Modelo para o Sensor CEI

Este dataset **"DataCEI.csv"** possui informações dispostas em colunas sobre as características dos objetos que passam pelo sensor:

- **Tamanho**: Segue a classificação do CEI2020 (Tamanho='0' - Grande 100%).
- **Referencia**: Referência dinâmica do *Threshold.
- **NumAmostra**: Número de amostras adquiridas.
- **Area**: Somatório das Amplitudes das amostras.
- **Delta**: Máxima Amplitude da amostra.
- **Output1**: Peça tipo 1.
- **Output2**: Peça tipo 2.

Bibliotecas

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

#Função do cáculo da sigmóide
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

Carregando os dados

Vamos começar lendo o arquivo DataCEI.csv em um dataframe do pandas.

In [2]:
```
DataSet=pd.read_csv('arruela .csv')
```

In [3]:
```
DataSet.head()
```

Out[3]:

	Hora	Tamanho	Referencia	NumAmostra	Area	Delta	Output1	Output2
0	13:00:06	53	25	69	81	68	1	0
1	13:00:07	53	26	89	87	56	1	0
2	13:00:08	53	27	68	69	55	1	0
3	13:00:09	53	28	36	50	80	1	0
4	13:00:10	53	29	71	72	50	1	0

In [4]:
```
DataSet.drop(['Hora','Tamanho','Referencia'],axis=1,inplace=True)
```

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

Out[5]:

	NumAmostra	Area	Delta	Output1	Output2
0	69	81	68	1	0
1	89	87	56	1	0
2	68	69	55	1	0
3	36	50	80	1	0
4	71	72	50	1	0

In [6]:
```
DataSet.describe()
```

Out[6]:

	NumAmostra	Area	Delta	Output1	Output2
count	261.000000	261.000000	261.000000	261.000000	261.000000
mean	59.777778	63.697318	54.747126	0.375479	0.624521
std	17.293075	30.629366	35.548413	0.485177	0.485177
min	3.000000	6.000000	17.000000	0.000000	0.000000
25%	50.000000	46.000000	38.000000	0.000000	0.000000
50%	59.000000	56.000000	44.000000	0.000000	1.000000
75%	69.000000	68.000000	54.000000	1.000000	1.000000

NumAmostra	Area	Delta	Output1	Output2

Váriaveis do *Dataset*

In [7]:
```
DataSet.columns
```

Out[7]:
```
Index(['NumAmostra', 'Area', 'Delta', 'Output1', 'Output2'], dtype='object')
```

Número de Peças

Vamos classificar os grupos pelo número de peças:

1. Grupo com uma peça
2. Grupo com duas peças

In [8]:
```
sns.set_style('whitegrid')
sns.countplot(x='Output2',data=DataSet,palette='RdBu_r')
plt.show()
```

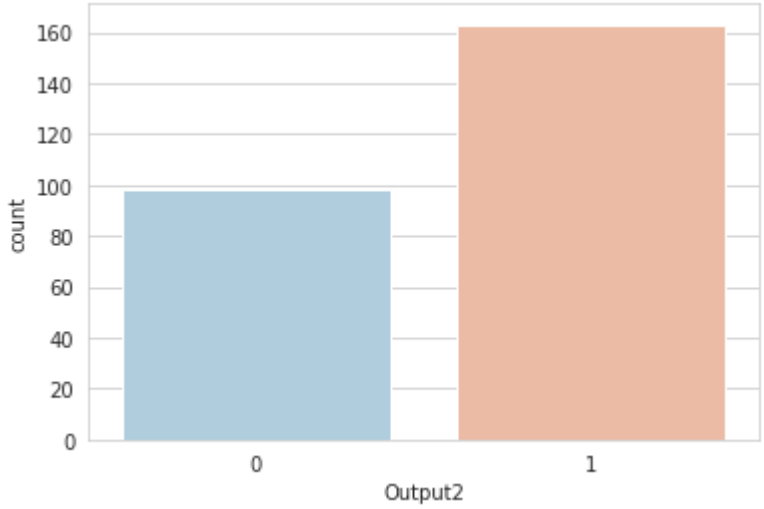
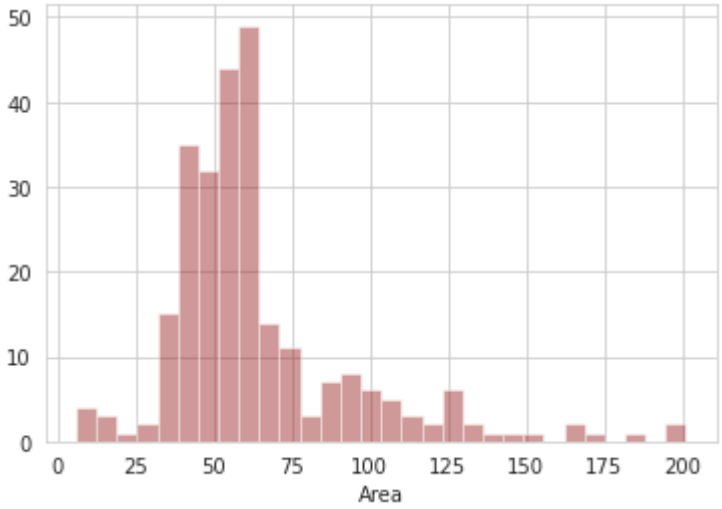


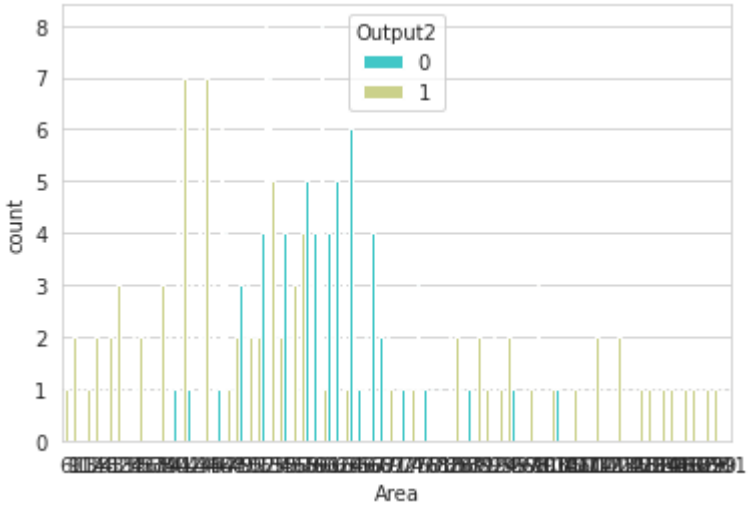
Gráfico da distribuição das áreas das peças

In [9]:
```
sns.distplot(DataSet['Area'].dropna(),kde=False,color='darkred',bins=30)
plt.show()
```

