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 [126]:
```
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 [127]:
```
DataSet=pd.read_csv('arruela .csv')
```

In [128]:
```
DataSet.head()
```

Out[128]:

	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 [129]:
```
DataSet.drop(['Hora','Tamanho','Referencia'],axis=1,inplace=True)
```

In [130]:
```
DataSet.head()
```

Out[130]:

	NumAmostra	Area	Delta	Output1	Output2
0	69	81	68	1	0

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

In [131]:
```
DataSet.describe()
```

Out[131]:

	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
max	120.000000	201.000000	251.000000	1.000000	1.000000

Váriaveis do *Dataset*

In [132]:
```
DataSet.columns
```

Out[132]:
```
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 [133]:
```
sns.set_style('whitegrid')
sns.countplot(x='Output2',data=DataSet,palette='RdBu_r')
plt.show()
```

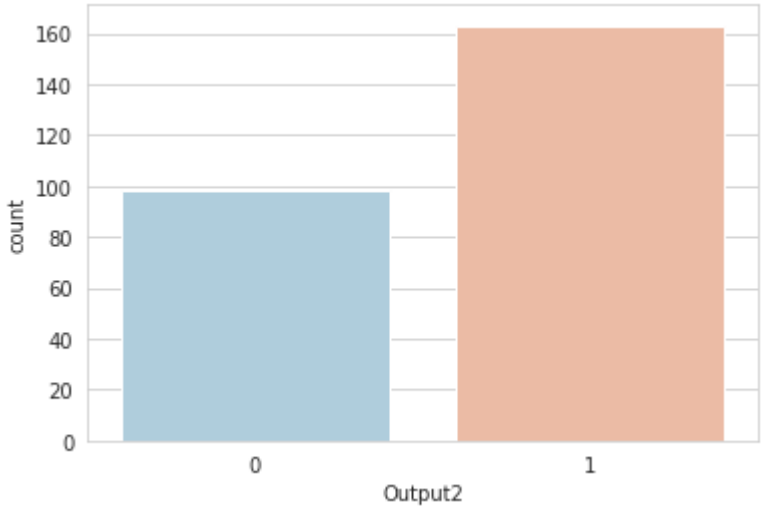
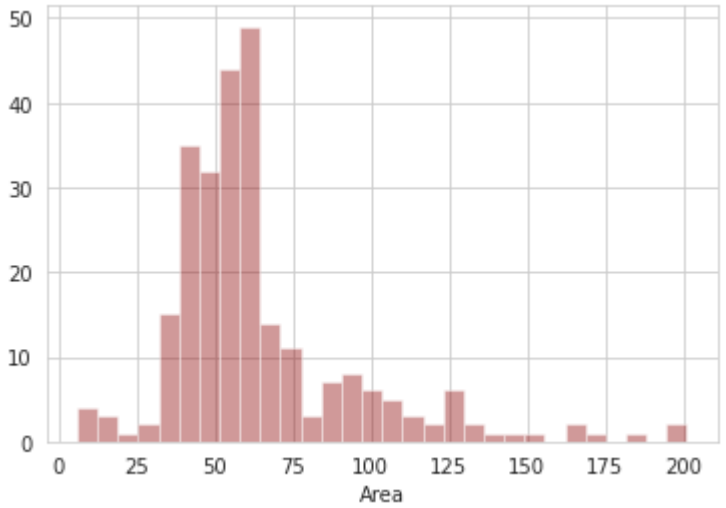


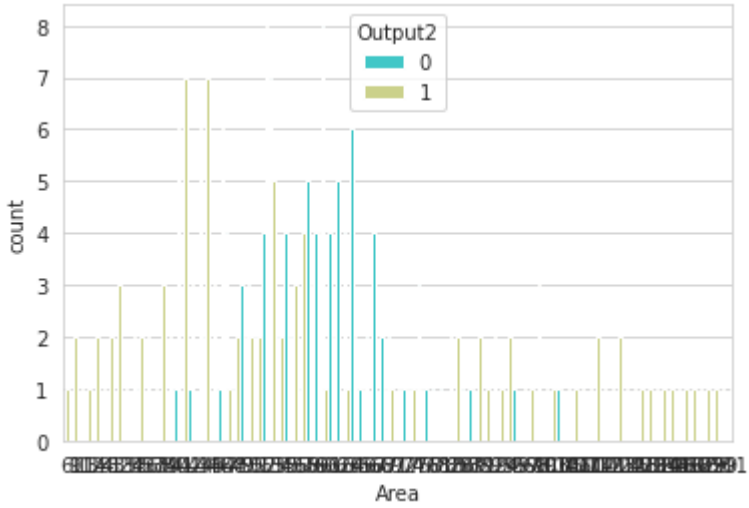
Gráfico da distribuição das áreas das peças

In [134]:
```
sns.distplot(DataSet['Area'].dropna(),kde=False,color='darkred',bins=
plt.show()
```

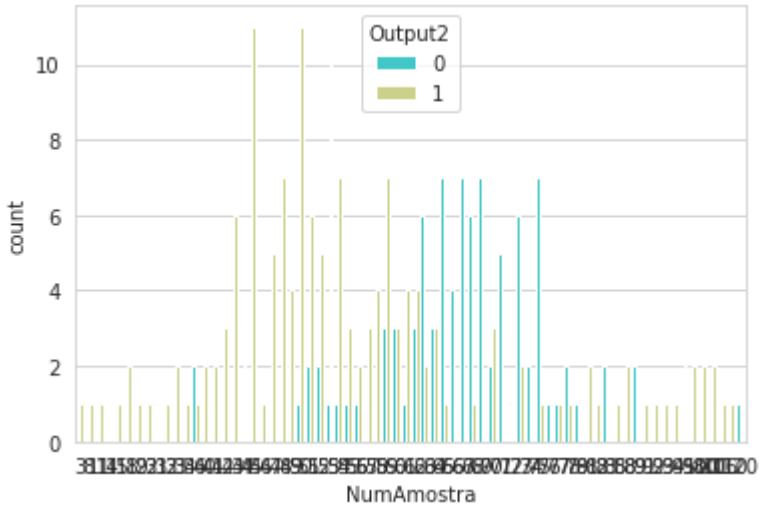
```
/home/darlan/.local/lib/python3.8/site-packages/seaborn/distributio
ns.py:2619: FutureWarning: `distplot` is a deprecated function and
will be removed in a future version. Please adapt your code to use
either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level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



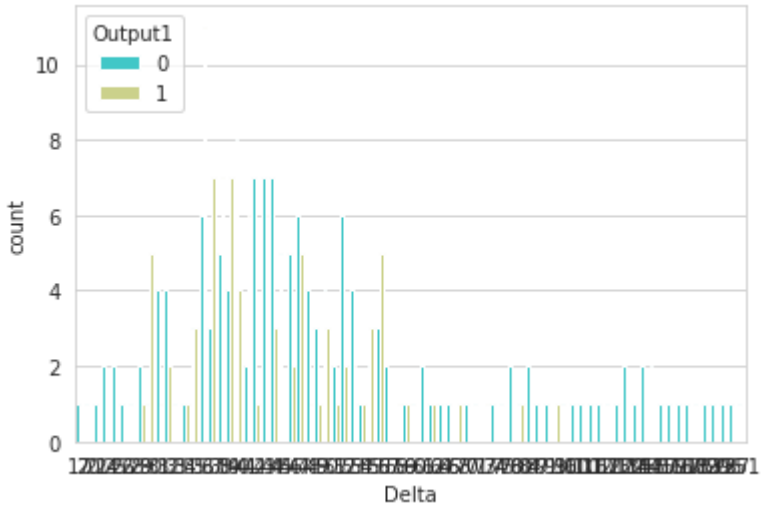
In [135]:
```
sns.set_style('whitegrid')
sns.countplot(x='Area',hue='Output2',data=DataSet,palette='rainbow')
plt.show()
```



In [136]:
```
sns.set_style('whitegrid')
sns.countplot(x='NumAmostra',hue='Output2',data=DataSet,palette='rain
plt.show()
```



In [137]:
```
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 [138]:
```
#X = DataSet[[ 'NumAmostra', 'Area', 'Delta']]
#y = DataSet[['Output1','Output2']]
```

Relação entre as variáveis preditoras

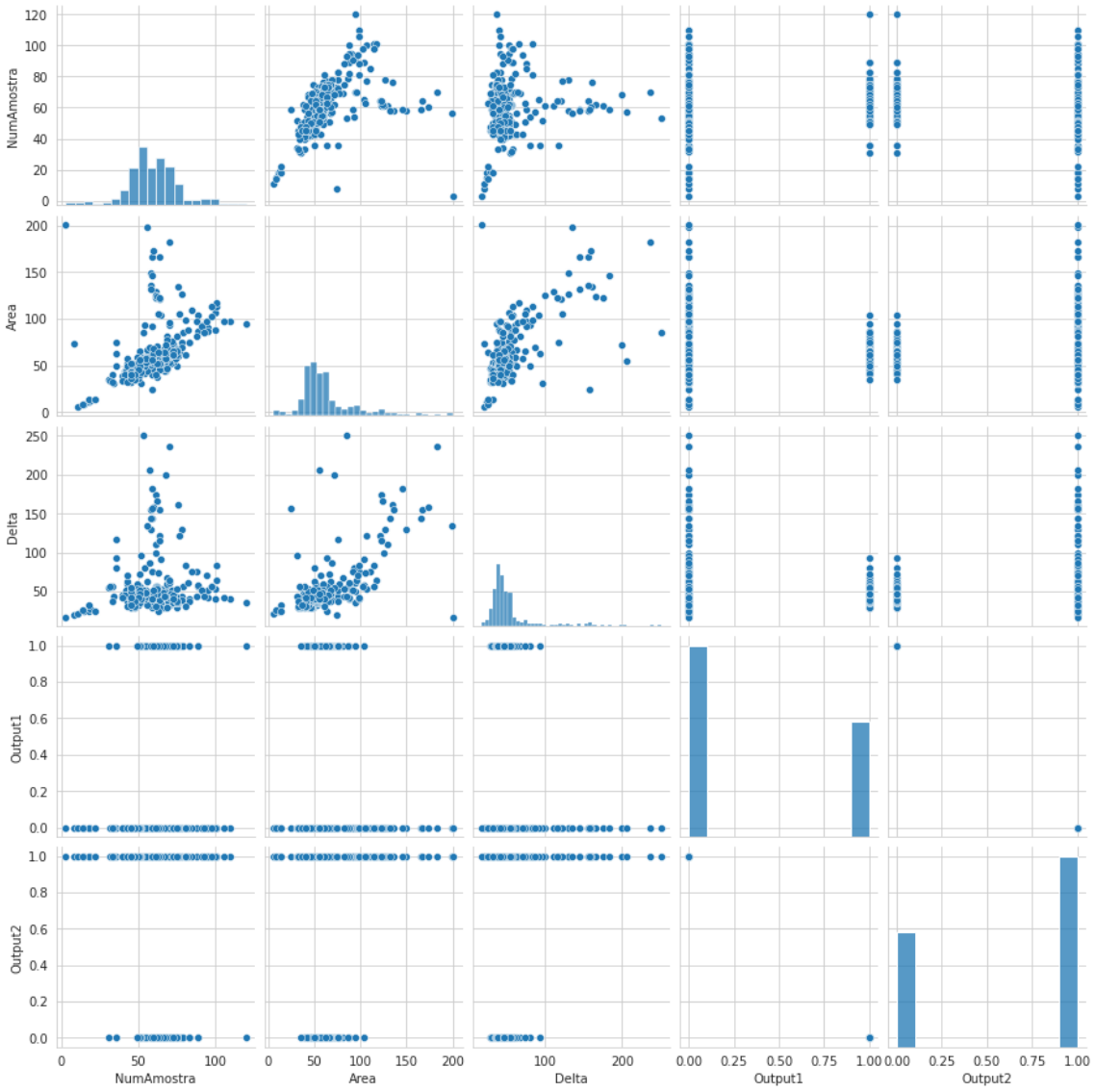
Algumas questões importantes

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

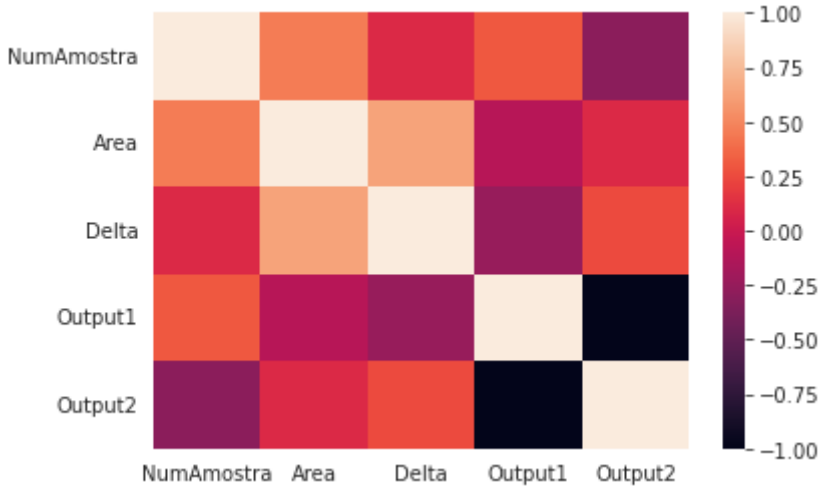
In [139]:
```
sns.pairplot(DataSet)
plt.show()
```



Mapa de Calor

O gráfico abaixo mostra através de uma escala de cores a correlação entre as variáveis do *Dataset*. Se observarmos as cores deste gráfico, a variável preditora **'Area'** possui maior correlação com a variável de resposta **'Output'** e a variável **'NumAmostra'** a menor.

