APRENDIZAGEM DE MÁQUINA

PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br
https://github.com/josenalde/machinelearning

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

PREPARAÇÃO/CONHECIMENTO DOS DADOS

Limpeza (observações duplicadas? valores ausentes (nan)? presença de outliers? (boxplot)
Dados que não contribui com o aprendizado (id, nomes...) e inconsistentes (sabidamente errados: peso 120 idade 3)
Dados redundantes: exemplo data nascimento e idade na mesma base

Transformação (padronização (StandardScaler) ou escalonamento (Min-MaxScaler))?
Balanceamento das bases

Codificação de colunas categóricas?

Análise Exploratória – tipos de dados (info), faixas de valores (describe), correlações de features (pairplot, corr, heatmap)

[1] Caracterizado tipo de problema? Classificação, regressão, associação, agrupamento?

[2] Engenharia de feature (agregar/transformar features?), criar features? Selecionar features (transformadas)

[3] Selecionar modelo (ou modelos, se for para comparar), sintonia de hiperparâmetros (gridsearch)
[4] Treinos e validações cruzadas (métricas, matriz de confusão (se classificação))
[5] Uma vez selecionado modelo "final" pensar nas estratégias de implantação e necessidades de retreino em produção (frequência, tipo (batch, online), pipeline para novos dados)

LIMPEZA: DADOS FALTANTES (MISSING VALUES)

Um dos métodos mais utilizados é o **fillna**, onde podem ser passados valores constantes, operações entre os valores, mapeamentos etc. sendo bastante versátil

	0	1	2
0	0.667995	-1.762965	0.639751
1	2.033357	-0.629081	-0.531672
2	-0.286754	0.181214	0.484688
3	-1.851530	-0.876192	2.399583
4	-0.294887	-0.566266	2.266865
5	-1.610551	0.548917	-0.475550
6	-1.801883	-0.680614	-0.833749

```
# criar toy-dataframe para estes exemplos
df2 = pd.DataFrame(data=np.random.randn(7,3))
```

```
1 df2.iloc[:4,1] = np.nan
2 df2.iloc[:2,2] = np.nan
3 df2
```

	0	1	2
0	0.667995	NaN	NaN
1	2.033357	NaN	NaN
2	-0.286754	NaN	0.484688
3	-1.851530	NaN	2.399583
4	-0.294887	-0.566266	2.266865
5	-1.610551	0.548917	-0.475550
6	-1.801883	-0.680614	-0.833749

```
df2.fillna(0)
```

	0	1	2
0	0.667995	0.000000	0.000000
1	2.033357	0.000000	0.000000
2	-0.286754	0.000000	0.484688
3	-1.851530	0.000000	2.399583
4	-0.294887	-0.566266	2.266865
5	-1.610551	0.548917	-0.475550
6	-1.801883	-0.680614	-0.833749

```
df2.fillna({1:0.5, 2: -1})
```

	0	1	2
0	0.667995	0.500000	-1.000000
1	2.033357	0.500000	-1.000000
2	-0.286754	0.500000	0.484688
3	-1.851530	0.500000	2.399583
4	-0.294887	-0.566266	2.266865
5	-1.610551	0.548917	-0.475550
6	-1.801883	-0.680614	-0.833749

```
1 df2.fillna(df2.loc[:,0].mean())
```

	0	1	2
0	0.667995	-0.449179	-0.449179
1	2.033357	-0.449179	-0.449179
2	-0.286754	-0.449179	0.484688
3	-1.851530	-0.449179	2.399583
4	-0.294887	-0.566266	2.266865
5	-1.610551	0.548917	-0.475550
6	-1.801883	-0.680614	-0.833749

```
# verificar quantos dados faltantes por coluna
df2.isna().sum() # df2.isnull().sum().sum(), df2.isnull().any(), df2.isnull().values.any()
```

LIMPEZA: DADOS FALTANTES (MISSING VALUES)

Algumas estratégias:

- Ignorar os registros (eliminar) - **dropna**
- Completar manualmente (não assegura padrão, nem sempre é fácil)
- Constante global para todos
- Usar medidas de tendência central (como médias) do atributo ou de outros atributos
- Usar o valor mais provável, mais frequente (moda, por exemplo em colunas categóricas). Exemplo, preenchendo dados faltantes da coluna Embarked do dataset Titanic com o valor mais frequente, no caso 'S'

```
# fill missing values using mode of the categorical column
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
```

Usando a classe SimpleInputer

```
#4. Preencher com a média

from sklearn.impute import SimpleImputer
import numpy as np

impute_mean = SimpleImputer(missing_values = np.nan, strategy='mean')

df[['salario', 'bonus']] = impute_mean.fit_transform(df[['salario', 'bonus']])
```

LIMPEZA: EXEMPLOS DROPNA

```
data_dict = {
             "Data":    ["12/11/2020","13/11/2020","14/11/2020","15/11/2020","16/11/2020","17/11/2020"],
             "Aberto" : [1, 2, np.nan, 4, 5, 7],
             "Fechado": [5, 6, 7, 8, 9, np.nan],
             "Volume":  [np.nan, 200, 300, 400, 500, 600]
            }
df3 = pd.DataFrame(data=data_dict)
df3
```

	Data	Aberto	Fechado	Volume
0	12/11/2020	1.0	5.0	NaN
1	13/11/2020	2.0	6.0	200.0
2	14/11/2020	NaN	7.0	300.0
3	15/11/2020	4.0	8.0	400.0
4	16/11/2020	5.0	9.0	500.0
5	17/11/2020	7.0	NaN	600.0

```
df3.dropna(axis=1)
```

	Data
0	12/11/2020
1	13/11/2020
2	14/11/2020
3	15/11/2020
4	16/11/2020
5	17/11/2020

```
df3.dropna(axis=0) # remove linhas que em qualquer coluna tenha NaN
```

	Data	Aberto	Fechado	Volume
1	13/11/2020	2.0	6.0	200.0
3	15/11/2020	4.0	8.0	400.0
4	16/11/2020	5.0	9.0	500.0

```
df3.dropna(how='all')
```

	Data	Aberto	Fechado	Volume
0	12/11/2020	1.0	5.0	NaN
1	13/11/2020	2.0	6.0	200.0
2	14/11/2020	NaN	7.0	300.0
3	15/11/2020	4.0	8.0	400.0
4	16/11/2020	5.0	9.0	500.0
5	17/11/2020	7.0	NaN	600.0

```
df3.dropna(subset=["Aberto", "Volume"], inplace=True)
```

```
# ao fim df3.dropna(inplace=True) para concluir as alterações
#df3.dropna(inplace=True)
df3
```

	Data	Aberto	Fechado	Volume
1	13/11/2020	2.0	6.0	200.0
3	15/11/2020	4.0	8.0	400.0
4	16/11/2020	5.0	9.0	500.0
5	17/11/2020	7.0	NaN	600.0

LIMPEZA: VALORES DUPLICADOS

- Observações duplicadas impactam treinamento e na aleatoriedade da separação treino/teste/validação podem aparecer em mais de um conjunto!

Uso do método duplicated, que retorna Series booleana com ocorrências

```
1 df = pd.DataFrame({'k1': ['um', 'dois']*3 + ['dois'],
2                    'k2': [1,1,2,3,3,4,4]
3                   })
4 df
```

	k1	k2
0	um	1
1	dois	1
2	um	2
3	dois	3
4	um	3
5	dois	4
6	dois	4

```
1 df.duplicated()
```

```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

Neste caso, há um método de tratamento drop_duplicates(), que remove linhas duplicadas

Pode-se também apagar duplicadas com base em colunas específicas

Qual será a saída de drop_duplicates(['k1'])?

LIMPEZA: OUTLIERS

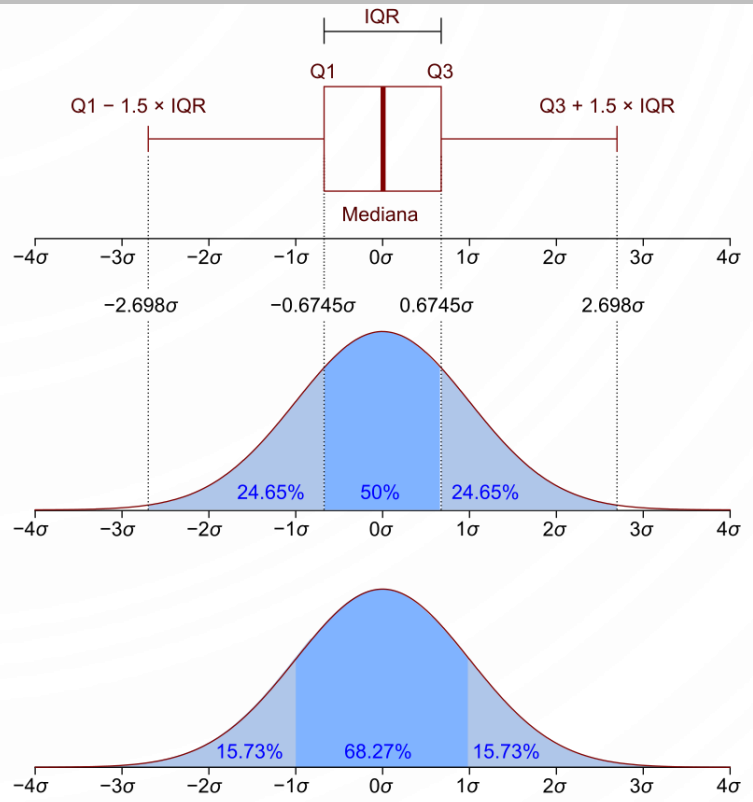
- O boxplot é o gráfico indicado para esta detecção (df['coluna'].plot(kind='box')) ou seaborn

- Tratamento de OUTLIERS

Exemplo: IIQ ou IQR (Intervalo Interquartil)

IQR = Q3 – Q1
lowV = Q1 – 1.5xIQR
highV = Q3 + 1.5xIQR

Se x > highV, x = highV
Se x < lowV, x = lowV



LIMPEZA: OUTLIERS

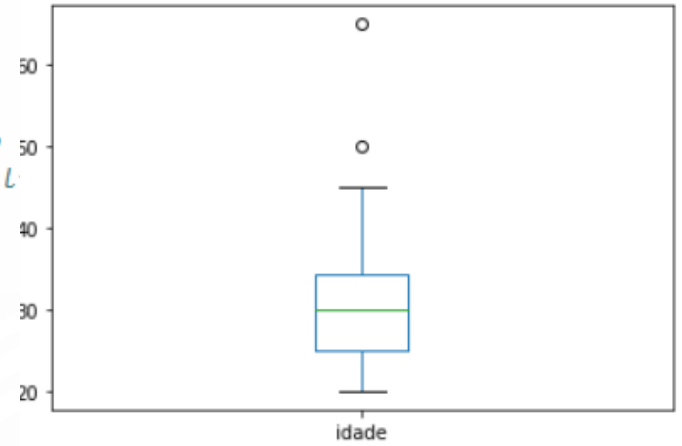
- O boxplot é o gráfico indicado para esta detecção (df['coluna'].plot(kind='box')) ou seaborn

- Tratamento de OUTLIERS (exemplo: coluna idade com outliers)

```
#trata outlier pela regra do 1.5*IQR (variação interquartil)

dfTeste = df.copy() #fazendo uma cópia apenas para testar o comando (não será aplicado ao df)

q1 = dfTeste['idade'].quantile(0.25)
q3 = dfTeste['idade'].quantile(0.75)
iqr = q3 - q1
lLim = q1 - 1.5 * iqr
hLim = q3 + 1.5 * iqr
dfTeste.loc[dfTeste['idade'] < lLim, 'idade'] = lLim #substitui os valores abaixo do limite inferior pelo
dfTeste.loc[dfTeste['idade'] > hLim, 'idade'] = hLim #substitui os valores acima do limite superior pelo l

print(dfTeste)
```

```
df['idade'].plot(kind = 'box')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f786
```



TRANSFORMAÇÃO ENTRE ATRIBUTOS NUMÉRICOS

- Quando os atributos numéricos estão em escalas diferentes pode fazer com que o algoritmo de AM classifique erradamente com base apenas nos atributos com escala maior, salvo raras exceções.

- Para resolver esse problema, devemos escalonar os dados, a fim de que todos os atributos fiquem em uma mesma escala. Geralmente não é necessário escalonar os valores alvo (target)
- Inicialmente, transformar só os dados de treinamento. Só depois os de teste e os dados novos
- Muito importante principalmente para algoritmos baseados em distância (knn, svm,...)

•O **escalonamento por reescala (MinMax, normalização)** dimensiona cada variável de entrada separadamente para o intervalo 0:1 (ou -1:1 quando tem valores negativos). Admite um hiperparâmetro *feature_range* se você desejar outra faixa.

$$vNovo = \min + \frac{vAtual - menor}{maior - menor}(\max - \min)$$

Redes neurais, por exemplo, em geral esperam valores entre 0 e 1

•O **escalonamento por padronização (Standard)** dimensiona cada variável de entrada separadamente para ter uma média 0 e um desvio padrão 1 (*distribuição normal padrão*).

$$vNovo = \frac{vAtual - \mu}{\sigma}$$

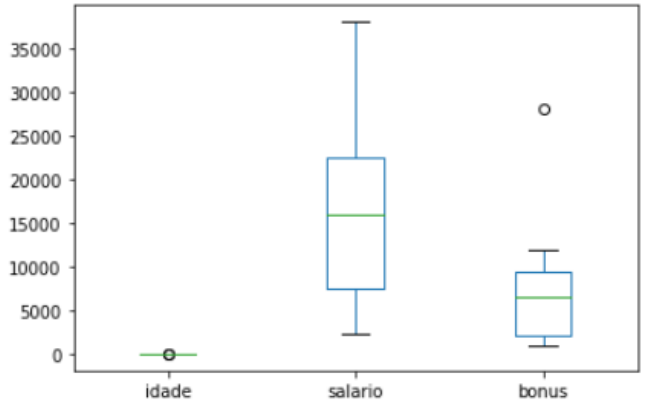
Não limita faixa de valores, não é tão afetada pelos outliers

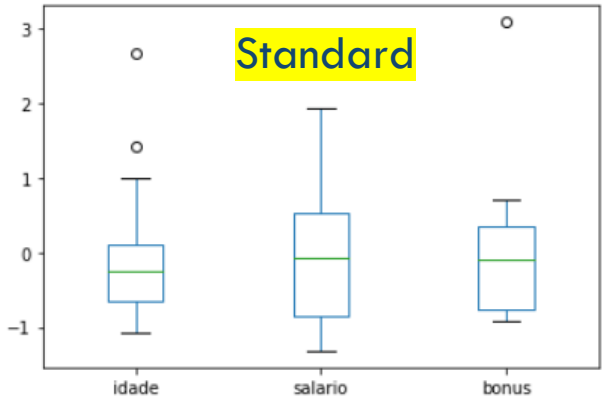
Dica: Use normalização por reescala quando seus dados não tiverem distribuição normal e você desejar *manter a distribuição original*. A normalização por padronização trará seus dados para uma distribuição normal.

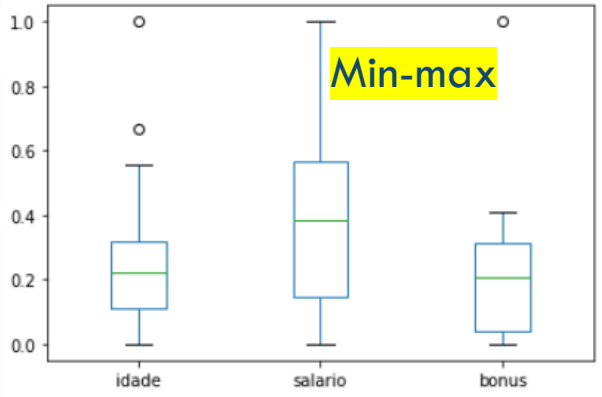
TRANSFORMAÇÃO ENTRE ATRIBUTOS NUMÉRICOS

- O **escalonamento robusto (RobustScaler)** dimensiona com base no intervalo interquartil já tratando os outliers

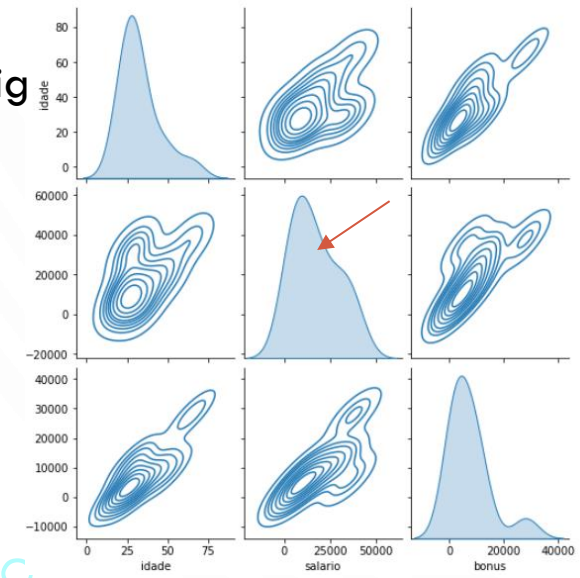
Exemplos de dados originais com escalas bem diferentes:

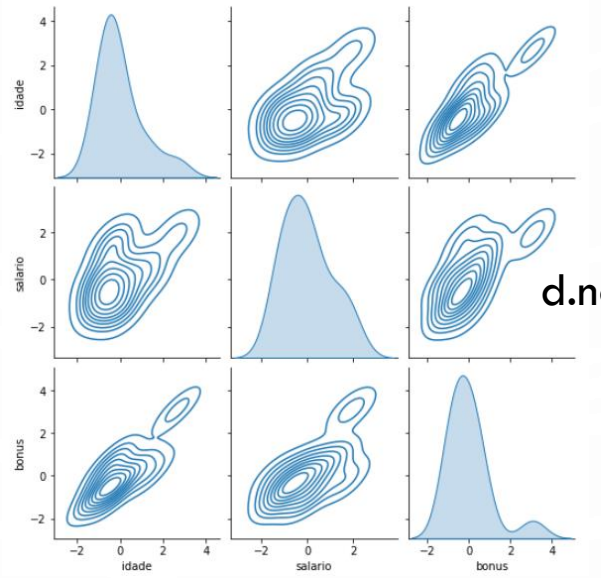




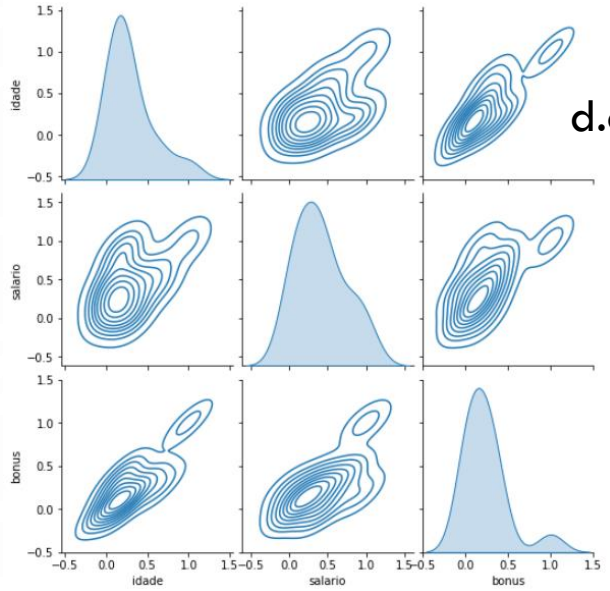


d.orig





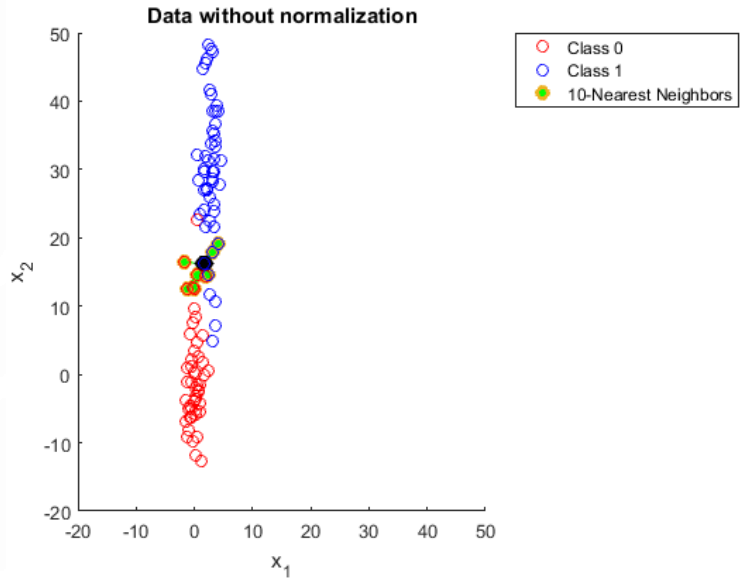
d.normal

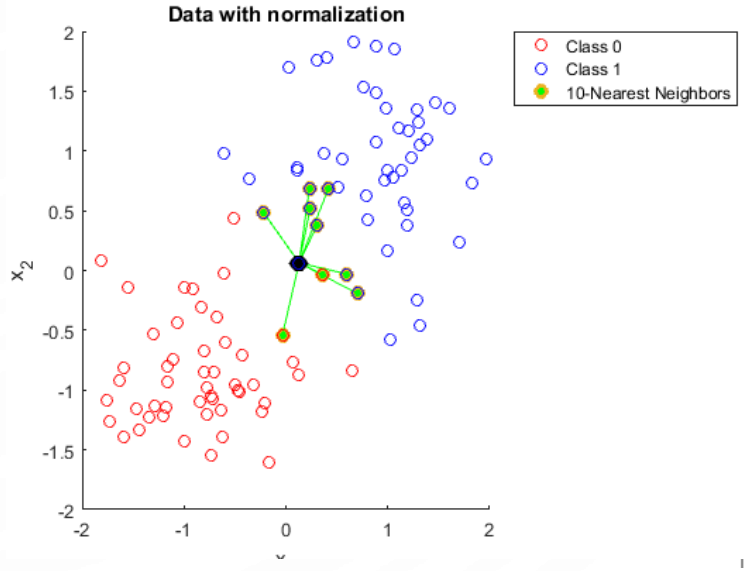


d.orig

TRANSFORMAÇÃO ENTRE ATRIBUTOS NUMÉRICOS

Exemplo de efeito no algoritmo KNN:





CODIFICAÇÃO DE DADOS CATEGÓRICOS

- Encoder Ordinal (importante para o treino a ponderação)
- Label Encoder
- OneHot Encoder (get_dummies: pandas)

Alguns algoritmos de AM só lidam com dados numéricos

Nesses casos, quando a base possui dados simbólicos, faz-se necessário converter o tipo do dado

Se o dado simbólico for ordinal, ou seja, for passível de ordenação, pode-se fazer uma conversão simples para valores inteiros ordenáveis

Caso o dado simbólico seja nominal, ou seja, **se não houver relação de ordem, então deve-se convertê-lo de forma que a diferença entre quaisquer dois valores do atributo seja sempre a mesma**

- Uma forma de conseguir isso é codificar cada valor nominal por uma sequência de c bits, onde **c é igual a quantidade de valores possíveis para aquele atributo**
- Dessa forma, se usarmos a distância de **Hamming** entre os valores convertidos, teremos que para quaisquer dois valores, a distância é 2

Atributo nominal	Código 1 – de – c
Azul	100000
Amarelo	010000
Verde	001000
Preto	000100
Marrom	000010
Branco	000001

PIPELINE DE TRATAMENTOS

• Para aplicar as transformações na sequência correta podemos usar pipelines do sklearn

• Seja um dataset df, exemplo de aplicação sucessiva de preenchimento de dados faltantes com a mediana e posterior aplicação de padronização standard

```
mypipeline = Pipeline([
             (‘imputer’, SimpleInputer(strategy=‘median’)),
             (‘std_scaler’, StandardScaler()),
             [)
df_tr = mypipeline.fit_transform(df)
```

Podemos usar de sklearn.compose a classe ColumnTransformer para aplicar transformações nas colunas numéricas e categóricas simultaneamente.

CONVERSÃO NUMÉRICO-CATEGÓRICO

- Pode ser necessário converter faixas de valores para grupos/categorias (como faixas etárias, salariais, etc.)

- Sugestão do pandas: método **cut**

Nesse caso, **os atributos devem ser discretizados**, ou seja, deve-se definir intervalos de valores para o atributo e cada intervalo deve receber um rótulo

Estratégias:

Larguras iguais: Divide o intervalo original em subintervalos com mesma largura

Algoritmos de agrupamento

CONVERSÃO NUMÉRICO-CATEGÓRICO

- DISCRETIZAÇÃO E COMPARTIMENTALIZAÇÃO (BINS): agrupar dados em conjuntos para análise

Exemplo: seja um conjunto de idades. Vamos agrupar nas classes
Jovem: 18,25
Jovem Adulto: 26,35
Adulto: 36, 60
Senior: 61 em diante

- Vamos utilizar o método CUT

```
1 idades = [20,22,25,27,21,23,37,31,61,45,41,32]
```

```
[0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1]
```

```
1 intervalos = [18,25,35,60,100]
```

Supondo não haver > 100 anos, mas pode ser adaptado...

```
1 categorias = pd.cut(idades,intervalos)
```

```
1 categorias
```

```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

Aloca cada dado de idades numa categoria intervalar

.codes/ .categories

```
0: (18, 25],
1: (25, 35],
2: (35, 60],
3: (60, 100]
```

CONVERSÃO NUMÉRICO-CATEGÓRICO

- DISCRETIZAÇÃO E COMPARTIMENTALIZAÇÃO (BINS): agrupar dados em conjuntos para análise

Exemplo: seja um conjunto de idades. Vamos agrupar nas classes
Jovem: 18,25
Jovem Adulto: 26,35
Adulto: 36, 60
Senior: 61 em diante

- Vamos utilizar o método CUT (ver variação QCUT)

```
1 label_categ = ['jovem', 'jovem adulto', 'adulto', 'senior']
```

```
1 classes = pd.cut(idades, intervalos, labels=label_categ)
```

```
1 classes
```

```
['jovem', 'jovem', 'jovem', 'jovem adulto', 'jovem', ..., 'jovem adulto', 'senior', 'adulto', 'adulto', 'jovem adulto']
Length: 12
Categories (4, object): ['jovem' < 'jovem adulto' < 'adulto' < 'senior']
```

```
1 pd.value_counts(classes)
```

```
jovem          5
jovem adulto   3
adulto         3
senior         1
dtype: int64
```