Pre Processamento

In [2]:

```
import pandas as pd
```

In [3]:

```
from io import StringIO
```

Criar um arquivo CSV

In [44]:

```
# Criar um arquivo CSV
csv_data = \
'''A, B, C, D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''
```

In [45]:

```
csv_data
```

Out[45]:

```
'A, B, C, D\n1.0,2.0,3.0,4.0\n5.0,6.0,,8.0\n10.0,11.0,12.0,'
```

Ler uma string para transformar num arquivo CSV

In [47]:

```
# ler uma string para transformar num arquivo CSV
df = pd.read_csv(StringIO(csv_data))
# criar um backup (dfb)
dfb = df
```

In [48]:

```
df
```

Out[48]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

In [49]:

```
df.describe()
```

Out[49]:

	A	B	C	D
count	3.000000	3.000000	2.000000	2.000000
mean	5.333333	6.333333	7.500000	6.000000
std	4.509250	4.509250	6.363961	2.828427
min	1.000000	2.000000	3.000000	4.000000
25%	3.000000	4.000000	5.250000	5.000000
50%	5.000000	6.000000	7.500000	6.000000
75%	7.500000	8.500000	9.750000	7.000000
max	10.000000	11.000000	12.000000	8.000000

Ver os dados nulos (Sim ou Não)

In [50]:

```
# ver os dados nulos (Sim ou Não)
df.isnull()
```

Out[50]:

	A	B	C	D
0	False	False	False	False
1	False	False	True	False
2	False	False	False	True

Contar dados nulos em cada coluna

In [51]:

```
# Contar dados nulos em cada coluna
df.isnull().sum()
```

Out[51]:

```
A    0
 B    0
 C    1
 D    1
dtype: int64
```

In [52]:

```
# Outra forma de contar os nulos
count_nan = len(df) - df.count()
count_nan
```

Out[52]:

```
A     0
 B    0
 C    1
 D    1
dtype: int64
```

Ver se tem nulos apresentando True ou False

In [53]:

```
# Ver se tem nulos apresentando True ou False
df.isnull().any()
```

Out[53]:

```
A     False
 B    False
 C     True
 D     True
dtype: bool
```

In [54]:

```
# Ver se tem nulos apresentando True ou False
df1 = df.isnull().sum() > 0
df1
```

Out[54]:

```
A     False
 B    False
 C     True
 D     True
dtype: bool
```

Tornar o DataFrame num array para usar em modelos Deep Learning

In [55]:

```
# Tornar o DataFrame num array para usar em modelos Deep Learning
# Deep Learning não trabalha com DataFrame, temos de transformar num array de phyton
# O Array de numpy tem operações e rapidez que não são executaveis no array de phyton
df1 = df.values
df1
```

Out[55]:

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6., nan,  8.],
       [10., 11., 12., nan]])
```

In [56]:

```
# Como remover as linhas que faltam valores
# 1 - coluna, 0 é linha
df1 = df.dropna(axis=0)
df1
```

Out[56]:

	A	B	C	D
0	1.0	2.0	3.0	4.0

Como remover as linhas que faltam valores

In [57]:

```
# Como remover as linhas que faltam valores
# 1 - coluna, 0 é linha
# inplace - torna efetiva a alteração do drop
df = pd.read_csv(StringIO(csv_data))
df.dropna(axis=1, inplace = True )
```

In [58]:

```
df
```

Out[58]:

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

In [59]:

```
# se não informar o axis, o default é 0 ,axis=0 = linha
df3 = df.dropna(how='any')
df3
```

Out[59]:

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

In [60]:

```
# se não informar o axis, o default é 0 ,axis=0 = linha
df3 = df.dropna(how='any')
df3
```

Out[60]:

	A	B
0	1.0	2.0
1	5.0	6.0
2	10.0	11.0

In [61]:

```
# Tresh - pelo menos um valor da linha não nulo
df = dfb
df.dropna(thresh=1)
```

Out[61]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

Substituir os valores nulos pelo valor médio da coluna

In [79]:

