IMPORTANDO BIBLIOTECAS

In [ ]:
```
pip install scikeras
```

```
Requirement already satisfied: scikeras in /usr/local/lib/python3.10/dist-packages (0.12.0)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (23.2)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.2.0)
```

In [ ]:
```python
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.layers import Dense
from keras.models import Sequential
from sklearn.compose import ColumnTransformer
from scikeras.wrappers import KerasClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.feature_selection import f_classif
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split, StratifiedKFold, learning_curve
from sklearn.metrics import confusion_matrix, recall_score, ConfusionMatrixDisplay, classification_report
```

IMPORTANDO BASE DE DADOS

In [ ]:
```python
data = pd.read_excel('e_commerce_dataset.xlsx')
data
```

Out[167]:

	CustomerID	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gender	HourSpendOnApp	NumberOfDeviceRegistered	PreferedOrderCat	Satis
0	50001	1	4.0	Mobile Phone	3	6.0	Debit Card	Female	3.0	3	Laptop & Accessory	
1	50002	1	NaN	Phone	1	8.0	UPI	Male	3.0	4	Mobile	
2	50003	1	NaN	Phone	1	30.0	Debit Card	Male	2.0	4	Mobile	
3	50004	1	0.0	Phone	3	15.0	Debit Card	Male	2.0	4	Laptop & Accessory	
4	50005	1	0.0	Phone	1	12.0	CC	Male	NaN	3	Mobile	
...	...	...	...	...	...	...	...	...	...	...	...	
5625	55626	0	10.0	Computer	1	30.0	Credit Card	Male	3.0	2	Laptop & Accessory	
5626	55627	0	13.0	Mobile Phone	1	13.0	Credit Card	Male	3.0	5	Fashion	
5627	55628	0	1.0	Mobile Phone	1	11.0	Debit Card	Male	3.0	2	Laptop & Accessory	
5628	55629	0	23.0	Computer	3	9.0	Credit Card	Male	4.0	5	Laptop & Accessory	
5629	55630	0	8.0	Mobile Phone	1	15.0	Credit Card	Male	3.0	2	Laptop & Accessory	

5630 rows × 20 columns

ELIMINANDO VALORES NULOS

In [ ]:
```python
data = data.dropna()
data
```

Out[168]:

	CustomerID	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gender	HourSpendOnApp	NumberOfDeviceRegistered	PreferedOrderCat	Satis
0	50001	1	4.0	Mobile Phone	3	6.0	Debit Card	Female	3.0	3	Laptop & Accessory	
3	50004	1	0.0	Phone	3	15.0	Debit Card	Male	2.0	4	Laptop & Accessory	
5	50006	1	0.0	Computer	1	22.0	Debit Card	Female	3.0	5	Mobile Phone	
11	50012	1	11.0	Mobile Phone	1	6.0	Debit Card	Male	3.0	4	Fashion	
12	50013	1	0.0	Phone	1	11.0	COD	Male	2.0	3	Mobile	
...	...	...	...	...	...	...	...	...	...	...	...	
5624	55625	0	1.0	Mobile Phone	3	12.0	UPI	Female	2.0	5	Mobile Phone	
5625	55626	0	10.0	Computer	1	30.0	Credit Card	Male	3.0	2	Laptop & Accessory	
5627	55628	0	1.0	Mobile Phone	1	11.0	Debit Card	Male	3.0	2	Laptop & Accessory	
5628	55629	0	23.0	Computer	3	9.0	Credit Card	Male	4.0	5	Laptop & Accessory	
5629	55630	0	8.0	Mobile Phone	1	15.0	Credit Card	Male	3.0	2	Laptop & Accessory	

3774 rows × 20 columns

ELIMINANDO A COLUNA "CustomerID", VISTO QUE ELA NÃO POSSUI IMPACTO NA PREDIÇÃO

In [ ]:
```
data = data.drop("CustomerID", axis = 1)
data
```

Out[169]:

	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gender	HourSpendOnApp	NumberOfDeviceRegistered	PreferedOrderCat	SatisfactionScore
0	1	4.0	Mobile Phone	3	6.0	Debit Card	Female	3.0	3	Laptop & Accessory	2
3	1	0.0	Phone	3	15.0	Debit Card	Male	2.0	4	Laptop & Accessory	5
5	1	0.0	Computer	1	22.0	Debit Card	Female	3.0	5	Mobile Phone	5
11	1	11.0	Mobile Phone	1	6.0	Debit Card	Male	3.0	4	Fashion	3
12	1	0.0	Phone	1	11.0	COD	Male	2.0	3	Mobile	3
...	...	...	...	...	...	...	...	...	...	...	...
5624	0	1.0	Mobile Phone	3	12.0	UPI	Female	2.0	5	Mobile Phone	3
5625	0	10.0	Computer	1	30.0	Credit Card	Male	3.0	2	Laptop & Accessory	1
5627	0	1.0	Mobile Phone	1	11.0	Debit Card	Male	3.0	2	Laptop & Accessory	4
5628	0	23.0	Computer	3	9.0	Credit Card	Male	4.0	5	Laptop & Accessory	4
5629	0	8.0	Mobile Phone	1	15.0	Credit Card	Male	3.0	2	Laptop & Accessory	3

3774 rows × 19 columns

DIVIDINDO AS VARIÁVEIS DE ENTRADA E SAÍDA

In [ ]:
```
y = data.iloc[:,0]
x = data.iloc[:,1:]
x
```

Out[170]:

	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gender	HourSpendOnApp	NumberOfDeviceRegistered	PreferedOrderCat	SatisfactionScore	MaritalS
0	4.0	Mobile Phone	3	6.0	Debit Card	Female	3.0	3	Laptop & Accessory	2	:
3	0.0	Phone	3	15.0	Debit Card	Male	2.0	4	Laptop & Accessory	5	:
5	0.0	Computer	1	22.0	Debit Card	Female	3.0	5	Mobile Phone	5	:
11	11.0	Mobile Phone	1	6.0	Debit Card	Male	3.0	4	Fashion	3	:
12	0.0	Phone	1	11.0	COD	Male	2.0	3	Mobile	3	:
...	...	...	...	...	...	...	...	...	...	...	
5624	1.0	Mobile Phone	3	12.0	UPI	Female	2.0	5	Mobile Phone	3	:
5625	10.0	Computer	1	30.0	Credit Card	Male	3.0	2	Laptop & Accessory	1	M
5627	1.0	Mobile Phone	1	11.0	Debit Card	Male	3.0	2	Laptop & Accessory	4	M
5628	23.0	Computer	3	9.0	Credit Card	Male	4.0	5	Laptop & Accessory	4	M
5629	8.0	Mobile Phone	1	15.0	Credit Card	Male	3.0	2	Laptop & Accessory	3	M

3774 rows × 18 columns

TRANSFORMANDO DADOS CATEGÓRICOS EM NUMÉRICOS

In [ ]:
```
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1, 4, 5, 8, 10])],
                       remainder='passthrough')
```

In [ ]:
```
x = ct.fit_transform(x)
x
pd.DataFrame(x)
```

Out[172]:

	0	1	2	3	4	5	6	7	8	9	...	24	25	26	27	28	29	30	31	32	33
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	3.0	3.0	2.0	9.0	1.0	11.0	1.0	1.0	5.0	159.93
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	2.0	4.0	5.0	8.0	0.0	23.0	0.0	1.0	3.0	134.07
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	3.0	5.0	5.0	2.0	1.0	22.0	4.0	6.0	7.0	139.19
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	3.0	4.0	3.0	10.0	1.0	13.0	0.0	1.0	0.0	153.81
4	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	2.0	3.0	3.0	2.0	1.0	13.0	2.0	2.0	2.0	134.41
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3769	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	2.0	5.0	3.0	2.0	0.0	19.0	2.0	2.0	1.0	154.66
3770	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	3.0	2.0	1.0	6.0	0.0	18.0	1.0	2.0	4.0	150.71
3771	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	3.0	2.0	4.0	3.0	1.0	21.0	1.0	2.0	4.0	186.42
3772	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	4.0	5.0	4.0	4.0	0.0	15.0	2.0	2.0	9.0	178.90
3773	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	3.0	2.0	3.0	4.0	0.0	13.0	2.0	2.0	3.0	169.04

3774 rows × 34 columns

In [ ]:
```
colunas = ["PreferredLoginDevice1", "PreferredLoginDevice2", "PreferredLoginDevice3", "PreferredPaymentMode1",  "PreferredPaymentMode2",
           "PreferredPaymentMode3", "PreferredPaymentMode4", "PreferredPaymentMode5", "PreferredPaymentMode6", "PreferredPaymentMode7",
           "Gender1", "Gender2",    "PreferedOrderCat1", "PreferedOrderCat2",   "PreferedOrderCat3",    "PreferedOrderCat4",    "PreferedOrderCat5",
           "PreferedOrderCat6", "MaritalStatus1",   "MaritalStatus2", "MaritalStatus3", "Tenure", "CityTier",   "WarehouseToHome",
           "HourSpendOnApp",    "NumberOfDeviceRegistered", "SatisfactionScore", "NumberOfAddress",
           "Complain",  "OrderAmountHikeFromlastYear",  "CouponUsed",    "OrderCount",    "DaySinceLastOrder",    "CashbackAmount"]
x = pd.DataFrame(x, columns=colunas)
x
```

Out[173]:

	PreferredLoginDevice1	PreferredLoginDevice2	PreferredLoginDevice3	PreferredPaymentMode1	PreferredPaymentMode2	PreferredPaymentMode3	PreferredPaymentMode4	PreferredPaym
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	1.0	0.0	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	
3769	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
3770	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
3771	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
3772	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
3773	0.0	1.0	0.0	0.0	0.0	0.0	1.0	

3774 rows × 34 columns

SELECIONANDO FEATURES

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
model = RandomForestClassifier(n_estimators=50)
model.fit(x,y)
```

Out[175]:
```
RandomForestClassifier(n_estimators=50)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [ ]:
```
feature_importance = pd.DataFrame (model.feature_importances_, index = x.columns, columns = ['importance']).sort_values('importance',
                                                                                                                         ascending=False)
feature_importance
```

Out[176]:

	importance
Tenure	0.208650
WarehouseToHome	0.077524
CashbackAmount	0.075486
DaySinceLastOrder	0.063636
NumberOfAddress	0.062370
OrderAmountHikeFromlastYear	0.055737
Complain	0.055156
SatisfactionScore	0.045150
NumberOfDeviceRegistered	0.038459
OrderCount	0.034037
CouponUsed	0.025859
CityTier	0.022629
MaritalStatus3	0.021763
HourSpendOnApp	0.021440
PreferedOrderCat5	0.017700
PreferedOrderCat3	0.017400
MaritalStatus2	0.016538
Gender2	0.015582
PreferredPaymentMode5	0.014542
PreferredPaymentMode4	0.014481
PreferredLoginDevice1	0.014164
PreferredLoginDevice2	0.013591
Gender1	0.013044
PreferredPaymentMode2	0.009707
PreferredPaymentMode6	0.009013
PreferredLoginDevice3	0.008804
PreferedOrderCat1	0.008216
PreferredPaymentMode7	0.007009
MaritalStatus1	0.005461
PreferedOrderCat4	0.002614
PreferredPaymentMode1	0.001497
PreferredPaymentMode3	0.001487
PreferedOrderCat6	0.001217
PreferedOrderCat2	0.000034

FEATURES COM GRAUS DE IMPORTÂNCIAS SUPERIORES À 0,05 (5%)

In [ ]:
```
best_feature_importance = feature_importance[feature_importance ['importance'] >=0.05]
best_feature_importance
```

Out[177]:

	importance
Tenure	0.208650
WarehouseToHome	0.077524
CashbackAmount	0.075486
DaySinceLastOrder	0.063636
NumberOfAddress	0.062370
OrderAmountHikeFromlastYear	0.055737
Complain	0.055156

NOVO DATAFRAME DE X COM AS MELHORES FEATURES

In [ ]:
```
x_best = x[['Tenure', 'CashbackAmount', 'WarehouseToHome', 'NumberOfAddress', 'OrderAmountHikeFromlastYear',
            'DaySinceLastOrder','Complain']]
x_best
```

Out[178]:

	Tenure	CashbackAmount	WarehouseToHome	NumberOfAddress	OrderAmountHikeFromlastYear	DaySinceLastOrder	Complain
0	4.0	159.93	6.0	9.0	11.0	5.0	1.0
1	0.0	134.07	15.0	8.0	23.0	3.0	0.0
2	0.0	139.19	22.0	2.0	22.0	7.0	1.0
3	11.0	153.81	6.0	10.0	13.0	0.0	1.0
4	0.0	134.41	11.0	2.0	13.0	2.0	1.0
...	...	...	...	...	...	...	...
3769	1.0	154.66	12.0	2.0	19.0	1.0	0.0
3770	10.0	150.71	30.0	6.0	18.0	4.0	0.0
3771	1.0	186.42	11.0	3.0	21.0	4.0	1.0
3772	23.0	178.90	9.0	4.0	15.0	9.0	0.0
3773	8.0	169.04	15.0	4.0	13.0	3.0	0.0

3774 rows × 7 columns

DIVIDINDO DADOS DE TREINO E TESTE

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3, stratify = y)
```

In [ ]:
```
x_train
```

Out[180]:

	PreferredLoginDevice1	PreferredLoginDevice2	PreferredLoginDevice3	PreferredPaymentMode1	PreferredPaymentMode2	PreferredPaymentMode3	PreferredPaymentMode4	PreferredPaym
419	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
1604	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
1500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
1458	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
2383	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
1832	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
3520	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
2047	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
2153	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
1819	0.0	1.0	0.0	0.0	0.0	0.0	0.0	

2641 rows × 34 columns

In [ ]:
```
y_train
```

Out[181]:
```
753     0
2779    0
2602    1
2541    0
3820    0
       ..
3081    0
5311    0
3358    0
3509    0
3066    0
Name: Churn, Length: 2641, dtype: int64
```

NORMALIZANDO DADOS

In [ ]:
```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [ ]:
```
x_train
```

Out[183]:
```
array([[ 1.56597815, -1.03271979, -0.49099025, ..., -0.73516934,
        -0.4751971 , -0.08341337],
       [ 1.56597815, -1.03271979, -0.49099025, ..., -0.73516934,
        -1.38858541, -1.24077633],
       [-0.63857851,  0.96831687, -0.49099025, ..., -0.73516934,
        -1.38858541,  0.70019188],
       ...,
       [-0.63857851, -1.03271979,  2.03670031, ...,  0.06730317,
        -0.4751971 , -0.66209485],
       [-0.63857851,  0.96831687, -0.49099025, ..., -0.33393309,
         0.13372844, -0.32027869],
       [-0.63857851,  0.96831687, -0.49099025, ..., -0.33393309,
         1.04711675,  0.39944344]])
```

In [ ]:
```
x_test
```

Out[184]:
```
array([[-0.63857851,  0.96831687, -0.49099025, ...,  0.06730317,
        -0.4751971 ,  0.14266649],
       [-0.63857851,  0.96831687, -0.49099025, ..., -0.33393309,
        -0.17073433,  0.72259245],
       [-0.63857851,  0.96831687, -0.49099025, ..., -0.33393309,
        -0.17073433, -0.21615751],
       ...,
       [-0.63857851,  0.96831687, -0.49099025, ..., -0.73516934,
         0.74265398, -0.62766434],
       [ 1.56597815, -1.03271979, -0.49099025, ..., -0.73516934,
        -0.4751971 , -1.23330947],
       [-0.63857851,  0.96831687, -0.49099025, ...,  2.07348444,
         1.04711675,  1.52693897]])
```

SELEÇÃO DO NÚMERO DE NEURÔNIOS DA CAMADA DE PROCESSAMENTO

In [ ]:
```
def Return_Recall(x_test, y_test):
    y_pred = ann.predict(x_test)
    y_pred = (y_pred > 0.5)

    score = recall_score(y_test, y_pred)*100
    list_ReturnRecall.append (score)

    return print("Recall com a validação (%): ", ((score)))
```

In [ ]:
```
list_ReturnRecall = []
list_neurons = [2,4,6,8]

for i in list_neurons:
    ann = Sequential()

    ann.add (tf.keras.layers.Dense (units=i, activation='relu', kernel_initializer = 'he_normal'))
    ann.add (tf.keras.layers.Dense (units=1, activation='sigmoid', kernel_initializer = 'he_normal'))

    optimize = tf.keras.optimizers.Adam(learning_rate=0.01)
    ann.compile(optimizer=optimize, loss='binary_crossentropy', metrics=[tf.keras.metrics.Recall()])

    ann.fit(x_train, y_train, batch_size=32, epochs=50)

    Return_Recall(x_test, y_test)
```

```
Epoch 1/50
83/83 [==============================] - 1s 2ms/step - loss: 0.4979 - recall_31: 0.3032
Epoch 2/50
83/83 [==============================] - 0s 2ms/step - loss: 0.3235 - recall_31: 0.4367
Epoch 3/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2861 - recall_31: 0.5452
Epoch 4/50
83/83 [==============================] - 0s 3ms/step - loss: 0.2734 - recall_31: 0.5679
Epoch 5/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2723 - recall_31: 0.5701
Epoch 6/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2692 - recall_31: 0.5656
Epoch 7/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2649 - recall_31: 0.5588
Epoch 8/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2666 - recall_31: 0.5882
Epoch 9/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2625 - recall_31: 0.5317
Epoch 10/50
```

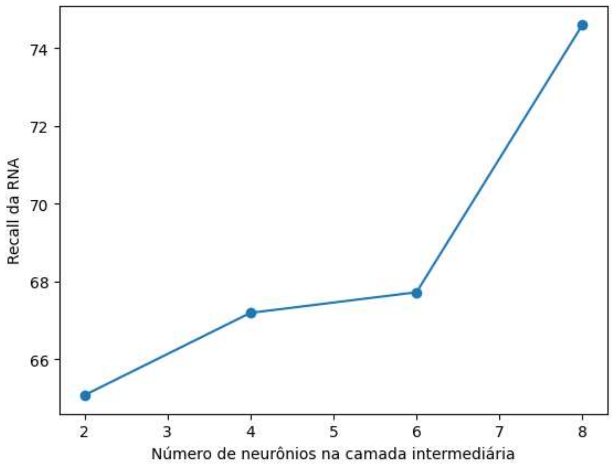
RECALL PARA CADA QUANTIDADE DE NEURÔNIOS NA CAMADA INTERMEDIÁRIA

In [ ]:
```
df_neurons = pd.DataFrame (list_ReturnRecall, index = list_neurons, columns = ['Recall']).sort_values('Recall',
                                                                                                       ascending=False)
df_neurons
```

Out[187]:

	Recall
8	74.603175
6	67.724868
4	67.195767
2	65.079365

In [ ]:
```
plt.plot(list_neurons, list_ReturnRecall, marker = 'o')
plt.xlabel ('Número de neurônios na camada intermediária')
plt.ylabel ('Recall da RNA')
plt.show()
```



TREINO DA RNA

In [ ]:
```
ann = Sequential()

ann.add (tf.keras.layers.Dense (units=8, activation='relu', kernel_initializer = 'he_normal'))
ann.add (tf.keras.layers.Dense (units=1, activation='sigmoid', kernel_initializer = 'he_normal'))

optimize = tf.keras.optimizers.Adam(learning_rate=0.01)
ann.compile(optimizer=optimize, loss='binary_crossentropy', metrics=[tf.keras.metrics.Recall()])

ann.fit(x_train, y_train, batch_size=32, epochs=50)
```

```
Epoch 1/50
83/83 [==============================] - 1s 2ms/step - loss: 0.3709 - recall_35: 0.2149
Epoch 2/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2713 - recall_35: 0.5271
Epoch 3/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2538 - recall_35: 0.5905
Epoch 4/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2434 - recall_35: 0.5928
Epoch 5/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2405 - recall_35: 0.5905
Epoch 6/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2316 - recall_35: 0.6244
Epoch 7/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2232 - recall_35: 0.6086
Epoch 8/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2139 - recall_35: 0.6584
Epoch 9/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2102 - recall_35: 0.6403
Epoch 10/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2060 - recall_35: 0.6810
Epoch 11/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2014 - recall_35: 0.6697
Epoch 12/50
83/83 [==============================] - 0s 2ms/step - loss: 0.2003 - recall_35: 0.7014
Epoch 13/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1934 - recall_35: 0.6968
Epoch 14/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1930 - recall_35: 0.7036
Epoch 15/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1891 - recall_35: 0.7262
Epoch 16/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1898 - recall_35: 0.7014
Epoch 17/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1835 - recall_35: 0.7059
Epoch 18/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1829 - recall_35: 0.7443
Epoch 19/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1840 - recall_35: 0.7398
Epoch 20/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1767 - recall_35: 0.7353
Epoch 21/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1778 - recall_35: 0.7330
Epoch 22/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1766 - recall_35: 0.7398
Epoch 23/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1795 - recall_35: 0.7285
Epoch 24/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1742 - recall_35: 0.7240
Epoch 25/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1733 - recall_35: 0.7421
Epoch 26/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1712 - recall_35: 0.7421
Epoch 27/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1714 - recall_35: 0.7353
Epoch 28/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1690 - recall_35: 0.7421
Epoch 29/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1697 - recall_35: 0.7466
Epoch 30/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1663 - recall_35: 0.7692
Epoch 31/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1655 - recall_35: 0.7557
Epoch 32/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1625 - recall_35: 0.7489
Epoch 33/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1652 - recall_35: 0.7534
Epoch 34/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1637 - recall_35: 0.7308
Epoch 35/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1622 - recall_35: 0.7511
Epoch 36/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1584 - recall_35: 0.7715
Epoch 37/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1574 - recall_35: 0.7466
Epoch 38/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1550 - recall_35: 0.7670
Epoch 39/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1567 - recall_35: 0.7466
Epoch 40/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1509 - recall_35: 0.7557
Epoch 41/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1569 - recall_35: 0.7602
Epoch 42/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1549 - recall_35: 0.7760
Epoch 43/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1533 - recall_35: 0.7624
Epoch 44/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1510 - recall_35: 0.7647
Epoch 45/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1524 - recall_35: 0.7602
Epoch 46/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1541 - recall_35: 0.7557
Epoch 47/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1515 - recall_35: 0.7670
Epoch 48/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1476 - recall_35: 0.7624
Epoch 49/50
83/83 [==============================] - 0s 3ms/step - loss: 0.1495 - recall_35: 0.7715
Epoch 50/50
83/83 [==============================] - 0s 2ms/step - loss: 0.1463 - recall_35: 0.7919
```

Out[189]: <keras.src.callbacks.History at 0x78e453d94190>

PREDIÇÃO DA RNA E RESULTADOS

In [ ]:
```
y_pred = ann.predict(x_test)
y_pred = (y_pred > 0.5)

pred_array = 1 * y_pred.reshape(len(y_pred), 1)
test_array = y_test.values.reshape(len(y_test), 1)
```

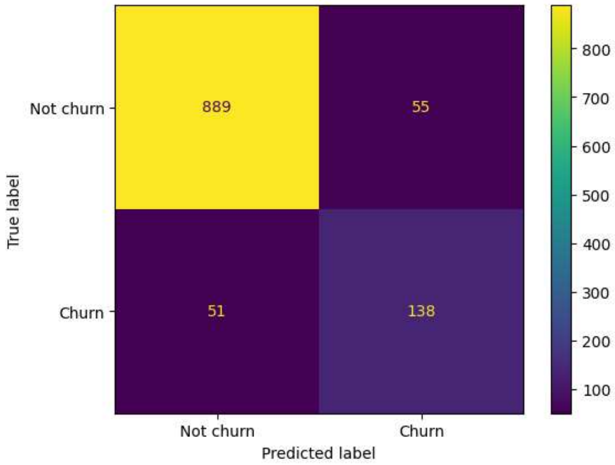
```
36/36 [==============================] - 0s 2ms/step
```

In [ ]:
```
cm = confusion_matrix(test_array, pred_array)

cm_display = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = ["Not churn", "Churn"])

cm_display.plot()
plt.show()

print("Recall da validação (%): ", ((recall_score(y_test, y_pred)*100)))
```




```
Recall da validação (%):  73.01587301587301
```

Utilizou-se o "Recall" como métrica de avaliação desta predição pelo fato deste indicador ter uma maior efetividade no estudo dos falsos negativos. A aparição de falsos negativos é bastante prejudicial nesta ocasião, dado que o erro na previsão da não saída de clientes pode ocasionar prejuízos financeiros significativos para a instituição. O valor de Recall obtido foi satisfatório. Para averiguar o impacto do tamanho do banco de dados no valor do Recall, será construída uma curva de aprendizagem a seguir.

CURVA DE APRENDIZAGEM

In [ ]:
```
classif = MLPClassifier(hidden_layer_sizes=(8,), max_iter=5000, activation='relu', random_state=42,
                        solver = 'adam', batch_size = 20,verbose=1)
```

In [ ]:
```
train_sizes_abs, train_scores, test_scores = learning_curve(classif, x, y, train_sizes=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],
                                                            scoring = 'recall_macro', cv=5)
```

```
Iteration 1, loss = 0.72517385
Iteration 2, loss = 0.66288144
Iteration 3, loss = 0.60861482
Iteration 4, loss = 0.56322066
Iteration 5, loss = 0.52666802
Iteration 6, loss = 0.49523238
Iteration 7, loss = 0.46140777
Iteration 8, loss = 0.44156267
Iteration 9, loss = 0.41771109
Iteration 10, loss = 0.39612568
Iteration 11, loss = 0.35667143
Iteration 12, loss = 0.33396216
Iteration 13, loss = 0.38038834
Iteration 14, loss = 0.31519656
Iteration 15, loss = 0.30037974
Iteration 16, loss = 0.29951427
Iteration 17, loss = 0.30791356
Iteration 18, loss = 0.32252870
Iteration 19, loss = 0.25925532
```

In [ ]:
```
train_sizes_abs
```

Out[160]:
```
array([ 301,  603,  905, 1207, 1509, 1811, 2113, 2415, 2717])
```

In [ ]:
```
train_scores
```

Out[161]:
```
array([[0.78717221, 0.7134902 , 0.7134902 , 0.7134902 , 0.7134902 ],
       [0.77597159, 0.8082859 , 0.8082859 , 0.8082859 , 0.8082859 ],
       [0.74993242, 0.77655709, 0.69240019, 0.69240019, 0.69240019],
       [0.65472621, 0.66609347, 0.68523222, 0.68523222, 0.68523222],
       [0.76249465, 0.75507289, 0.79954161, 0.76191641, 0.76191641],
       [0.74273788, 0.73468042, 0.7376995 , 0.79099669, 0.79099669],
       [0.73736041, 0.77579076, 0.77035639, 0.72921021, 0.72921021],
       [0.80938868, 0.71895858, 0.7131225 , 0.74615143, 0.77890421],
       [0.73586211, 0.732111  , 0.8062796 , 0.7681506 , 0.75516484]])
```

In [ ]:
```
test_scores
```

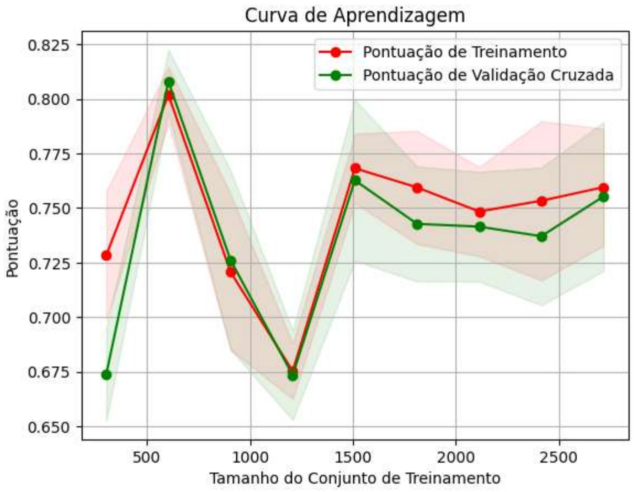
Out[162]:
```
array([[0.67853829, 0.66348071, 0.70235572, 0.68489918, 0.63963452],
       [0.79019133, 0.80075454, 0.8245199 , 0.82610972, 0.79825094],
       [0.77762802, 0.77061726, 0.70156711, 0.70792641, 0.67298554],
       [0.64328076, 0.65951876, 0.69125218, 0.69919499, 0.67378172],
       [0.76050705, 0.74205087, 0.82689202, 0.7682199 , 0.71580224],
       [0.7361327 , 0.71903626, 0.72695384, 0.79360663, 0.73800677],
       [0.75347309, 0.77538042, 0.7388523 , 0.74125596, 0.69837479],
       [0.78864286, 0.71586292, 0.69522043, 0.74759634, 0.73803205],
       [0.74794373, 0.72300452, 0.8030888 , 0.78567643, 0.71661106]])
```

In [ ]:
```
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
```

In [ ]:
```
plt.figure()
plt.title("Curva de Aprendizagem")
plt.xlabel("Tamanho do Conjunto de Treinamento")
plt.ylabel("Pontuação")
plt.grid()

plt.fill_between(train_sizes_abs, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes_abs, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes_abs, train_scores_mean, 'o-', color="r",
         label="Pontuação de Treinamento")
plt.plot(train_sizes_abs, test_scores_mean, 'o-', color="g",
         label="Pontuação de Validação Cruzada")

plt.legend(loc="best")
plt.show()
```



Nota-se que, a partir de 1500 dados de treinamento, o modelo preditivo já apresenta uma estabilidade de pontuação, o que demonstra que possívelmente o aumento do banco de dados não traria uma alteração significativa neste indicador.