```
### -----------------------------------------------------------------------
### Trabalho Final  Inteligência Artificial
### Adriano G e Jean M.
### -----------------------------------------------------------------------
```

```
#import h1 as h1
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#import seaborn as sns
import statistics  as sts
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

Clique duas vezes (ou pressione "Enter") para editar

Clique duas vezes (ou pressione "Enter") para editar

```
df=pd.read_csv('Titanic-Dataset.csv')
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Resumo geral da base de dados

```
print ("Linhas: " , df.shape[0])
print ("Colunas: " , df.shape[1])
print ("\nAtributos : \n" , df.columns.tolist())
print ("\nValores faltantes :  ", df.isnull().sum().values.sum())
print ("\nValores únicos :  \n",df.nunique())
```

```
Linhas:  891
Colunas:  12

Atributos :
 ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']

Valores faltantes :   866

Valores únicos :
 PassengerId    891
Survived         2
Pclass           3
Name           891
Sex              2
Age             88
SibSp            7
Parch            7
Ticket         681
Fare           248
Cabin          147
Embarked         3
dtype: int64
```

✓ 0s conclusão: 19:27

Pré-processamento

Remoção ID feature e verificação de dados faltantes

```
# Verifica a quantidade de dados faltrantes
df.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
### Substituição NAs Age pela mediana
mediana = df['Age'].median()
#preenche NAs
df['Age'].fillna(mediana, inplace=True)
df['Age'].isna().sum()
```

```
0
```

```
df['Embarked'].fillna('S', inplace=True)
df['Embarked'].isna().sum()
```

```
0
```

```
###Troca Cabines vazias pela moda
cabin_mode = df['Cabin'].mode()[0]
df['Cabin'].fillna(cabin_mode, inplace=True)
df['Cabin'].isna().sum()
```

```
0
```

Carregar dados para previsao da Cabine com um treino de random forest

```
### crio uma nova variavel com as colunas relevantes para previsao
###df_cabines = df[['Pclass','Sex','Age','Fare','Cabin','Embarked']]
#df_cabines
```

```
### separo em cabines conhecidas e cabines desconhecidas
#df_conhecidas = df_cabines.dropna(subset=['Cabin'])
#df_desconhecidas = df_cabines[df_cabines['Cabin'].isna()]
#df_desconhecidas
```

```
#df_conhecidas
```

```
### removendo a coluna cabin para separarmos atributos e rotulos
### Separar os dados conhecidos em atributos e rótulos
#atributos_conhecidos = df_conhecidas.drop('Cabin', axis=1)
#cabines_conhecidas = df_conhecidas['Cabin']
```

```
### convertendo para Dummy
#atributos_conhecidos = pd.get_dummies(atributos_conhecidos)
#cabines_conhecidas = pd.get_dummies(cabines_conhecidas)
```

```
###treinar com o modelo random forest
#modelo = RandomForestClassifier()
#modelo.fit(atributos_conhecidos,cabines_conhecidas)
```

```
###prever valores faltantes na coluna cabin
#atributos_desconhecidos = df_desconhecidas.drop('Cabin',axis=1)
#atributos_desconhecidos = pd.get_dummies(atributos_desconhecidos)
#prever_cabine = modelo.predict(atributos_desconhecidos)
#prever_cabine = prever_cabine[:, 0]


#prever_cabine
```

```
# Criar DataFrame com as previsões codificadas
#df_previsoes = pd.DataFrame(prever_cabine, columns=[f'Cabin_{i}' for i in range(len(modelo.classes_))])

#df_previsoes_revertidas = df_previsoes.idxmax(axis=1)

#df.loc[df['Cabin'].isna(), 'Cabin'] = df_previsoes_revertidas.values
```

```
# Verifica a quantidade de dados faltantes
#df.isna().sum()
```

```
#df
```

##

```
df=df.drop(columns='PassengerId',axis=1)
df=df.drop(columns='Ticket',axis=1)
df=df.drop(columns='Name',axis=1)
df
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	B96 B98	S
1	1	1	female	38.0	1	0	71.2833	C85	C
2	1	3	female	26.0	0	0	7.9250	B96 B98	S
3	1	1	female	35.0	1	0	53.1000	C123	S
4	0	3	male	35.0	0	0	8.0500	B96 B98	S
...	...	...	...	...	...	...	...	...	...

886	0	2	male	27.0	0	0	13.0000	B96 B98	S
887	1	1	female	19.0	0	0	30.0000	B42	S
888	0	3	female	28.0	1	2	23.4500	B96 B98	S
889	1	1	male	26.0	0	0	30.0000	C148	C
890	0	3	male	32.0	0	0	7.7500	B96 B98	Q

891 rows × 9 columns

```
df[["Survived","Sex"]].value_counts()
```

```
Survived  Sex
0         male      468
1         female    233
          male      109
0         female     81
Name: count, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder

labelencoder = LabelEncoder()
df[["Survived","Sex"]] = \
df[["Survived","Sex"]].apply(labelencoder.fit_transform)
```

Atributos com mais de 2 valores,

```
df = pd.get_dummies(data=df, columns=['Pclass'])
df = pd.get_dummies(data=df, columns=['SibSp'])
df = pd.get_dummies(data=df, columns=['Parch'])
df = pd.get_dummies(data=df, columns=['Cabin'])
df = pd.get_dummies(data=df, columns=['Embarked'])
```

```
df.columns
```

```
Index(['Survived', 'Sex', 'Age', 'Fare', 'Pclass_1', 'Pclass_2', 'Pclass_3',
       'SibSp_0', 'SibSp_1', 'SibSp_2',
       ...
       'Cabin_F G73', 'Cabin_F2', 'Cabin_F33', 'Cabin_F38', 'Cabin_F4',
       'Cabin_G6', 'Cabin_T', 'Embarked_C', 'Embarked_Q', 'Embarked_S'],
      dtype='object', length=171)
```

```
std=StandardScaler()
columns = ['Age','Fare']
scaled = std.fit_transform(df[['Age','Fare']])
scaled = pd.DataFrame(scaled,columns=columns)
df=df.drop(columns=columns,axis=1)
```

```
df=df.merge(scaled, left_index=True, right_index=True, how = "right")
df
```

	Survived	Sex	Pclass_1	Pclass_2	Pclass_3	SibSp_0	SibSp_1	SibSp_2	SibSp_3	SibSp_4	...	Cabin_F33	Cabin_F38	Cabin_F4	Cabin_G6	Cabin_T	Embarked_C	Embarked_Q	Embarked_S	Age	
0	0	1	False	False	True	False	True	False	False	False	...	False	False	False	False	False	False	False	True	-0.565736	-0.5
1	1	0	True	False	False	False	True	False	False	False	...	False	False	False	False	False	True	False	False	0.663861	0.7
2	1	0	False	False	True	True	False	False	False	False	...	False	False	False	False	False	False	False	True	-0.258337	-0.4
3	1	0	True	False	False	False	True	False	False	False	...	False	False	False	False	False	False	False	True	0.433312	0.4
4	0	1	False	False	True	True	False	False	False	False	...	False	False	False	False	False	False	False	True	0.433312	-0.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
886	0	1	False	True	False	True	False	False	False	False	...	False	False	False	False	False	False	False	True	-0.181487	-0.3
887	1	0	True	False	False	True	False	False	False	False	...	False	False	False	False	False	False	False	True	-0.796286	-0.0
888	0	0	False	False	True	False	True	False	False	False	...	False	False	False	False	False	False	False	True	-0.104637	-0.1
889	1	1	True	False	False	True	False	False	False	False	...	False	False	False	False	False	True	False	False	-0.258337	-0.0
890	0	1	False	False	True	True	False	False	False	False	...	False	False	False	False	False	False	True	False	0.202762	-0.4

891 rows × 171 columns

Separação entre treino e teste (70% e 30%)

```
X = df.drop(['Survived'], axis=1).values
y = df['Survived'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Aplica balanceamento nas classes

```
from imblearn.over_sampling import SMOTE
```

```
sm = SMOTE()
x_train_oversampled, y_train_oversampled = sm.fit_resample(X_train, y_train)
```

```
print(x_train_oversampled.shape)
print(X_train.shape)
```

```
(772, 170)
(623, 170)
```

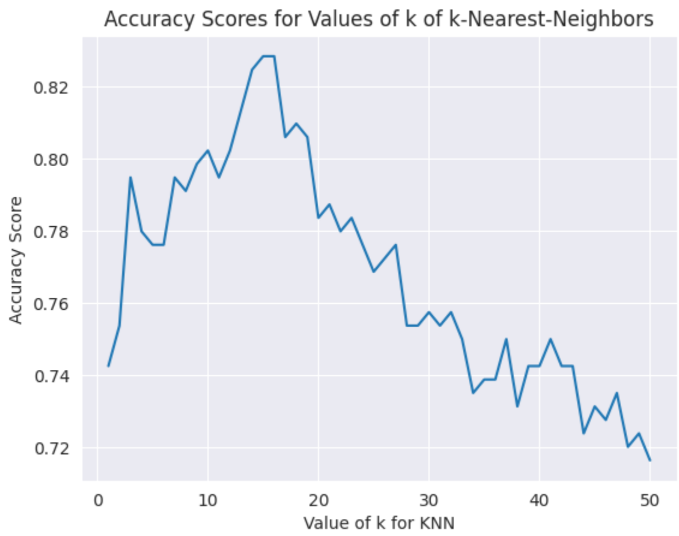
KNN Classifier

```
df.columns
```

```
Index(['Survived', 'Sex', 'Pclass_1', 'Pclass_2', 'Pclass_3', 'SibSp_0',
       'SibSp_1', 'SibSp_2', 'SibSp_3', 'SibSp_4',
       ...
       'Cabin_F33', 'Cabin_F38', 'Cabin_F4', 'Cabin_G6', 'Cabin_T',
       'Embarked_C', 'Embarked_Q', 'Embarked_S', 'Age', 'Fare'],
      dtype='object', length=171)
```

```
# Teste para diferentes valores de k
k_range = list(range(1,51))
scores = []
for k in k_range:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=8)
    knn = KNeighborsClassifier(n_neighbors=k,metric = 'euclidean')
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

plt.plot(k_range, scores)
```

```
plt.xlabel('Value of k for KNN')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Values of k of k-Nearest-Neighbors')
plt.show()
```




```
## Normalização dos dados

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=8)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(623, 170)
(623,)
(268, 170)
(268,)
```

```
X_train
```

```
array([[0, False, False, ..., True, -0.10463740114712752,
        -0.32425318983493084],
       [1, False, True, ..., True, -0.2583370877897099,
        -0.43700743807979686],
       [0, True, False, ..., True, -0.5657364610748746,
        2.4029901895877646],
       ...,
       [1, False, True, ..., False, -0.027787557825836348,
        -0.090272016977708476],
       [1, False, True, ..., True, -2.1027333275006983,
        -0.12491978668775715],
       [1, False, False, ..., True, -0.10463740114712752,
        -0.2463983948905696]], dtype=object)
```

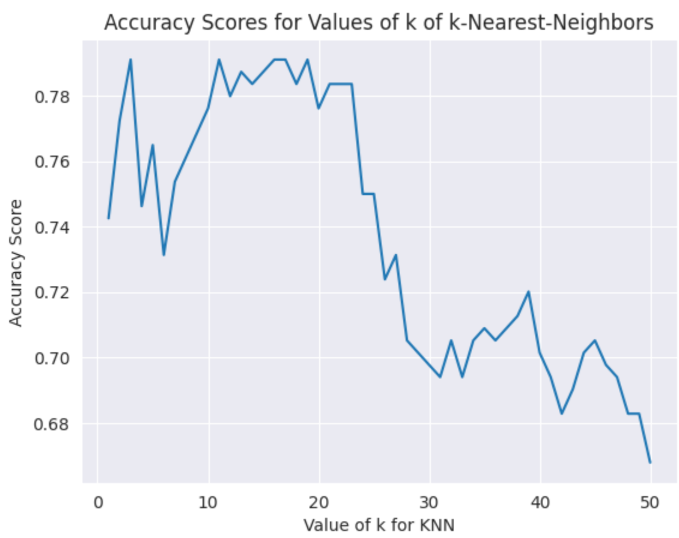
```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaler.fit(X_train)

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

```
X_train
```

```
array([[-1.32079271, -0.59279843, -0.52098807, ...,  0.63245553,
        -0.14762916, -0.33963155],
       [ 0.75712108, -0.59279843,  1.91942974, ...,  0.63245553,
        -0.29626272, -0.44159032],
       [-1.32079271,  1.68691405, -0.52098807, ...,  0.63245553,
        -0.59352985,  2.12649611],
       ...,
       [ 0.75712108, -0.59279843,  1.91942974, ..., -1.58113883,
        -0.07331237, -0.12805254],
       [ 0.75712108, -0.59279843,  1.91942974, ...,  0.63245553,
        -2.0798655 , -0.15938302],
       [ 0.75712108, -0.59279843, -0.52098807, ...,  0.63245553,
        -0.14762916, -0.26923084]])
```

```
# experimenting with different n values
k_range = list(range(1,51))
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k, metric = 'euclidean')
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

plt.plot(k_range, scores)
plt.xlabel('Value of k for KNN')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Values of k of k-Nearest-Neighbors')
plt.show()
```



Random Forest Classifier

```
forest = RandomForestClassifier()
```

```
param_grid = {'criterion': ['gini', 'entropy', 'log_loss'],
              'max_features':['sqrt','log2'],
              'n_estimators': [10, 20, 30, 60]}
```

```
# cria o objeto g_search
g_search = GridSearchCV(estimator = forest, param_grid = param_grid,
                        refit=True, scoring='accuracy', cv = 10)
```

```
# Faz o treinamento
g_search.fit(x_train_oversampled, y_train_oversampled);
print(g_search.best_params_)
```

```
{'criterion': 'entropy', 'max_features': 'sqrt', 'n_estimators': 60}
```

```
# Carrega todos os dados do GridSearch em um Dataframe
g_results =  pd.DataFrame(g_search.cv_results_)
# Nome de todos atributos gerados pelo GridSearch
g_search.cv_results_.keys()
```

```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_criterion', 'param_max_features', 'param_n_estimators', 'params', 'split0_test_score',
'split1_test_score', 'split2_test_score', 'split3_test_score', 'split4_test_score', 'split5_test_score', 'split6_test_score', 'split7_test_score', 'split8_test_score', 'split9_test_score',
'mean_test_score', 'std_test_score', 'rank_test_score'])
```

```
# Obtém a média das acurácias (10 folds) referente ao conjunto treino
g_results.loc[g_search.best_index_,'mean_test_score']
```

```
0.8654179154179154
```

```
# Avalia o conjunto teste  com o melhor conjunto de parâmetros encontrado
# best_estimator_ .Para tanto, o parâmetro refit precisa ser igual a True
model = g_search.best_estimator_
model.score(X_test,y_test)
```

```
0.44776119402985076
```

Amostragem por validação cruzada estratificada

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
```

```
model = RandomForestClassifier()
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=skf)
mean_score = scores.mean()
mean_score
```

```
0.80019975031211
```

Clique duas vezes (ou pressione "Enter") para editar

```
train_scores = cross_val_score(model, X_train, y_train, cv=skf)
mean_train_accuracy = train_scores.mean()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
test_accuracy
```

```
0.8171641791044776
```

Redes Neurais?

```
from sklearn.neural_network import MLPClassifier
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=8)
rna = MLPClassifier(hidden_layer_sizes=(10,10, 10), activation='relu', solver='sgd', max_iter =800,
                            tol=0.0001, random_state = 3, verbose = True)
rna.fit(X_train, y_train)
```

```
Iteration 1, loss = 0.67145487
Iteration 2, loss = 0.67099706
Iteration 3, loss = 0.67027239
Iteration 4, loss = 0.66925550
Iteration 5, loss = 0.66799033
Iteration 6, loss = 0.66655698
Iteration 7, loss = 0.66518815
Iteration 8, loss = 0.66382538
Iteration 9, loss = 0.66256233
Iteration 10, loss = 0.66124234
Iteration 11, loss = 0.65983154
Iteration 12, loss = 0.65824093
Iteration 13, loss = 0.65671758
Iteration 14, loss = 0.65526609
Iteration 15, loss = 0.65380661
Iteration 16, loss = 0.65260234
Iteration 17, loss = 0.65143065
Iteration 18, loss = 0.65032130
Iteration 19, loss = 0.64942780
Iteration 20, loss = 0.64850545
Iteration 21, loss = 0.64776365
Iteration 22, loss = 0.64708682
Iteration 23, loss = 0.64643215
Iteration 24, loss = 0.64579651
Iteration 25, loss = 0.64511742
Iteration 26, loss = 0.64448010
Iteration 27, loss = 0.64376328
Iteration 28, loss = 0.64308842
Iteration 29, loss = 0.64248080
Iteration 30, loss = 0.64182538
Iteration 31, loss = 0.64119216
Iteration 32, loss = 0.64052061
Iteration 33, loss = 0.63985296
Iteration 34, loss = 0.63920933
Iteration 35, loss = 0.63855072
Iteration 36, loss = 0.63786753
Iteration 37, loss = 0.63719039
Iteration 38, loss = 0.63656041
Iteration 39, loss = 0.63596623
Iteration 40, loss = 0.63534838
Iteration 41, loss = 0.63478247
Iteration 42, loss = 0.63413037
Iteration 43, loss = 0.63351641
Iteration 44, loss = 0.63289265
Iteration 45, loss = 0.63225689
Iteration 46, loss = 0.63164502
Iteration 47, loss = 0.63102053
Iteration 48, loss = 0.63037903
Iteration 49, loss = 0.62970201
Iteration 50, loss = 0.62903096
Iteration 51, loss = 0.62839546
Iteration 52, loss = 0.62773101
Iteration 53, loss = 0.62703347
Iteration 54, loss = 0.62636412
Iteration 55, loss = 0.62570258
Iteration 56, loss = 0.62501471
Iteration 57, loss = 0.62433732
```

```
Iteration 58, loss = 0.62359741
Iteration 59, loss = 0.62280220
Iteration 60, loss = 0.62208911
Iteration 61, loss = 0.62135834
Iteration 62, loss = 0.62056254
Iteration 63, loss = 0.61973175
Iteration 64, loss = 0.61890111
Iteration 65, loss = 0.61809176
Iteration 66, loss = 0.61722596
Iteration 67, loss = 0.61636653
Iteration 68, loss = 0.61555786
Iteration 69, loss = 0.61468966
Iteration 70, loss = 0.61385587
Iteration 71, loss = 0.61300624
Iteration 72, loss = 0.61213171
Iteration 73, loss = 0.61128530
Iteration 74, loss = 0.61046752
Iteration 75, loss = 0.60959054
Iteration 76, loss = 0.60867686
Iteration 77, loss = 0.60781566
Iteration 78, loss = 0.60686836
Iteration 79, loss = 0.60597767
Iteration 80, loss = 0.60503895
Iteration 81, loss = 0.60404477
Iteration 82, loss = 0.60308705
Iteration 83, loss = 0.60218493
Iteration 84, loss = 0.60122819
Iteration 85, loss = 0.60035258
Iteration 86, loss = 0.59951911
Iteration 87, loss = 0.59866740
Iteration 88, loss = 0.59772472
Iteration 89, loss = 0.59679925
Iteration 90, loss = 0.59582697
Iteration 91, loss = 0.59475792
Iteration 92, loss = 0.59375412
Iteration 93, loss = 0.59273064
Iteration 94, loss = 0.59184059
Iteration 95, loss = 0.59081138
Iteration 96, loss = 0.58986070
Iteration 97, loss = 0.58895414
Iteration 98, loss = 0.58803148
Iteration 99, loss = 0.58720095
Iteration 100, loss = 0.58639768
Iteration 101, loss = 0.58566520
Iteration 102, loss = 0.58480394
Iteration 103, loss = 0.58396743
Iteration 104, loss = 0.58311079
Iteration 105, loss = 0.58218479
Iteration 106, loss = 0.58131984
Iteration 107, loss = 0.58030548
Iteration 108, loss = 0.57940531
Iteration 109, loss = 0.57852096
Iteration 110, loss = 0.57765768
Iteration 111, loss = 0.57678621
Iteration 112, loss = 0.57594360
Iteration 113, loss = 0.57516423
Iteration 114, loss = 0.57434054
Iteration 115, loss = 0.57349075
Iteration 116, loss = 0.57262369
Iteration 117, loss = 0.57178356
Iteration 118, loss = 0.57092639
Iteration 119, loss = 0.57000301
Iteration 120, loss = 0.56912894
Iteration 121, loss = 0.56819983
Iteration 122, loss = 0.56732186
Iteration 123, loss = 0.56641886
Iteration 124, loss = 0.56553219
Iteration 125, loss = 0.56463388
Iteration 126, loss = 0.56373849
Iteration 127, loss = 0.56284916
Iteration 128, loss = 0.56196452
Iteration 129, loss = 0.56106948
Iteration 130, loss = 0.56012847
Iteration 131, loss = 0.55927766
Iteration 132, loss = 0.55837685
Iteration 133, loss = 0.55744953
Iteration 134, loss = 0.55648965
Iteration 135, loss = 0.55559630
Iteration 136, loss = 0.55474651
Iteration 137, loss = 0.55383882
Iteration 138, loss = 0.55301300
Iteration 139, loss = 0.55215296
Iteration 140, loss = 0.55129363
Iteration 141, loss = 0.55047143
Iteration 142, loss = 0.54963620
Iteration 143, loss = 0.54880013
Iteration 144, loss = 0.54794028
Iteration 145, loss = 0.54726869
Iteration 146, loss = 0.54646917
Iteration 147, loss = 0.54555498
Iteration 148, loss = 0.54472618
Iteration 149, loss = 0.54385105
Iteration 150, loss = 0.54298363
Iteration 151, loss = 0.54204561
Iteration 152, loss = 0.54122877
Iteration 153, loss = 0.54028929
Iteration 154, loss = 0.53942054
Iteration 155, loss = 0.53848421
Iteration 156, loss = 0.53762540
Iteration 157, loss = 0.53673801
Iteration 158, loss = 0.53593470
Iteration 159, loss = 0.53510925
Iteration 160, loss = 0.53430467
Iteration 161, loss = 0.53353336
Iteration 162, loss = 0.53268929
Iteration 163, loss = 0.53190212
Iteration 164, loss = 0.53112895
Iteration 165, loss = 0.53039681
Iteration 166, loss = 0.52959078
Iteration 167, loss = 0.52880614
Iteration 168, loss = 0.52803688
Iteration 169, loss = 0.52733095
Iteration 170, loss = 0.52661256
Iteration 171, loss = 0.52587978
Iteration 172, loss = 0.52514577
Iteration 173, loss = 0.52441622
Iteration 174, loss = 0.52368535
Iteration 175, loss = 0.52296430
Iteration 176, loss = 0.52220082
Iteration 177, loss = 0.52151851
Iteration 178, loss = 0.52069986
Iteration 179, loss = 0.51992858
Iteration 180, loss = 0.51915478
Iteration 181, loss = 0.51837019
Iteration 182, loss = 0.51757808
Iteration 183, loss = 0.51681081
Iteration 184, loss = 0.51606932
Iteration 185, loss = 0.51532145
Iteration 186, loss = 0.51477541
Iteration 187, loss = 0.51409702
Iteration 188, loss = 0.51345898
Iteration 189, loss = 0.51275297
Iteration 190, loss = 0.51199067
Iteration 191, loss = 0.51130709
Iteration 192, loss = 0.51071877
Iteration 193, loss = 0.50993374
Iteration 194, loss = 0.50936680
Iteration 195, loss = 0.50864649
Iteration 196, loss = 0.50801561
Iteration 197, loss = 0.50726639
Iteration 198, loss = 0.50677676
Iteration 199, loss = 0.50603849
Iteration 200, loss = 0.50534178
```

```
Iteration 201, loss = 0.50457844
Iteration 202, loss = 0.50382864
Iteration 203, loss = 0.50312901
Iteration 204, loss = 0.50247024
Iteration 205, loss = 0.50182656
Iteration 206, loss = 0.50116788
Iteration 207, loss = 0.50046622
Iteration 208, loss = 0.49985716
Iteration 209, loss = 0.49915846
Iteration 210, loss = 0.49841535
Iteration 211, loss = 0.49773902
Iteration 212, loss = 0.49707877
Iteration 213, loss = 0.49637612
Iteration 214, loss = 0.49574061
Iteration 215, loss = 0.49516761
Iteration 216, loss = 0.49453746
Iteration 217, loss = 0.49393185
Iteration 218, loss = 0.49336456
Iteration 219, loss = 0.49280242
Iteration 220, loss = 0.49228318
Iteration 221, loss = 0.49170774
Iteration 222, loss = 0.49112472
Iteration 223, loss = 0.49055076
Iteration 224, loss = 0.49003790
Iteration 225, loss = 0.48946114
Iteration 226, loss = 0.48890250
Iteration 227, loss = 0.48835571
Iteration 228, loss = 0.48778670
Iteration 229, loss = 0.48719198
Iteration 230, loss = 0.48664281
Iteration 231, loss = 0.48612420
Iteration 232, loss = 0.48549137
Iteration 233, loss = 0.48492709
Iteration 234, loss = 0.48443499
Iteration 235, loss = 0.48389081
Iteration 236, loss = 0.48332985
Iteration 237, loss = 0.48281695
Iteration 238, loss = 0.48240756
Iteration 239, loss = 0.48192689
Iteration 240, loss = 0.48153232
Iteration 241, loss = 0.48095737
Iteration 242, loss = 0.48024439
Iteration 243, loss = 0.47963806
Iteration 244, loss = 0.47901999
Iteration 245, loss = 0.47849317
Iteration 246, loss = 0.47791143
Iteration 247, loss = 0.47734184
Iteration 248, loss = 0.47681554
Iteration 249, loss = 0.47626487
Iteration 250, loss = 0.47574039
Iteration 251, loss = 0.47522968
Iteration 252, loss = 0.47465145
Iteration 253, loss = 0.47405343
Iteration 254, loss = 0.47355127
Iteration 255, loss = 0.47288899
Iteration 256, loss = 0.47230032
Iteration 257, loss = 0.47177451
Iteration 258, loss = 0.47124410
Iteration 259, loss = 0.47068875
Iteration 260, loss = 0.47017575
Iteration 261, loss = 0.46958078
Iteration 262, loss = 0.46904553
Iteration 263, loss = 0.46853884
Iteration 264, loss = 0.46794661
Iteration 265, loss = 0.46737747
Iteration 266, loss = 0.46682827
Iteration 267, loss = 0.46628147
Iteration 268, loss = 0.46573563
Iteration 269, loss = 0.46516194
Iteration 270, loss = 0.46462158
Iteration 271, loss = 0.46408491
Iteration 272, loss = 0.46358195
Iteration 273, loss = 0.46303148
Iteration 274, loss = 0.46250709
Iteration 275, loss = 0.46190492
Iteration 276, loss = 0.46135675
Iteration 277, loss = 0.46084416
Iteration 278, loss = 0.46063763
Iteration 279, loss = 0.45999908
Iteration 280, loss = 0.45950645
Iteration 281, loss = 0.45916439
Iteration 282, loss = 0.45849614
Iteration 283, loss = 0.45807065
Iteration 284, loss = 0.45768596
Iteration 285, loss = 0.45723206
Iteration 286, loss = 0.45683764
Iteration 287, loss = 0.45646595
Iteration 288, loss = 0.45600153
Iteration 289, loss = 0.45564781
Iteration 290, loss = 0.45518251
Iteration 291, loss = 0.45477580
Iteration 292, loss = 0.45439748
Iteration 293, loss = 0.45415316
Iteration 294, loss = 0.45376944
Iteration 295, loss = 0.45338249
Iteration 296, loss = 0.45289465
Iteration 297, loss = 0.45261935
Iteration 298, loss = 0.45214469
Iteration 299, loss = 0.45183936
Iteration 300, loss = 0.45148377
Iteration 301, loss = 0.45113658
Iteration 302, loss = 0.45083229
Iteration 303, loss = 0.45038318
Iteration 304, loss = 0.45004849
Iteration 305, loss = 0.44963614
Iteration 306, loss = 0.44935071
Iteration 307, loss = 0.44917969
Iteration 308, loss = 0.44891202
Iteration 309, loss = 0.44855504
Iteration 310, loss = 0.44819727
Iteration 311, loss = 0.44778967
Iteration 312, loss = 0.44747876
Iteration 313, loss = 0.44708686
Iteration 314, loss = 0.44671619
Iteration 315, loss = 0.44634845
Iteration 316, loss = 0.44601998
Iteration 317, loss = 0.44565224
Iteration 318, loss = 0.44532740
Iteration 319, loss = 0.44503496
Iteration 320, loss = 0.44464912
Iteration 321, loss = 0.44430055
Iteration 322, loss = 0.44398797
Iteration 323, loss = 0.44367171
Iteration 324, loss = 0.44336646
Iteration 325, loss = 0.44309965
Iteration 326, loss = 0.44279152
Iteration 327, loss = 0.44248893
Iteration 328, loss = 0.44218599
Iteration 329, loss = 0.44190429
Iteration 330, loss = 0.44160063
Iteration 331, loss = 0.44126878
Iteration 332, loss = 0.44093905
Iteration 333, loss = 0.44057943
Iteration 334, loss = 0.44052668
Iteration 335, loss = 0.44019076
Iteration 336, loss = 0.43993452
Iteration 337, loss = 0.43971455
Iteration 338, loss = 0.43943383
Iteration 339, loss = 0.43911315
Iteration 340, loss = 0.43890525
Iteration 341, loss = 0.43853740
Iteration 342, loss = 0.43826695
Iteration 343, loss = 0.43801185
```

```
Iteration 344, loss = 0.43773823
Iteration 345, loss = 0.43764324
Iteration 346, loss = 0.43738410
Iteration 347, loss = 0.43712365
Iteration 348, loss = 0.43673119
Iteration 349, loss = 0.43643526
Iteration 350, loss = 0.43628774
Iteration 351, loss = 0.43608502
Iteration 352, loss = 0.43576578
Iteration 353, loss = 0.43549738
Iteration 354, loss = 0.43524737
Iteration 355, loss = 0.43502232
Iteration 356, loss = 0.43478042
Iteration 357, loss = 0.43452003
Iteration 358, loss = 0.43427209
Iteration 359, loss = 0.43404622
Iteration 360, loss = 0.43383070
Iteration 361, loss = 0.43371114
Iteration 362, loss = 0.43356374
Iteration 363, loss = 0.43346516
Iteration 364, loss = 0.43347868
Iteration 365, loss = 0.43342343
Iteration 366, loss = 0.43316355
Iteration 367, loss = 0.43278704
Iteration 368, loss = 0.43234796
Iteration 369, loss = 0.43216271
Iteration 370, loss = 0.43201406
Iteration 371, loss = 0.43204092
Iteration 372, loss = 0.43181989
Iteration 373, loss = 0.43162365
Iteration 374, loss = 0.43133537
Iteration 375, loss = 0.43111643
Iteration 376, loss = 0.43073087
Iteration 377, loss = 0.43069928
Iteration 378, loss = 0.43045395
Iteration 379, loss = 0.43047756
Iteration 380, loss = 0.43018321
Iteration 381, loss = 0.42996842
Iteration 382, loss = 0.42968880
Iteration 383, loss = 0.42947653
Iteration 384, loss = 0.42931118
Iteration 385, loss = 0.42910927
Iteration 386, loss = 0.42891463
Iteration 387, loss = 0.42870224
Iteration 388, loss = 0.42850704
Iteration 389, loss = 0.42827648
Iteration 390, loss = 0.42809588
Iteration 391, loss = 0.42791389
Iteration 392, loss = 0.42775609
Iteration 393, loss = 0.42761506
Iteration 394, loss = 0.42740677
Iteration 395, loss = 0.42733314
Iteration 396, loss = 0.42730295
Iteration 397, loss = 0.42718286
Iteration 398, loss = 0.42706613
Iteration 399, loss = 0.42694214
Iteration 400, loss = 0.42683061
Iteration 401, loss = 0.42662102
Iteration 402, loss = 0.42641007
Iteration 403, loss = 0.42615385
Iteration 404, loss = 0.42603554
Iteration 405, loss = 0.42579871
Iteration 406, loss = 0.42557377
Iteration 407, loss = 0.42533663
Iteration 408, loss = 0.42518784
Iteration 409, loss = 0.42504549
Iteration 410, loss = 0.42490193
Iteration 411, loss = 0.42475478
Iteration 412, loss = 0.42462829
Iteration 413, loss = 0.42443645
Iteration 414, loss = 0.42425425
Iteration 415, loss = 0.42407510
Iteration 416, loss = 0.42386790
Iteration 417, loss = 0.42375318
Iteration 418, loss = 0.42360886
Iteration 419, loss = 0.42336748
Iteration 420, loss = 0.42322001
Iteration 421, loss = 0.42294941
Iteration 422, loss = 0.42295929
Iteration 423, loss = 0.42311115
Iteration 424, loss = 0.42314223
Iteration 425, loss = 0.42315027
Iteration 426, loss = 0.42302410
Iteration 427, loss = 0.42273331
Iteration 428, loss = 0.42228353
Iteration 429, loss = 0.42204810
Iteration 430, loss = 0.42179575
Iteration 431, loss = 0.42161986
Iteration 432, loss = 0.42145778
Iteration 433, loss = 0.42140220
Iteration 434, loss = 0.42122949
Iteration 435, loss = 0.42107235
Iteration 436, loss = 0.42099715
Iteration 437, loss = 0.42093680
Iteration 438, loss = 0.42086352
Iteration 439, loss = 0.42077682
Iteration 440, loss = 0.42063310
Iteration 441, loss = 0.42044577
Iteration 442, loss = 0.42021880
Iteration 443, loss = 0.42001301
Iteration 444, loss = 0.41986532
Iteration 445, loss = 0.41994414
Iteration 446, loss = 0.41970742
Iteration 447, loss = 0.41955934
Iteration 448, loss = 0.41954258
Iteration 449, loss = 0.41940144
Iteration 450, loss = 0.41916531
Iteration 451, loss = 0.41905341
Iteration 452, loss = 0.41880933
Iteration 453, loss = 0.41866694
Iteration 454, loss = 0.41853389
Iteration 455, loss = 0.41845020
Iteration 456, loss = 0.41836422
Iteration 457, loss = 0.41826780
Iteration 458, loss = 0.41811086
Iteration 459, loss = 0.41811692
Iteration 460, loss = 0.41786621
Iteration 461, loss = 0.41782550
Iteration 462, loss = 0.41768694
Iteration 463, loss = 0.41757914
Iteration 464, loss = 0.41744073
Iteration 465, loss = 0.41754857
Iteration 466, loss = 0.41737648
Iteration 467, loss = 0.41722208
Iteration 468, loss = 0.41701891
Iteration 469, loss = 0.41691421
Iteration 470, loss = 0.41678457
Iteration 471, loss = 0.41666324
Iteration 472, loss = 0.41656300
Iteration 473, loss = 0.41655838
Iteration 474, loss = 0.41646293
Iteration 475, loss = 0.41637570
Iteration 476, loss = 0.41621968
Iteration 477, loss = 0.41607721
Iteration 478, loss = 0.41597392
Iteration 479, loss = 0.41589296
Iteration 480, loss = 0.41575577
Iteration 481, loss = 0.41565065
Iteration 482, loss = 0.41571134
Iteration 483, loss = 0.41558680
Iteration 484, loss = 0.41549532
Iteration 485, loss = 0.41524852
Iteration 486, loss = 0.41509761
Iteration 487, loss = 0.41508474
```

```
Iteration 488, loss = 0.41491745
Iteration 489, loss = 0.41498059
Iteration 490, loss = 0.41495744
Iteration 491, loss = 0.41481261
Iteration 492, loss = 0.41464447
Iteration 493, loss = 0.41451265
Iteration 494, loss = 0.41435184
Iteration 495, loss = 0.41424607
Iteration 496, loss = 0.41412778
Iteration 497, loss = 0.41402267
Iteration 498, loss = 0.41381866
Iteration 499, loss = 0.41369638
Iteration 500, loss = 0.41358236
Iteration 501, loss = 0.41348937
Iteration 502, loss = 0.41333329
Iteration 503, loss = 0.41327220
Iteration 504, loss = 0.41314830
Iteration 505, loss = 0.41307275
Iteration 506, loss = 0.41301661
Iteration 507, loss = 0.41281784
Iteration 508, loss = 0.41276124
Iteration 509, loss = 0.41268042
Iteration 510, loss = 0.41252197
Iteration 511, loss = 0.41241457
Iteration 512, loss = 0.41258023
Iteration 513, loss = 0.41256181
Iteration 514, loss = 0.41254060
Iteration 515, loss = 0.41276602
Iteration 516, loss = 0.41277606
Iteration 517, loss = 0.41269863
Iteration 518, loss = 0.41261037
Iteration 519, loss = 0.41250035
Iteration 520, loss = 0.41257467
Iteration 521, loss = 0.41248021
Iteration 522, loss = 0.41238334
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

```
▾ MLPClassifier
MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=800, random_state=3,
              solver='sgd', verbose=True)
```

```
previsoes = rna.predict(X_test)
previsoes
```

```
array([0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1])
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
print("Acurácia: %.2f%%" % (accuracy_score(y_test, previsoes) * 100.0))
```

```
Acurácia: 81.34%
```

```
confusion_matrix(y_test, previsoes)
```

```
array([[148,  20],
       [ 30,  70]])
```

```
print(classification_report(y_test, previsoes))
```

```
              precision    recall  f1-score   support

           0       0.83      0.88      0.86       168
           1       0.78      0.70      0.74       100

    accuracy                           0.81       268
   macro avg       0.80      0.79      0.80       268
weighted avg       0.81      0.81      0.81       268
```

```
previsoes_treino = rna.predict(X_train)
previsoes_treino
```

```
array([1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0])
```

```
accuracy_score(y_train, previsoes_treino)
```

```
0.8346709470304976
```

```
unique_classes_y = np.unique(y)
unique_classes_previsoes = np.unique(previsoes)
print("Classes em y:", unique_classes_y)
print("Classes em previsoes:", unique_classes_previsoes)
```

```
Classes em y: [0 1]
Classes em previsoes: [0 1]
```

```
##########################################################################################
```

```
forest = RandomForestClassifier(random_state=42)
#==
forest.fit(x_train_oversampled, y_train_oversampled)
```

```
#==
#Score
#==
forest_score = forest.score(x_train_oversampled, y_train_oversampled)
forest_test = forest.score(X_test, y_test)
#==
#testing model
#==
y_pred = forest.predict(X_test)
#==
#evaluation
#==
cm = confusion_matrix(y_test,y_pred)
print('Training Score',forest_score)
print('Testing Score \n',forest_test)
print(cm)
```

```
Training Score 0.9909326424870466
Testing Score
 0.9029850746268657
[[154  14]
 [ 12  88]]
```

```
##########################################################################################################
############################################
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier()
# param_grid = {'criterion': ['gini', 'entropy', 'log_loss'],
#               'max_features':['sqrt','log2'],
#               'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 100, 200, 300]}
param_grid = {'criterion': ['gini', 'entropy', 'log_loss'],
              'max_features':['sqrt','log2'],
              'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 100, 200, 300]}
#### Exemplo com GridSearchCV
# - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
g_search = GridSearchCV(estimator=forest, param_grid=param_grid,
                        cv=10, return_train_score=True)
sm = SMOTE()
x_train_oversampled, y_train_oversampled = sm.fit_resample(X_train, y_train)
print(np.unique(y_train_oversampled, return_counts=True))
g_search.fit(x_train_oversampled, y_train_oversampled)
print(g_search.best_params_)
print(g_search.best_score_)
```

```
(array([0, 1]), array([381, 381]))
```