In [140]:
```
sns.heatmap(DataSet.corr())
plt.show()
```



Normalização dos Dados

In [141]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
DataScaled=scaler.fit_transform(DataSet)
DataSetScaled=pd.DataFrame(np.array(DataScaled),columns = ['NumAmostr
```

In [142]:
```
DataSetScaled.head()
```

Out[142]:

	NumAmostra	Area	Delta	Output1	Output2
0	0.534314	0.565990	0.373528	1.289676	-1.289676
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
4	0.650190	0.271590	-0.133796	1.289676	-1.289676

Conjunto de dados para o treinamento

In [143]:
```
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 [144]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0

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
38    -0.045063 -0.775166 -0.697490
232   -0.566502 -0.906011 -0.641121
89     0.476377 -0.186366 -0.331089
43     1.055754  0.369724 -0.331089
..          ...       ...       ...
115    1.866882  0.991235 -0.133796
10     0.881941 -0.480766 -0.697490
209   -2.420509 -1.723789 -0.866598
12     0.418439 -0.120943 -0.302904
70     0.708128  0.402435 -0.077427

[76 rows x 3 columns]
```

Criando o Modelo de MPL

In [145]:
```
#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 [146]:
```
#Pesos da Camada Oculta (Inicialização Aleatória)
weights_input_hidden = np.random.normal(0, scale=0.1, size=(N_input,
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_hidder
print('Pesos da Camada de Saída:')
print(weights_hidden_output)
```

```
Pesos da Camada Oculta:
[[-0.05024701  0.03579085 -0.04499742  0.02160817  0.01755982 -0.01
62043
  -0.01833555 -0.05623326]
 [-0.05907907 -0.07755983 -0.09863729 -0.06375237  0.07772127 -0.01
590317
  -0.16412521 -0.0657305 ]
 [-0.00909589  0.1445905   0.10704606 -0.09075483 -0.07121848  0.03
41477
  -0.13634234 -0.09563915]]
Pesos da Camada de Saída:
[[-0.18692787 -0.03906768]
 [-0.061203    0.09576956]
 [-0.00628076 -0.05688909]
 [-0.16617777  0.02665909]
 [-0.2104961   0.0427086 ]
 [-0.02666952  0.03875231]
 [-0.04316108 -0.00822638]
 [ 0.07342516  0.15934462]]
```

Algoritmo Backpropagation

In [147]:
```
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_

        #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
        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 d
        hidden_error_term = hidden_error * hidden_layer_output * (1 -

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

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

```
Erro quadrático no treinamento:  0.19846882624524795
Erro quadrático no treinamento:  0.25149893493836717  Atenção: O er
ro está aumentando
Erro quadrático no treinamento:  0.37474412327389306  Atenção: O er
ro está aumentando
Erro quadrático no treinamento:  0.22801640414496277
Erro quadrático no treinamento:  0.14878436494636074
Erro quadrático no treinamento:  0.11011898692240663
Erro quadrático no treinamento:  0.09110057359133714
Erro quadrático no treinamento:  0.08042440546866703
Erro quadrático no treinamento:  0.07378277900582117
Erro quadrático no treinamento:  0.06944541746716763
Erro quadrático no treinamento:  0.06671503524838217
Erro quadrático no treinamento:  0.06531912107725217
Erro quadrático no treinamento:  0.06510906515945197
Erro quadrático no treinamento:  0.06584016217615082  Atenção: O er
ro está aumentando
Erro quadrático no treinamento:  0.06721748888597032  Atenção: O er
ro está aumentando
Erro quadrático no treinamento:  0.0693036308882595  Atenção: O err
```

```
o está aumentando
Erro quadrático no treinamento:  0.07225423771069311  Atenção: O er
ro está aumentando
Erro quadrático no treinamento:  0.07580780404768325  Atenção: O er
ro está aumentando
Erro quadrático no treinamento:  0.07965293176651056  Atenção: O er
ro está aumentando
Erro quadrático no treinamento:  0.08390578260608561  Atenção: O er
ro está aumentando
```

In [148]:
```
### Gráfico da Evolucão do Erro
```

In [149]:
```
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 [150]:
```
# 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_

        #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_rec
```

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

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: