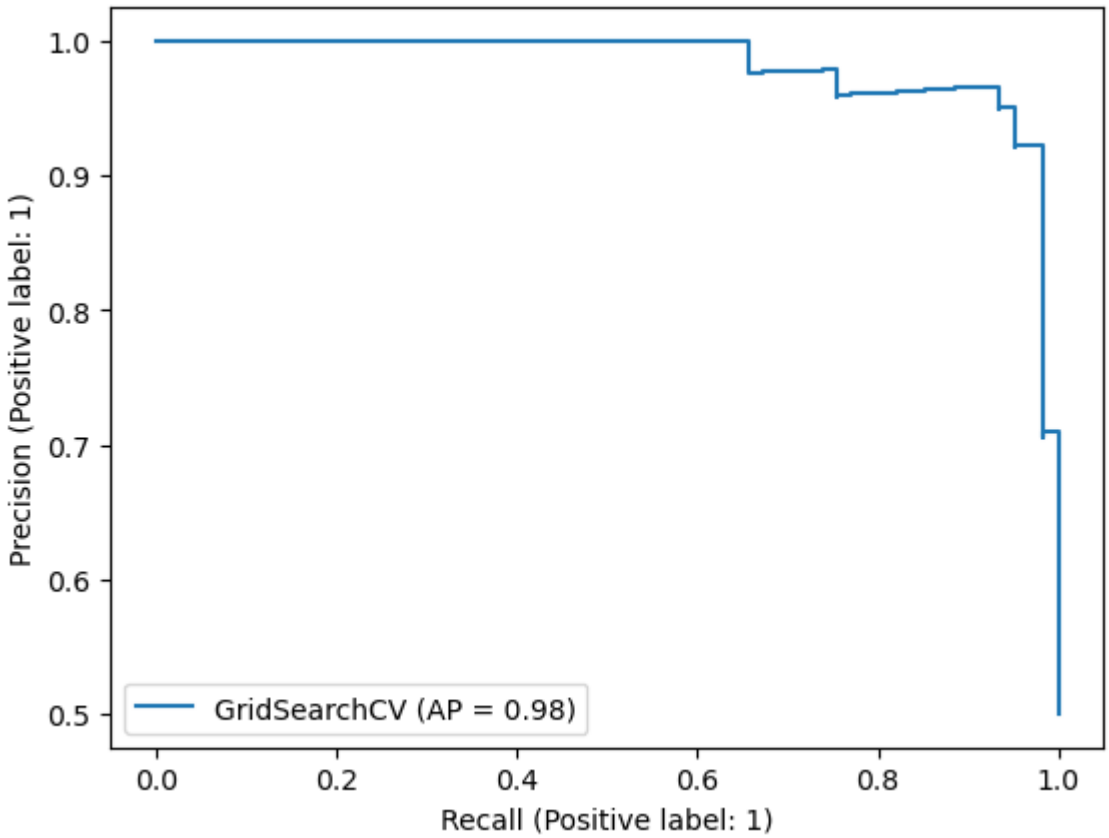
In [9]:
```
# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)


# 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

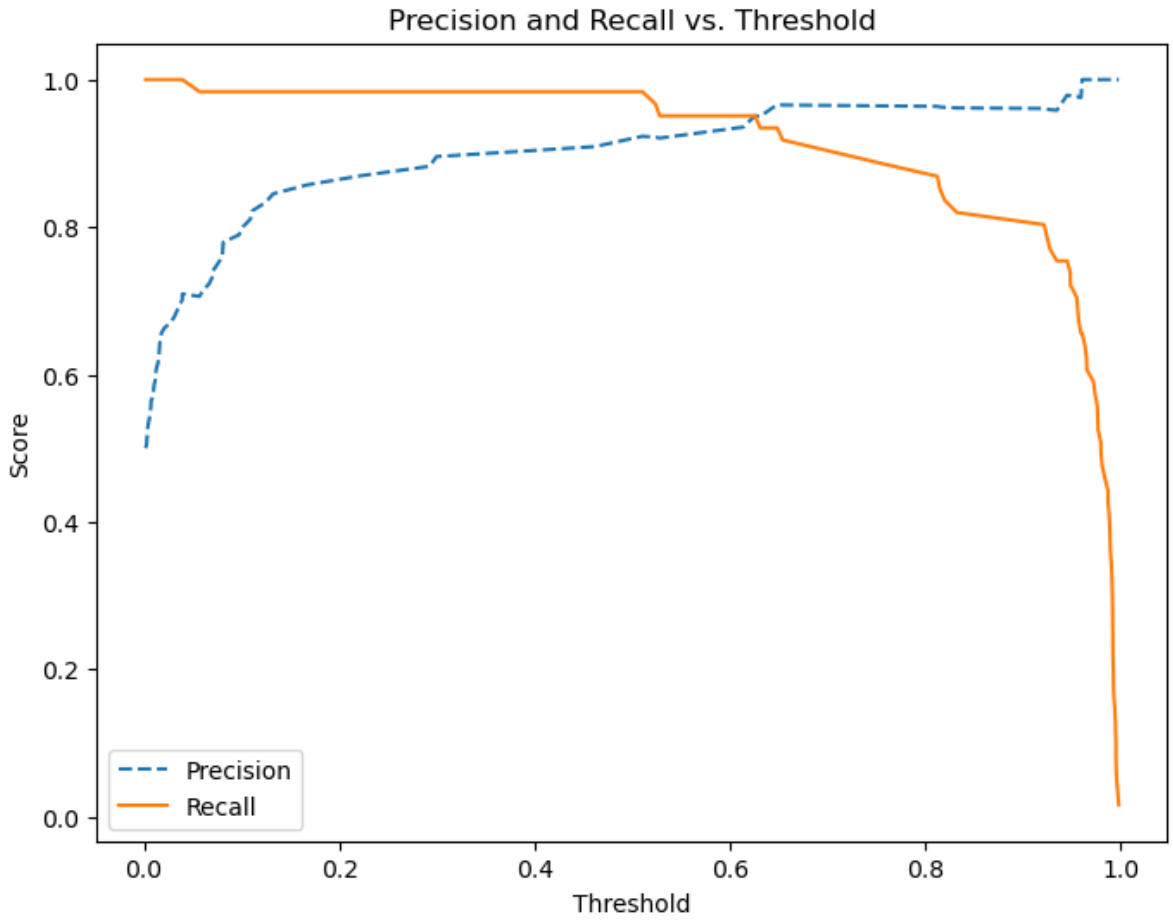
In [14]:
```
PrecisionRecallDisplay.from_estimator(modelo, x, y);
```

Out[14]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x16fb1d5e
dd0>
```



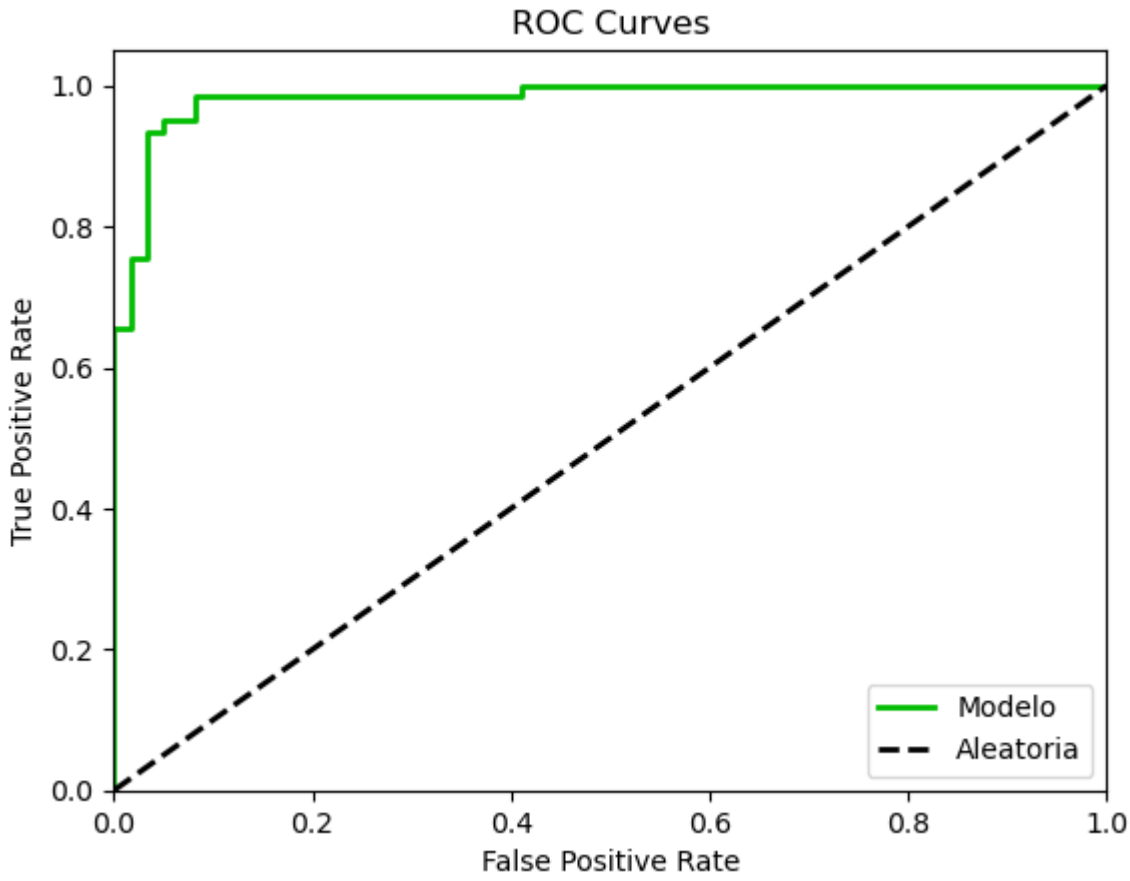
In [17]:
```
# Calcula la curva de precisión y recall para diferentes umbrales de corte
precision, recall, thresholds = precision_recall_curve(y, pred_proba)

# 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: 0.5102043
```



ROC Chart

In [11]:
```
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 [18]:
```
modelo = XGBClassifier(
                        learning_rate =  0.05,
                        max_depth =  5,
                        n_estimators =  500,
                        n_jobs =  -1,
                        reg_alpha =  0.1,
                        reg_lambda =  0.1,
                        verbosity = 0
                        )
```

CREAR Y GUARDAR EL PIPE FINAL DE ENTRENAMIENTO A

In [19]:
```
#crear pipe de entrenamiento
pipe_entrenamiento = make_pipeline(modelo)
```

In [20]:
```
# 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 [21]:
```
#Entrenar pipe de entrenamiento

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

In [22]:
```
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)
```

CONCLUSIÓN: El modelo ha dado muy buenos resultados obteniendo un AUC [1] del 98.2%.

- Roc AUC_proba: 0.9822628325718893
- Roc AUC: 0.9508196721311476
- Accuracy: 0.9508196721311475
- Classification Report:

```
              precision    recall  f1-score   support

           0       0.98      0.92      0.95        61
           1       0.92      0.98      0.95        61

    accuracy                           0.95       122
```

PRÓXIMOS PASOS: Vamos a probar el modelo definitivo con los datos de prueba.

10 - MODELO DE EJECUCIÓN CON LOS DATOS DE PRUEBA

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

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

CARGAR LOS DATOS

In [2]:
```
df = pd.read_csv('../../02_Datos/02_Validacion/prueba.csv', index_col= 0)
df.head()
```

Out[2]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_se
67	-0.33	0.50	1	0	0	1	1.0	-1	
15	1.00	0.81	1	1	0	0	1.0	1	
41	-1.00	0.56	1	1	0	0	0.8	1	
13	1.00	0.81	1	0	0	0	1.0	-1	
48	-0.33	0.64	1	1	1	0	0.8	-1	

ESTRUCTURA DEL DATASET

CALIDAD DE DATOS

In [3]:
```
# Cambio los valores manualmente
df['produccion'] = df.produccion.replace({'N': 0, 'O':1 })
df.head()
```

Out[3]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_se
67	-0.33	0.50	1	0	0	1	1.0	-1	
15	1.00	0.81	1	1	0	0	1.0	1	
41	-1.00	0.56	1	1	0	0	0.8	1	
13	1.00	0.81	1	0	0	0	1.0	-1	
48	-0.33	0.64	1	1	1	0	0.8	-1	

SEPARAR PREDICTORAS Y TARGET

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

In [5]:
```
df[df.produccion == 1]
```

Out[5]:

	estacion	edad	e_infantil	acc_grave	int_quirurgica	fiebre_ult_any	frec_alcohol	fumar	hr_se
84	-0.33	0.78	1	0	0	1	1.0	1	
29	1.00	0.67	0	0	1	0	0.6	0	
23	1.00	0.69	1	0	1	-1	1.0	-1	

MODELIZAR CON EL PIPE DE EJECUCIÓN

CARGAMOS EL PIPE DE EJECUCIÓN

In [6]:
```
modelo = pd.read_pickle('../../04_Modelos/pipe_ejecucion.pickle')
```

PREDECIR Y EVALUAR CON DATASET DE PRUEBA

PREDECIR SOBRE LOS DATOS

In [7]:
```
pred = modelo.predict(x)
pred_proba = modelo.predict_proba(x)[:,1]
```

EVALUAR SOBRE LOS DATOS

In [8]:
```
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: 0.728395061728395
Roc AUC: 0.6296296296296295
Accuracy: 0.8666666666666667
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        27
           1       0.33      0.33      0.33         3

    accuracy                           0.87        30
   macro avg       0.63      0.63      0.63        30
weighted avg       0.87      0.87      0.87        30
```

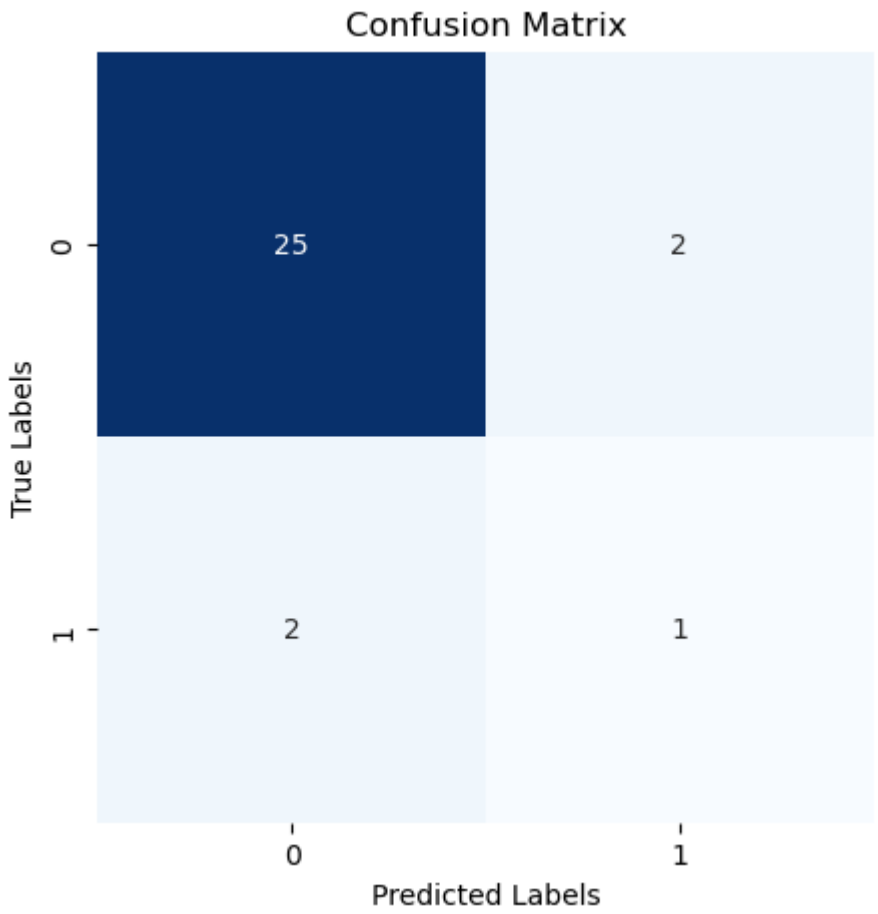
In [22]:
```
modelo.get_params
```

Out[22]:
```
<bound method Pipeline.get_params of Pipeline(steps=[('xgbclassifier',
                 XGBClassifier(base_score=None, booster=None, callbacks=None,
                               colsample_bylevel=None, colsample_bynode=None,
                               colsample_bytree=None,
                               early_stopping_rounds=None,
                               enable_categorical=False, eval_metric=None,
                               feature_types=None, gamma=None, gpu_id=None,
                               grow_policy=None, importance_type=None,
                               interaction_constraints=None, learning_rate=0.05,
                               max_bin=None, max_cat_threshold=None,
                               max_cat_to_onehot=None, max_delta_step=None,
                               max_depth=5, max_leaves=None,
                               min_child_weight=None, missing=nan,
                               monotone_constraints=None, n_estimators=500,
                               n_jobs=-1, num_parallel_tree=None,
                               predictor=None, random_state=None, ...))])>
```

REPORTING DEL MODELO

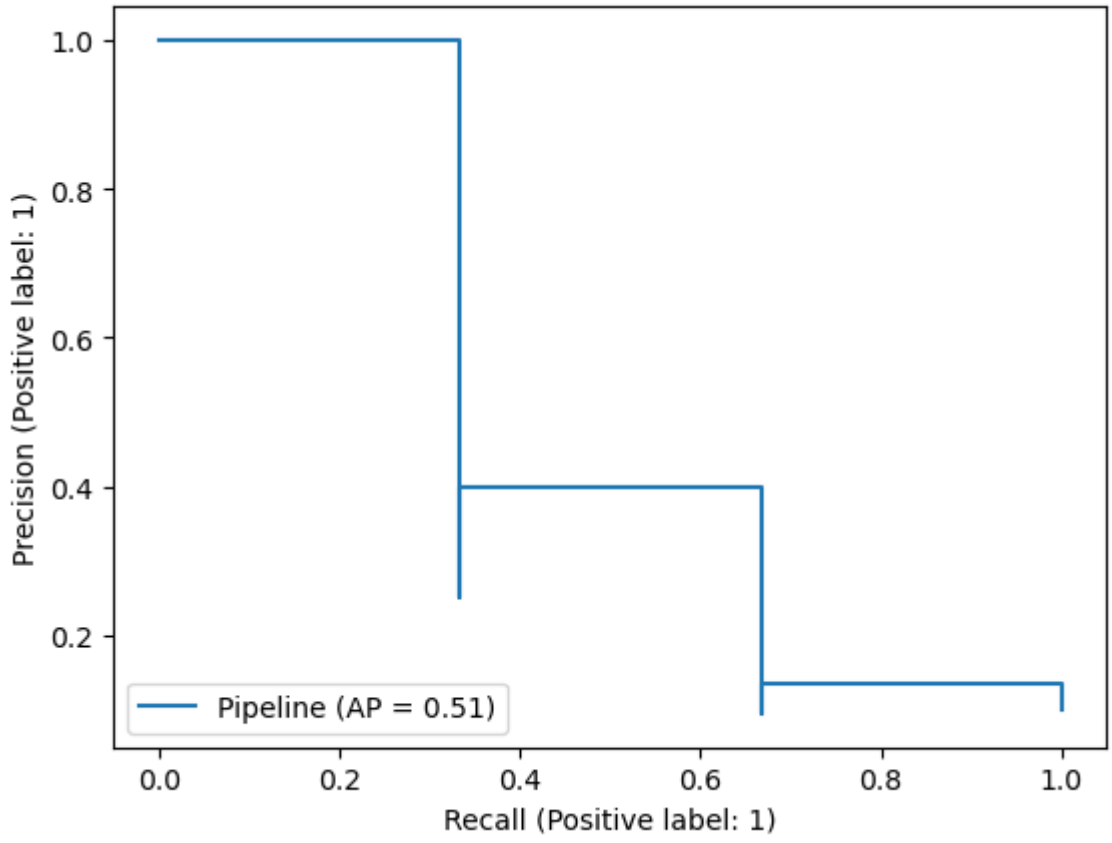
Matrix de Confusión MultiClass

In [9]:
```
# 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");
```



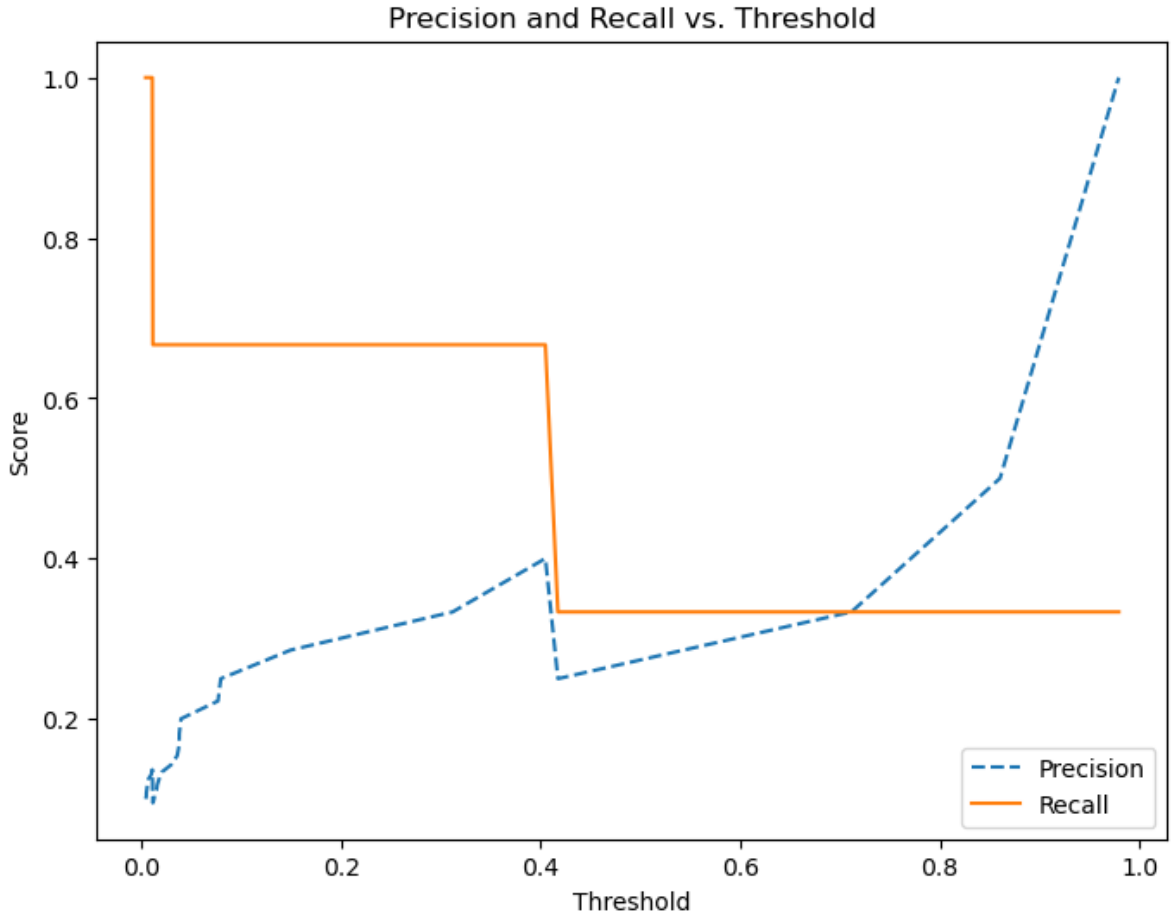
Precision-Recall

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



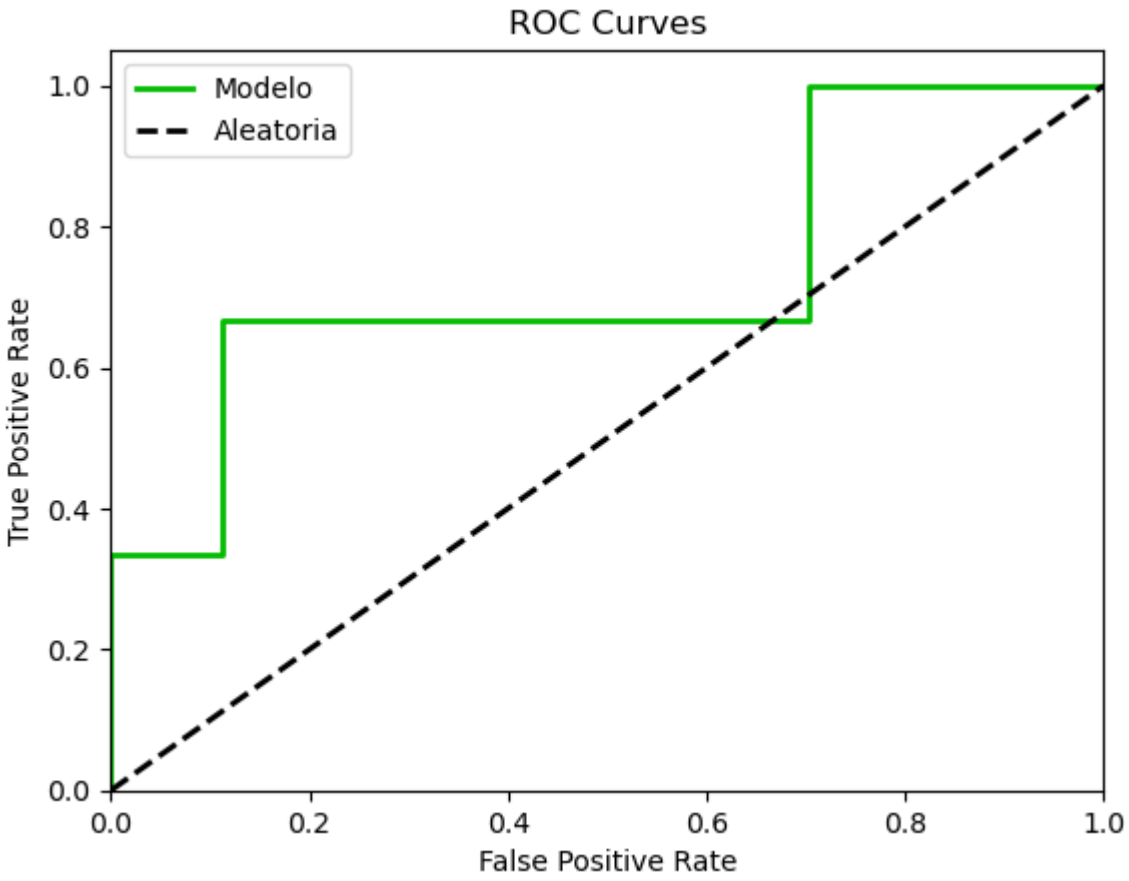
In [15]:
```
# Calcula la curva de precisión y recall para diferentes umbrales de corte
precision, recall, thresholds = precision_recall_curve(y, pred_proba)

# 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: 0.40465593
```



ROC Chart

In [10]:
```
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']);
```



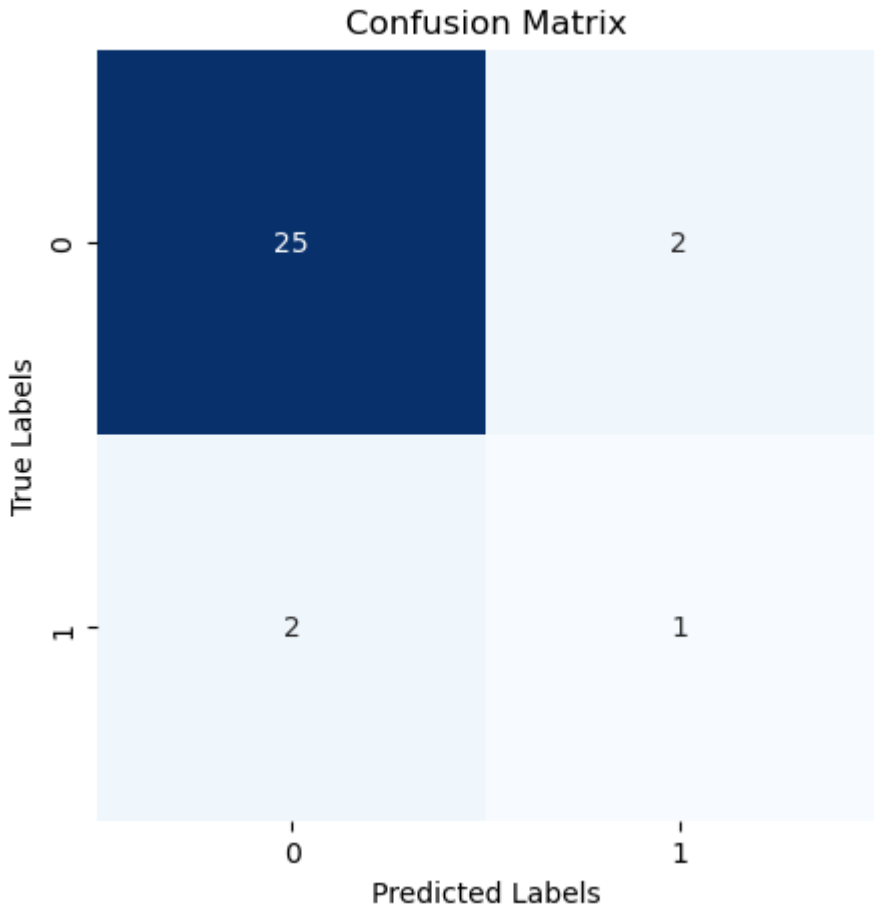
ESTABLECER UMBRAL DE CORTE

In [32]:
```
#Indicamos el umbral de corte establecido con el modelo.

umbral_corte = 0.64831114
```

In [33]:
```
def calcular_metricas(real, scoring, umbral):

    #CALCULAR LA DECISION SEGUN EL UMBRAL
    predicho = np.where(scoring > umbral,1,0)

    #CALCULAR TODAS LAS MÉTRICAS
    conf = confusion_matrix(real,predicho)

    tn, fp, fn, tp = conf.ravel()

    total_casos = y.shape[0]

    accuracy = (tn + tp) / total_casos
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    F1 = 2 * (precision * recall) / (precision + recall)

    #IMPRIMIR RESULTADOS
    # Crear un mapa de calor de la matriz de confusión
    plt.figure(figsize=(5, 5))
    sns.heatmap(conf, 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");
    print('\nUmbral de corte:', umbral)
    print('accuracy:',round(accuracy,3))
```

```
    print('precision:',round(precision,3))
    print('recall:',round(recall,3))
    print('F1:',round(F1,3))

calcular_metricas(y, pred_proba, umbral_corte)
```

```
Umbral de corte: 0.64831114
accuracy: 0.867
precision: 0.333
recall: 0.333
F1: 0.333
```



CONCLUSIÓN: El modelo con los datos de entrenamiento ha dado muy buenos resultados obteniendo un AUC [1] del 98.2%.

- Roc AUC_proba: 0.9822628325718893
- Roc AUC: 0.9508196721311476
- Accuracy: 0.9508196721311475
- Classification Report:

	precision	recall	f1-score	support
0	0.98	0.92	0.95	61
1	0.92	0.98	0.95	61
accuracy			0.95	122

Al predecir con datos de prueba (nuevos) el modelo no ha funcionado tan bien solo hemos podido . No podemos establecer que sea un mal modelo, pero necesitaríamos disponer de más datos para obtener un modelo más eficiente.

Modelo definitivo

- Roc AUC_proba: 0.728395061728395
- Roc AUC: 0.6296296296296295
- Accuracy: 0.8666666666666667
- Classification Report:

```
  precision      recall  f1-score   support

     0           0.93      0.93      0.93        27
     1           0.33      0.33      0.33         3
```

En este caso, tendríamos valorar la opción de negocio y para que utilizaremos los datos:

Alto Umbral

Ej. Farmacéutica: queremos probar medicación con pacientes con baja cualidad seminal, subiría el umbral para detectar solo aquellos casos que estemos seguros que son pacientes reales con baja cualidad.

Bajo Umbral

Ej. Nutricionista: Tenemos una dieta que mejora la cualidad seminal de los pacientes y queremos ofrecérselo a nuestros clientes. Nos interesa como empresa ofrecer al mayor número de clientes el producto aunque sean falsos positivos ya que mejoría su cualidad y llegaríamos a más clientes.