```
# Substituir os valores nulos pelo valor médio da coluna
# https://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn/
# https://stackoverflow.com/questions/38584184/imputer-on-some-dataframe-columns-in-python

df = dfb

from sklearn.preprocessing import Imputer

# gerar uma instancia da classe já que o Imputer é uma instancia
imp = Imputer(missing_values="NaN", strategy="mean", axis = 0 )
```

In [80]:

```
# Fit para definir
# Transform para aplicar
df
```

Out[80]:

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

In [83]:

```
# Fit para definir
imp = imp.fit(df.values)
```

In [84]:

```
# Transform para aplicar
imputed_data = imp.transform(df.values)
```

In [85]:

```
imputed_data
```

Out[85]:

```
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 5. ,  6. ,  7.5,  8. ],
       [10. , 11. , 12. ,  6. ]])
```

Lidando com valores nominais / categoricos

In [104]:

```
df = pd.DataFrame([["green", "M",10.1,"class1"],
                   ["red", "L", 13.5,"class2"],
                   ["blue", "XL", 15.3, "class1"]])
```

In [105]:

```
df.columns = ["color", "size", "price", "classlabel"]
```

In [106]:

```
dfb = df
df
```

Out[106]:

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class2
2	blue	XL	15.3	class1

Codificar a coluna Size (transformar num numero)

In [107]:

```
# Codificar a coluna Size  (transformar num numero)
# Criar um dicionario
size_mapping = {"XL": 3,
                "L": 2,
                "M": 1}
size_mapping
```

Out[107]:

```
{'XL': 3, 'L': 2, 'M': 1}
```

In [108]:

```
# Aplicar um dicionario a uma coluna
df["size"] = df["size"].map(size_mapping)
df
```

Out[108]:

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

In [116]:

```
# Criar um dicionario
class_mapping = {"class1": "vestido",
                 "class2": "camisa"}
class_mapping.items()
```

Out[116]:

```
dict_items([('class1', 'vestido'), ('class2', 'camisa')])
```

In [110]:

```
# Aplicar um dicionario a uma coluna
df["classlabel"] = df["classlabel"].map(class_mapping)
df
```

Out[110]:

	color	size	price	classlabel
0	green	1	10.1	vestido
1	red	2	13.5	camisa
2	blue	3	15.3	vestido

Desafio: como fazer o mapeamento inverso do código para o valor do texto?

In [114]:

```
# Desafio: como fazer o mapeamento inverso do código para o valor do texto?
# temos de fazer o contrario do exercicio anterior
```

In [115]:

```
size_mapping.items()
```

Out[115]:

```
dict_items([('XL', 3), ('L', 2), ('M', 1)])
```

In [117]:

```
# Inverte o dicionario
inv_size_mapping = {v: k for k, v in size_mapping.items()}
```

In [118]:

```
inv_size_mapping
```

Out[118]:

```
{3: 'XL', 2: 'L', 1: 'M'}
```

In [119]:

```
# Aplicar um dicionario a uma coluna, revertendo
df["size"] = df["size"].map(inv_size_mapping)
df
```

Out[119]:

	color	size	price	classlabel
0	green	M	10.1	vestido
1	red	L	13.5	camisa
2	blue	XL	15.3	vestido

In [120]:

```
# Gerar o dicionario automaticamente
import numpy as np
```

In [121]:

```
np.unique(df["classlabel"])
```

Out[121]:

```
array(['camisa', 'vestido'], dtype=object)
```

In [122]:

```
enumerate(np.unique(df["classlabel"]))
```

Out[122]:

```
<enumerate at 0x242a3c0ee58>
```

In [124]:

```
# criar as tuplas ( sequencia de valores, igual a uma lista)
list(enumerate(np.unique(df["classlabel"])))
```

Out[124]:

```
[(0, 'camisa'), (1, 'vestido')]
```

In [125]:

```
# criar o dicionario automaticamente
dict_class = {label: idx for idx, label in enumerate(np.unique(df["classlabel"]))}
```

In [126]:

```
dict_class
```

Out[126]:

```
{'camisa': 0, 'vestido': 1}
```

Usar o Encoder para substituir automaticamente o que fizemos no exercicio anterior (codificar o texto)

In [127]:

```
# Usar o Encoder para substituir automaticamente o que fizemos no exercicio anterior (codif
from sklearn.preprocessing import LabelEncoder
```

In [133]:

```
# gerar uma instancia da classe já que o LabelEncoder é uma instancia
encoder = LabelEncoder()
```

In [134]:

```
df["classlabel"].values
```

Out[134]:

```
array(['vestido', 'camisa', 'vestido'], dtype=object)
```

In [135]:

```
# Fit e Transformer para transformar os valores
result = encoder.fit_transform(df["classlabel"].values)
```

In [136]:

```
result
```

Out[136]:

```
array([1, 0, 1], dtype=int64)
```

In [137]:

```
df
```

Out[137]:

	color	size	price	classlabel
0	green	M	10.1	vestido
1	red	L	13.5	camisa
2	blue	XL	15.3	vestido

In [139]:

```
# Aplicar e Transform
df["classlabel"] = encoder.fit_transform(df["classlabel"].values)
```

In [140]:

```
df
```

Out[140]:

	color	size	price	classlabel
0	green	M	10.1	1
1	red	L	13.5	0
2	blue	XL	15.3	1

In [166]:

```
# Transformar tudo em colunas numericas
x = df[["color", "size", "price"]].values
x
```

Out[166]:

```
array([['green', 'M', 10.1],
       ['red', 'L', 13.5],
       ['blue', 'XL', 15.3]], dtype=object)
```

In [167]:

```
encoder2 = LabelEncoder()
```

In [173]:

```
# Encoder na coluna 0
x[:, 0] = encoder2.fit_transform(x[:,0])
# Encoder na coluna 1
x[:, 1] = encoder2.fit_transform(x[:,1])
```

In [174]:

```
x
```

Out[174]:

```
array([[1, 1, 10.1],
       [2, 0, 13.5],
       [0, 2, 15.3]], dtype=object)
```

In [175]:

```
from sklearn.preprocessing import OneHotEncoder
```

In [176]:

```
one_h_encoder = OneHotEncoder(categorical_features = [0])
```

In [177]:

```
one_h_encoder.fit_transform(x).toarray()
```

Out[177]:

```
array([[ 0. ,  1. ,  0. ,  1. , 10.1],
       [ 0. ,  0. ,  1. ,  0. , 13.5],
       [ 1. ,  0. ,  0. ,  2. , 15.3]])
```

In [179]:

```
one_h_encoder2 = OneHotEncoder(categorical_features = [0], sparse = False)
```

In [180]:

```
one_h_encoder2.fit_transform(x)
```

Out[180]:

```
array([[ 0. ,  1. ,  0. ,  1. , 10.1],
       [ 0. ,  0. ,  1. ,  0. , 13.5],
       [ 1. ,  0. ,  0. ,  2. , 15.3]])
```

One-hot encoding com pandas

In [182]:

```
df[["price", "color", "size"]]
```

Out[182]:

	price	color	size
0	10.1	green	M
1	13.5	red	L
2	15.3	blue	XL

In [183]:

```
# transformar categorical em colunas com 0 ou 1
pd.get_dummies(df[["price","color","size"]])
```

Out[183]:

	price	color_blue	color_green	color_red	size_L	size_M	size_XL
0	10.1	0	1	0	0	1	0
1	13.5	0	0	1	1	0	0
2	15.3	1	0	0	0	0	1

In [187]:

```
# Excluiu  a primeira coluna categorical que ele transforma
# exclui a primeira porque bastam as outras categorical (caso não seja nenhuma das categor
pd.get_dummies(df[["price","color","size"]], drop_first = True)
```

Out[187]:

	price	color_green	color_red	size_M	size_XL
0	10.1	1	0	1	0
1	13.5	0	1	0	0
2	15.3	0	0	0	1

In [ ]:

```

```