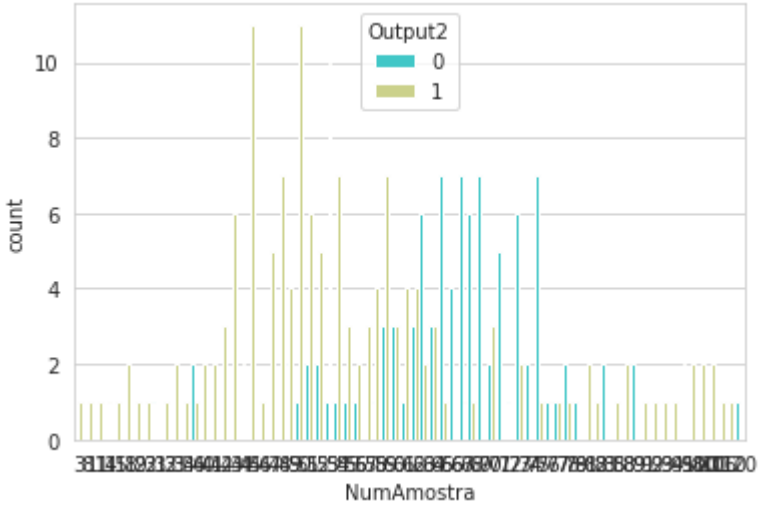
```
/home/darlan/.local/lib/python3.8/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



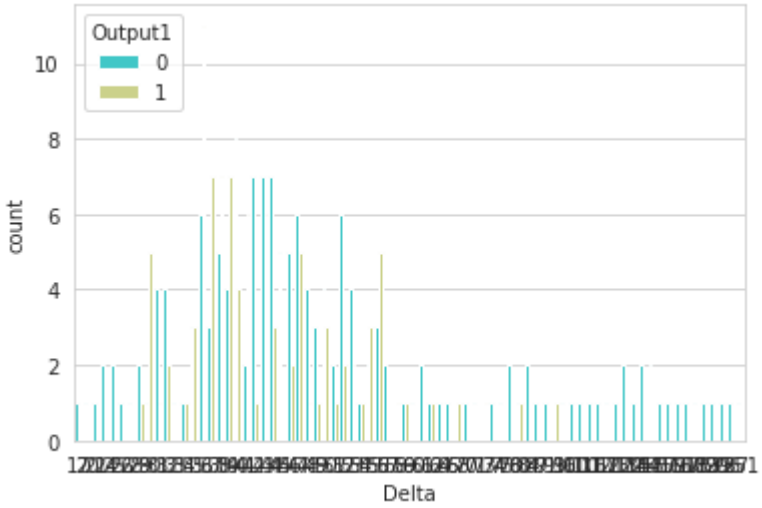
In [10]:
```
sns.set_style('whitegrid')
sns.countplot(x='Area',hue='Output2',data=DataSet,palette='rainbow')
plt.show()
```



In [11]:
```
sns.set_style('whitegrid')
sns.countplot(x='NumAmostra',hue='Output2',data=DataSet,palette='rainbow')
plt.show()
```



In [12]:
```
sns.set_style('whitegrid')
sns.countplot(x='Delta',hue='Output1',data=DataSet,palette='rainbow')
plt.show()
```



As variáveis preditoras e a variável de resposta

Para treinar o modelo de regressão, primeiro precisaremos dividir nossos dados em uma matriz **X** que contenha os dados das variáveis preditoras e uma matriz **y** com os dados da variável de destino.

Matrizes X e y

In [13]:
```
#X = DataSet[[ 'NumAmostra', 'Area', 'Delta']]
#y = DataSet[['Output1','Output2']]
```

Relação entre as variáveis preditoras

Algumas questões importantes

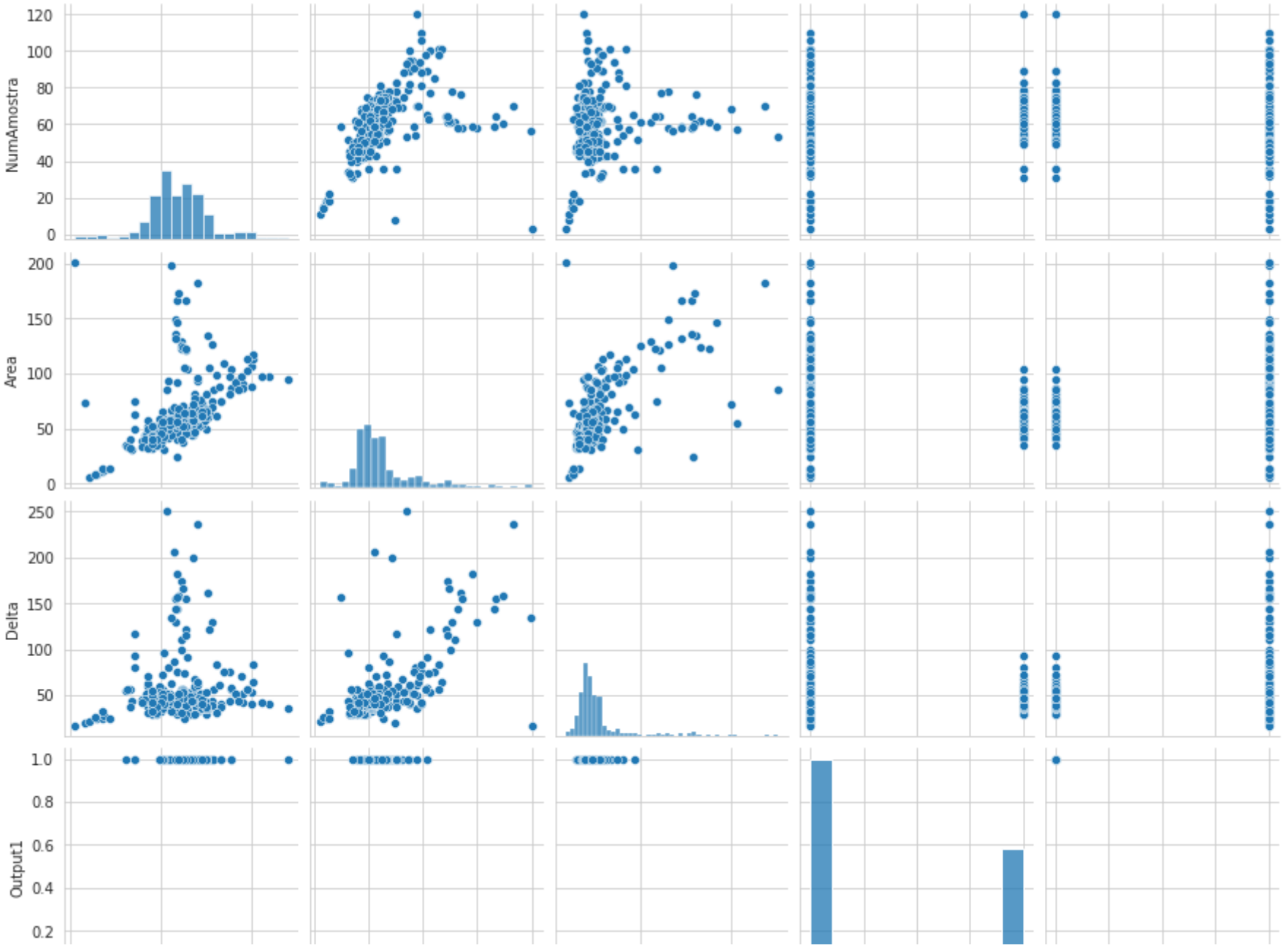
1. Pelo menos um dos preditores ***x1, x2, ... ,x5*** é útil na previsão da resposta?
2. Todos os preditores ajudam a explicar **y**, ou apenas um subconjunto dos preditores?

3. Quão bem o modelo se ajusta aos dados?
4. Dado um conjunto de valores de previsão, quais valores de resposta devemos prever e quais as métricas indicam um bom modelo de previsão?

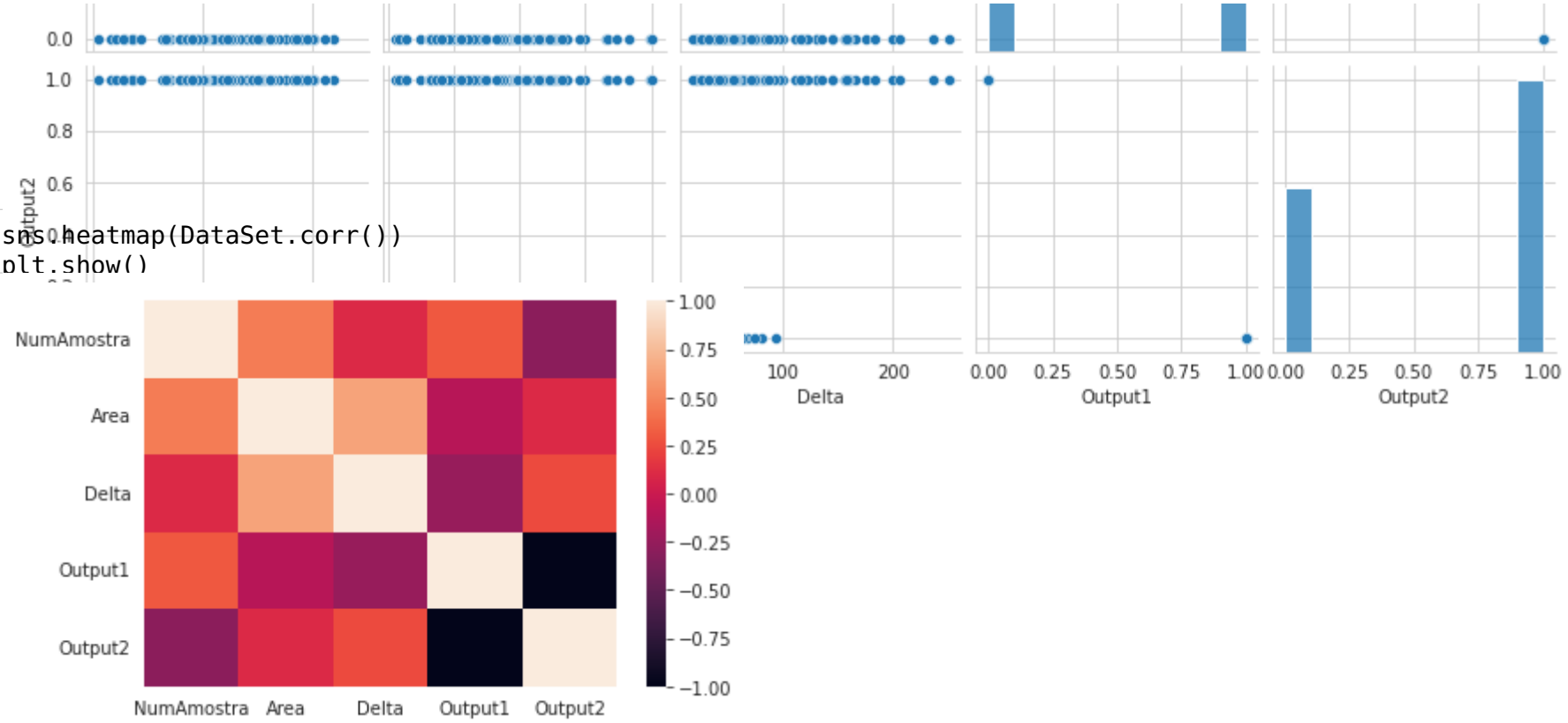
Gráficos simples de dispersão

Pelos gráficos abaixo percebemos ... nossa variável de resposta

In [14]:
```
sns.pairplot(DataSet)
plt.show()
```



In [15]:
```
sns.heatmap(DataSet.corr())
plt.show()
```



Normalização dos Dados

In [16]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
DataScaled=scaler.fit_transform(DataSet)
DataSetScaled=pd.DataFrame(np.array(DataScaled),columns = ['NumAmostra', 'Area', 'Delta', 'Output1','Output2
```

In [17]:
```
DataSetScaled.head()
```

Out[17]:

	NumAmostra	Area	Delta	Output1	Output2
0	0.534314	0.565990	0.373528	1.289676	-1.289676

	NumAmostra	Area	Delta	Output1	Output2
1	1.693069	0.762257	0.035312	1.289676	-1.289676
2	0.476377	0.173457	0.007127	1.289676	-1.289676
3	-1.377630	-0.448055	0.711745	1.289676	-1.289676

Conjunto de dados para o treinamento

In [18]:
```
X = DataSetScaled.drop(['Output1', 'Output2'],axis=1)
y = DataSet[['Output1','Output2']]
```

Separando os dados de treinamento e de validação

Agora vamos dividir os dados em um conjunto de treinamento e um conjunto de testes. Vamos treinar o modelo no conjunto de treinamento, em seguida, usar o conjunto de teste para validar o modelo.

Em nosso exemplo iremos separar de forma randômica 33% dos dados para validação. Estes dados não serão utilizados para determinação dos coeficientes preditores do modelo.

In [19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.29, random_state=3)

print(y_test)
print(X_test)
```

```
     Output1  Output2
257        0        1
38         1        0
232        0        1
89         1        0
43         1        0
..       ...      ...
115        0        1
10         1        0
209        0        1
12         1        0
70         1        0

[76 rows x 2 columns]
     NumAmostra      Area     Delta
257   -2.188758 -1.625656 -0.838414
```

Criando o Modelo de MPL

In [20]:
```
#Tamanho do DataSet de Treinamento
n_records, n_features = X_train.shape

#Arquitetura da MPL
N_input = 3
N_hidden = 8
N_output = 2
learnrate = 0.1
```

Inicialização dos pesos da MPL (Aleatório)

In [21]:
```
#Pesos da Camada Oculta (Inicialização Aleatória)
weights_input_hidden = np.random.normal(0, scale=0.1, size=(N_input, N_hidden))
print('Pesos da Camada Oculta:')
print(weights_input_hidden)

#Pesos da Camada de Saída (Inicialização Aleatória)
weights_hidden_output = np.random.normal(0, scale=0.1, size=(N_hidden, N_output))
print('Pesos da Camada de Saída:')
```

```
print(weights_hidden_output)
```

```
Pesos da Camada Oculta:
[[ 4.72915326e-02  4.60740100e-03 -1.15281410e-01 -3.22704791e-02
   2.69951847e-02  1.15730567e-02  1.76814705e-02 -1.21859984e-01]
 [-8.73930302e-02  6.15402701e-03  2.18680283e-01 -1.57085541e-01
  -2.45196192e-02  1.28065576e-01 -1.09162209e-01 -2.49876159e-02]
 [-2.05867862e-04  1.83997657e-03 -1.32674652e-03 -6.91854174e-03
  -3.51285328e-02 -9.38302690e-02  2.65842088e-02  2.95311052e-01]]
Pesos da Camada de Saída:
[[-0.05452357  0.14794183]
 [-0.09949464 -0.08270595]
 [-0.08574935  0.07253314]
 [-0.0453392   0.07652729]
 [-0.17747435  0.04745341]
 [-0.06101152  0.02034725]
 [ 0.0360105  -0.0575013 ]
 [-0.089793   -0.08417199]]
```

Algoritmo Backpropagation

In [22]:

```
epochs = 50000
last_loss=None
EvolucaoError=[]
IndiceError=[]

for e in range(epochs):
    delta_w_i_h = np.zeros(weights_input_hidden.shape)
    delta_w_h_o = np.zeros(weights_hidden_output.shape)
    for xi, yi in zip(X_train.values, y_train.values):

# Forward Pass
        #Camada oculta
        #Calcule a combinação linear de entradas e pesos sinápticos
        hidden_layer_input = np.dot(xi, weights_input_hidden)
        #Aplicado a função de ativação
        hidden_layer_output = sigmoid(hidden_layer_input)

        #Camada de Saída
        #Calcule a combinação linear de entradas e pesos sinápticos
```

```
        output_layer_in = np.dot(hidden_layer_output, weights_hidden_output)

        #Aplicado a função de ativação
        output = sigmoid(output_layer_in)
        #print('As saídas da rede são',output)
#-----------------------------------------

# Backward Pass
        ## TODO: Cálculo do Erro
        error = yi - output

        # TODO: Calcule o termo de erro de saída (Gradiente da Camada de Saída)
        output_error_term = error * output * (1 - output)

        # TODO: Calcule a contribuição da camada oculta para o erro
        hidden_error = np.dot(weights_hidden_output,output_error_term)

        # TODO: Calcule o termo de erro da camada oculta (Gradiente da Camada Oculta)
        hidden_error_term = hidden_error * hidden_layer_output * (1 - hidden_layer_output)

        # TODO: Calcule a variação do peso da camada de saída
        delta_w_h_o += output_error_term*hidden_layer_output[:, None]

        # TODO: Calcule a variação do peso da camada oculta
        delta_w_i_h += hidden_error_term * xi[:, None]

    #Atualização dos pesos na época em questão
    weights_input_hidden += learnrate * delta_w_i_h / n_records
    weights_hidden_output += learnrate * delta_w_h_o / n_records


    # Imprimir o erro quadrático médio no conjunto de treinamento

    if  e % (epochs / 20) == 0:
        hidden_output = sigmoid(np.dot(xi, weights_input_hidden))
        out = sigmoid(np.dot(hidden_output,
                             weights_hidden_output))
        loss = np.mean((out - yi) ** 2)

        if last_loss and last_loss < loss:
            print("Erro quadrático no treinamento: ", loss, " Atenção: O erro está aumentando")
```

```
        else:
            print("Erro quadrático no treinamento: ", loss)
        last_loss = loss

        EvolucaoError.append(loss)
        IndiceError.append(e)
```

```
Erro quadrático no treinamento:  0.20728102466606496
Erro quadrático no treinamento:  0.2878032325194561  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.3081567986896804  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.1791350706720894
Erro quadrático no treinamento:  0.1191150623732222
Erro quadrático no treinamento:  0.09409715720043799
Erro quadrático no treinamento:  0.08137943194221837
Erro quadrático no treinamento:  0.07384430443272436
Erro quadrático no treinamento:  0.06893009486503718
Erro quadrático no treinamento:  0.06562167217311082
Erro quadrático no treinamento:  0.06354266384174756
Erro quadrático no treinamento:  0.06262654370440127
Erro quadrático no treinamento:  0.06290284487990228  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.0642636267086854  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.06658696544938562  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.0699622458919548  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.07437625873598859  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.07951251924942154  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.08498981229506311  Atenção: O erro está aumentando
Erro quadrático no treinamento:  0.09032456776104966  Atenção: O erro está aumentando
```

In [23]:
```
### Gráfico da Evolução do Erro
```

In [24]:
```
plt.plot(IndiceError, EvolucaoError, 'r') # 'r' is the color red
plt.xlabel('')
plt.ylabel('Erro Quadrático')
plt.title('Evolução do Erro no treinamento da MPL')
plt.show()
```

Validação do modelo

In [25]:
```
# Calcule a precisão dos dados de teste
n_records, n_features = X_test.shape
predictions=0

for xi, yi in zip(X_test.values, y_test.values):

# Forward Pass
        #Camada oculta
        #Calcule a combinação linear de entradas e pesos sinápticos
        hidden_layer_input = np.dot(xi, weights_input_hidden)
        #Aplicado a função de ativação
        hidden_layer_output = sigmoid(hidden_layer_input)

        #Camada de Saída
```

```
        #Calcule a combinação linear de entradas e pesos sinápticos
        output_layer_in = np.dot(hidden_layer_output, weights_hidden_output)

        #Aplicado a função de ativação
        output = sigmoid(output_layer_in)

#-----------------------------------------

#Cálculo do Erro da Predição
        ## TODO: Cálculo do Erro
        if (output[0]>output[1]):
            if (yi[0]>yi[1]):
                predictions+=1

        if (output[1]>=output[0]):
            if (yi[1]>yi[0]):
                predictions+=1

print("A Acurácia da Predição é de: {:.3f}".format(predictions/n_records))
```

```
A Acurácia da Predição é de: 0.921
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: