TrabalhoRNA

July 10, 2021

0.1 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.

0.1.1 Bibliotecas

[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))
```

0.1.2 Carregando os dados

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

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

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

[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

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

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

[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

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

[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
max	120.000000	201.000000	251.000000	1.000000	1.000000

0.1.3 Váriaveis do *Dataset*

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

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

0.1.4 Número de Peças

Vamos classificar os grupos pelo número de peças:

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

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

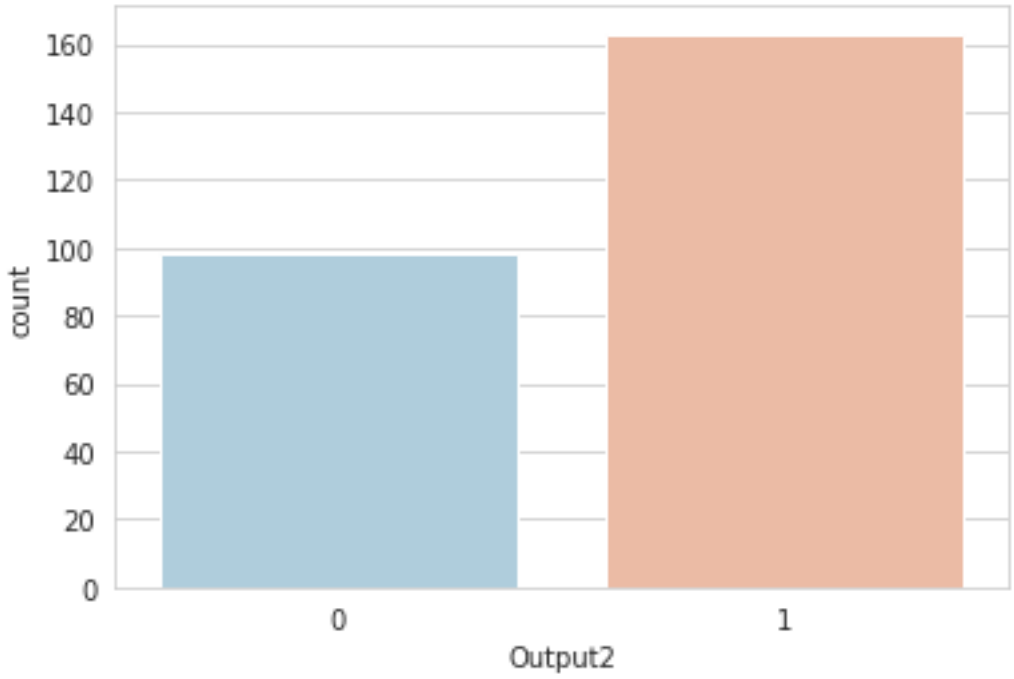
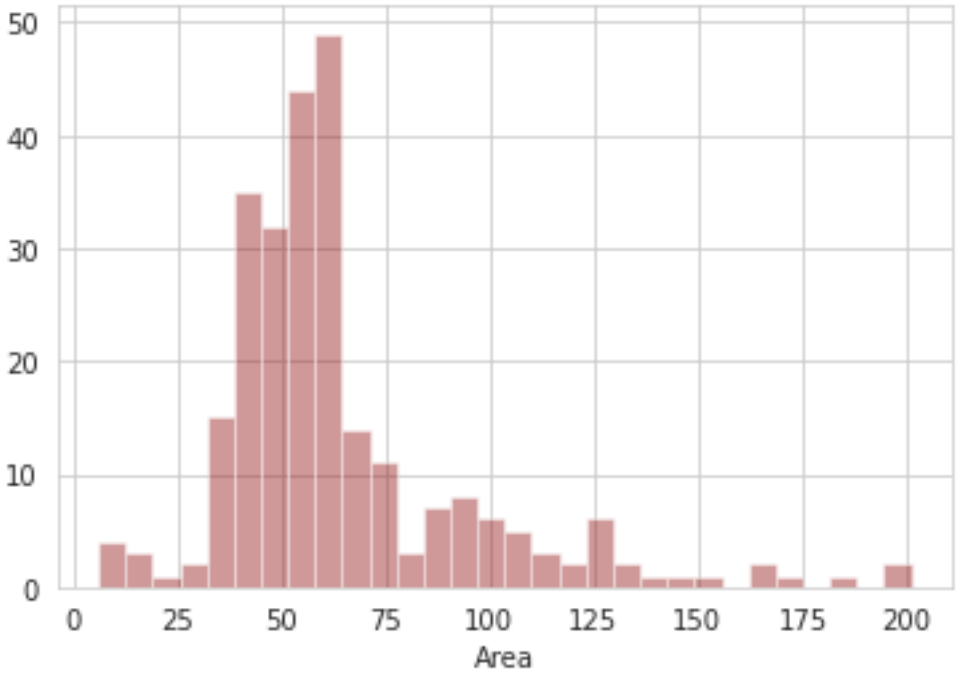


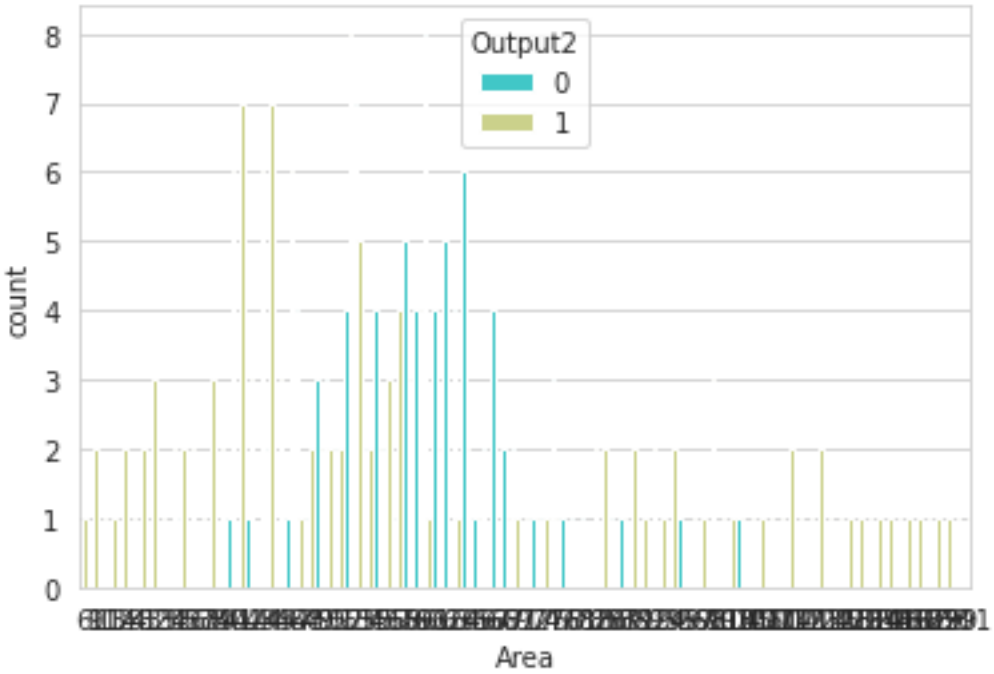
Gráfico da distribuição das áreas das peças

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

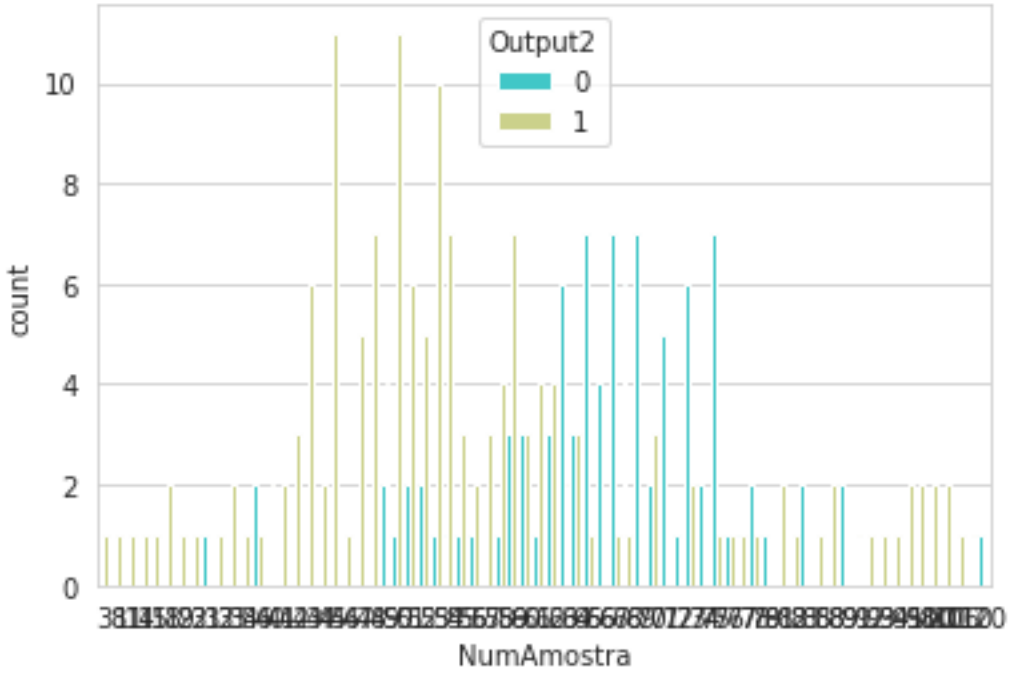
```
/home/vinicius-reis/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



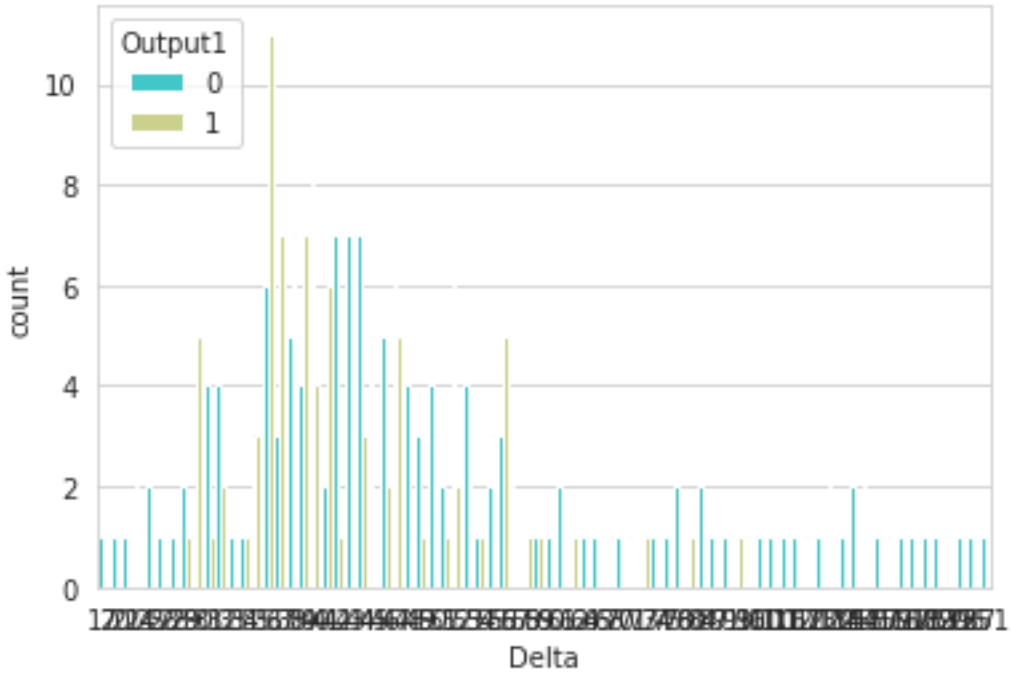
[10]:
```
sns.set_style('whitegrid')
sns.countplot(x='Area',hue='Output2',data=DataSet,palette='rainbow')
plt.show()
```



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



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



0.2 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.

0.2.1 Matrizes X e y

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

0.2.2 Relação entre as variáveis preditoras

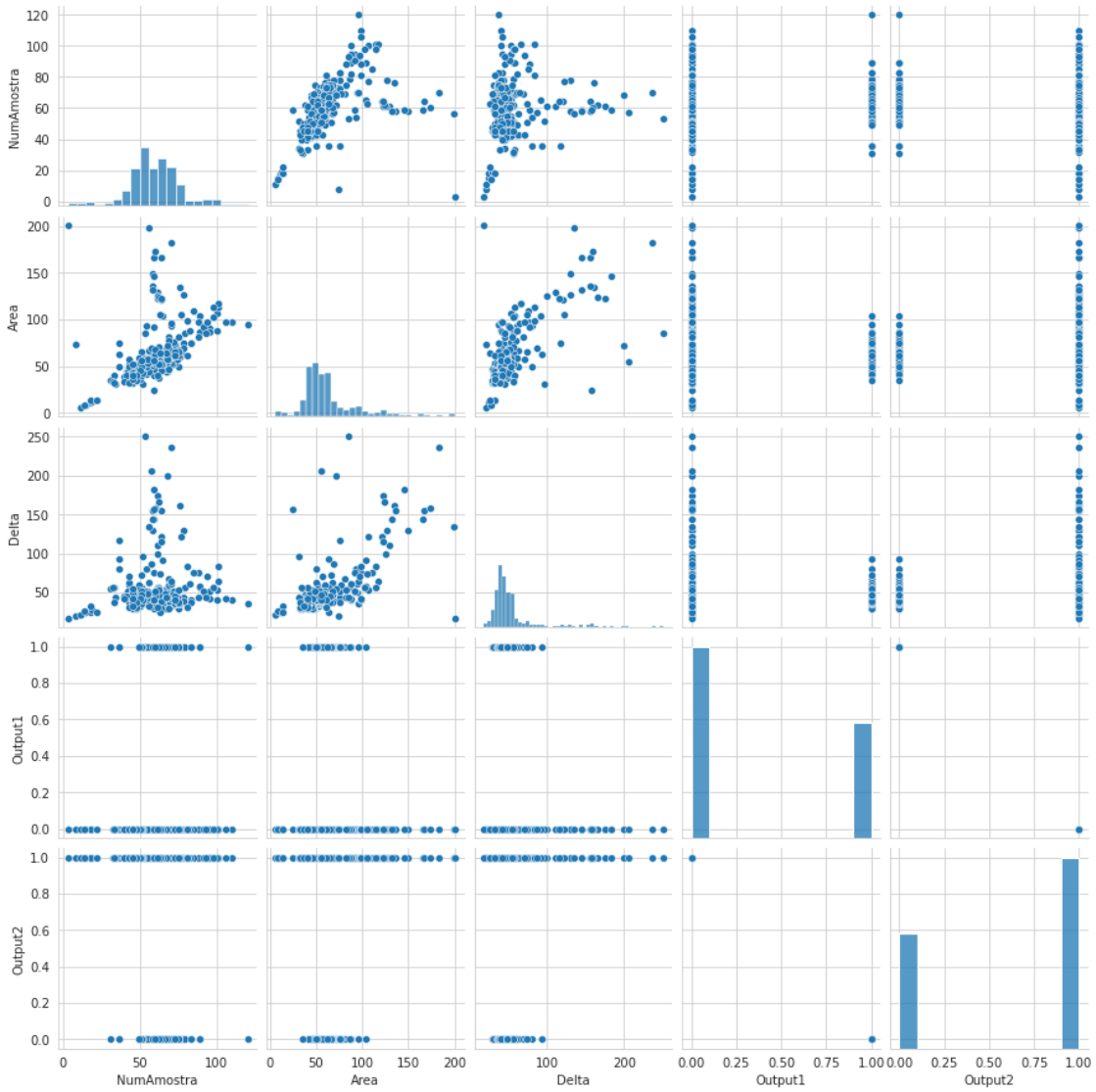
Algumas questões importantes

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

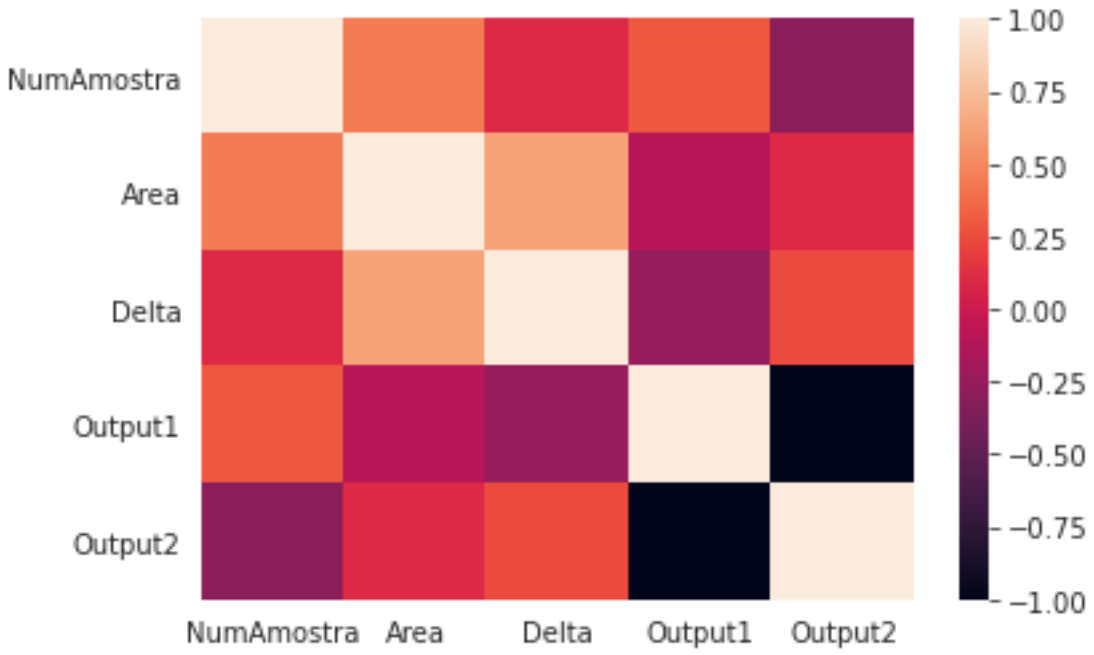
[14]:
```
sns.pairplot(DataSet)
plt.show()
```



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.

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



0.3 Normalização dos Dados

[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'])
```

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

[17]:

	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

0.3.1 Conjunto de dados para o treinamento

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

0.4 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.

[19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
 ↪random_state=101)

print(y_test)
print(X_test)
```

```
     Output1  Output2
89         1        0
212        0        1
218        0        1
96         1        0
88         1        0
..       …      …
198        0        1
167        0        1
235        0        1
51         1        0
18         1        0

[105 rows x 2 columns]
     NumAmostra      Area     Delta
89     0.476377 -0.186366 -0.331089
212   -0.856191 -1.036855 -0.725675
218    1.229567 -0.088232 -0.669306
96    -1.667319 -0.938722  0.007127
88    -0.103000 -0.415344 -0.472013
..          …       …       …
198   -0.045063 -1.298544  2.881966
167   -0.566502 -0.611610 -0.528382
235   -2.826073 -1.887345 -0.951153
51     0.128750 -0.480766 -0.528382
18     0.476377 -0.055521 -0.387459

[105 rows x 3 columns]
```

0.5 Criando o Modelo de MPL

[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 = 0.5
```

0.6 Inicialização dos pesos da MPL (Aleatório)

[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:
[[ 0.01024096  0.05237963  0.07041105  0.06883843  0.11229393 -0.0807053
   0.07551717 -0.21357359]
 [-0.12198196 -0.09356254  0.00844428  0.13894575  0.02121995 -0.04434153
  -0.11830622 -0.0734081 ]
 [-0.01430984  0.13576897 -0.10154366  0.024471    0.04944712 -0.00301077
   0.06304921  0.13948487]]
Pesos da Camada de Saída:
[[-0.05391214 -0.15763745]
 [ 0.07625524 -0.00695071]
 [ 0.01084427  0.07103054]
 [ 0.07948898 -0.00642063]
 [ 0.04187263  0.05895919]
 [ 0.01147494  0.13127165]
 [ 0.04395616 -0.01019492]
 [-0.00940967 -0.03032546]]
```

0.7 Algoritmo Backpropagation

[22]:
```
epochs = 35000
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 / 35) == 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:  0.24981113441659553
Erro quadrático no treinamento:  0.09737297964316961
Erro quadrático no treinamento:  0.04089195237689541
Erro quadrático no treinamento:  0.021442293139055834
Erro quadrático no treinamento:  0.01106802522119246
Erro quadrático no treinamento:  0.006385489316836882
Erro quadrático no treinamento:  0.003939538068961567
Erro quadrático no treinamento:  0.00253145430322401
Erro quadrático no treinamento:  0.0016908805467785444
Erro quadrático no treinamento:  0.0011703556540110164
Erro quadrático no treinamento:  0.0008351631584366497
Erro quadrático no treinamento:  0.0006137319897524628
Erro quadrático no treinamento:  0.0004691665557737976 7
Erro quadrático no treinamento:  0.000366001130610768
Erro quadrático no treinamento:  0.00028361675038329325
Erro quadrático no treinamento:  0.0002204381975137328
Erro quadrático no treinamento:  0.0001729794353947137
Erro quadrático no treinamento:  0.00013735249364434885
Erro quadrático no treinamento:  0.00011038061107648727
Erro quadrático no treinamento:  8.931389393396563e-05
Erro quadrático no treinamento:  7.267415689765652e-05
Erro quadrático no treinamento:  5.932905661632174e-05
Erro quadrático no treinamento:  4.818093601563984e-05
Erro quadrático no treinamento:  3.879569307668083e-05
Erro quadrático no treinamento:  3.1070247322497774e-05
Erro quadrático no treinamento:  2.4868571439517863e-05
Erro quadrático no treinamento:  1.996415839982547e-05
Erro quadrático no treinamento:  1.610520132392532e-05
```

```
Erro quadrático no treinamento:  1.3065498455875235e-05
Erro quadrático no treinamento:  1.066120181833332e-05
Erro quadrático no treinamento:  8.749214502474387e-06
Erro quadrático no treinamento:  7.219895338727334e-06
Erro quadrático no treinamento:  5.989496663959579e-06
Erro quadrático no treinamento:  4.9939005374853205e-06
Erro quadrático no treinamento:  4.183797246461777e-06
```

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

[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()
```

0.8 Validação do modelo

[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: 0.867
```

[ ]:

[ ]: