Dummificação.

Os algoritmos usados pelo *sklearn* não aceitam valores categóricos.

Quando falamos em valores categóricos, precisamos tomar cuidado em não achar que é apenas quando se tem letras. Variáveis com números (0 e 1, por exemplo) também podem representar valores categóricos. Nesses casos, os valores podem ser substituídos por Falso e Verdadeiro, respectivamente.

No caso da base de pacientes com câncer de mama, por exemplo, temos isso na variável *Metastasis*, sendo que quando o valor for 0, o paciente não está em metástase e quando o valor for 1, o paciente está em metástase.

Se considerarmos os números 0 e 1, existe uma relação onde devemos considerar 1 maior do que 0 e, consequentemente, quem tem metástase teria vantagem sobre quem não tem. Ou ainda que quem tem metástase tem algo a mais do que quem não tem. Por isso, transformar as strings '0' e '1' em números 0 e 1 não teria sentido.

Assim como não teria sentido transformar em números uma variável categórica que represente o estado onde uma pessoa mora. Teríamos, por exemplo, Bahia como 1, Rio Grande do Sul como 2, São Paulo como 3, Sergipe como 4 e assim por diante. Esses números representam relações que não fariam sentido para o caso. Se a gente criar um modelo para ajudar médicos a definirem a doença de um paciente, por exemplo, quando o modelo estiver criado e sendo usado já em produção, quando a doença de um novo paciente for definida, se o estado do paciente tiver código 5 ou 6, o algoritmo tende a aproximar ele mais das doenças dos pacientes de Aracaju do que das doenças dos de Salvador. Esse método de transformar variáveis categóricas em numéricas é chamado de *label encoding* e seria útil em alguns casos, que não veremos nesse material.

Assim, nos dois casos descritos, a melhor solução seria o uso do método *one-hot-enconding* (*dummy*), que faz a transformação da seguinte forma: para cada linha da base original vai existir uma linha na base modificada com *n* colunas, sendo *n* o número de possíveis valores do campo na base original. Cada coluna na base modificada vai ter o nome do campo da base original +"_"+ cada possível valor para aquela coluna na base original.

A cada linha, das *n* colunas, apenas uma delas vai ter valor 1, que vai ser aquela definida como o nome do campo da base original +"_"+ o valor do campo na linha correspondente da base original.

O caso da metástase, originalmente, seria como na tabela 1, por exemplo:

ID do Paciente	Metástase
1	0
2	1
3	1

Tabela 1

Depois da *dummificação*, ficou como na tabela 2:

ID do Paciente	Metástase_0	Metástase_1
1	1	0
2	0	1
3	0	1

Tabela 2

Já o dos estados, originalmente, seria como na tabela 3:

ID do Paciente	Estado
1	Bahia
2	São Paulo
3	Sergipe
4	Bahia

Tabela 3

Depois da *dummificação*, ficou como na tabela 4:

ID do Paciente	Estado_Bahia	Estado_São Paulo	Estado_Sergipe	Estado_Rio Grande do Sul
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	1	0	0	0

Tabela 4

Assim, quando na base original o valor for Bahia, o campo *Estado_Bahia* vai ser igual a 1 e *Estado_São Paulo*, *Estado_Sergipe* e *Estado_Rio Grande do Sul* terão valor 0.

Essa transformação é feita de maneira automática pela biblioteca pandas. Você precisa apenas passar a base onde deseja aplicar o processo. Aqui, vamos usar os atributos *Metastasis* e *Menopause* como exemplo. Usando os comandos da figura 1, temos a base que será usada aqui no exemplo.

```
base['Menopause'] = base['Menopause'].astype(str)
base_dummies = base[['Menopause','Metastasis']]
display(base_dummies)
```

Figura 1

Como a variável *Menopause* está como números inteiros, inicialmente vamos converter ela para string. Se você colocar o algoritmo para treinar com Menopause com valores inteiros, ele vai funcionar, mas não da maneira correta, pois os valores não fariam sentido, como dito anteriormente.

A base **sem *dummificação*** teria os 5 primeiros registros como na figura 2:

	Menopause	Metastasis
0	1	0
1	1	0
2	0	0
3	1	0
4	1	0

Figura 2

Usamos o comando *get_dummies* para realizar o processo de *dummificação* (ou one-hot-encoding):

```
base_dummies = pd.get_dummies(base_dummies, dtype=int)
display(base_dummies)
```

Figura 3

A *base_dummies* **após a *dummificação*** teria os 5 primeiros registros como na figura 4:

	Menopause_0	Menopause_1	Metastasis_#	Metastasis_0	Metastasis_1
0	0	1	0	1	0
1	0	1	0	1	0
2	1	0	0	1	0
3	0	1	0	1	0
4	0	1	0	1	0

Figura 4

Veja que na base original o valor do primeiro registro para *Menoupase* é 1. Portanto, na base modificada, *Menoupause_0* é 0 e, *Menoupause_1* é 1. Assim, na base modificada, a coluna que vai ter valor 1 é aquela onde o valor depois do "_" é igual ao da base original. Como na base original *Menopause* é igual a 1, na base dummificada Menopause_1 vai ser igual a 1. Se na base original *Menopause* fosse igual a 0, na base dummificada *Menopause_0* seria igual a 1.

É indicado que o processo seja feito sempre de uma só vez com todos os atributos a serem usados, como nos exemplos acima. Veja que na sua definição (Figura 1), *base_dummies* já tem *Menopause* e *Metastasis*.

A opção seria fazer uma base com *Menopause* e logo após a *dummificação* e depois outra base com *Metastasis* e outra *dummificação*. Depois disso, as duas bases *dummificadas* seriam concatenadas e colocadas na base que vai ser passada para o modelo.

Se isso for feito e cada base separada for passada como do tipo *Series*, o pandas não vai colocar o nome do atributo nas novas colunas da nova base. Se nós fizéssemos o processo apenas com o atributo *Menopause*, por exemplo, como resultado, ao invés de *Menoupause_0* e *Menopause_1*, ele retornaria as colunas 0 e 1. Para *Metastases* o procedimento funcionaria da mesma forma, retornando os atributos 0, 1 e #. Veja no exemplo da figura 5:

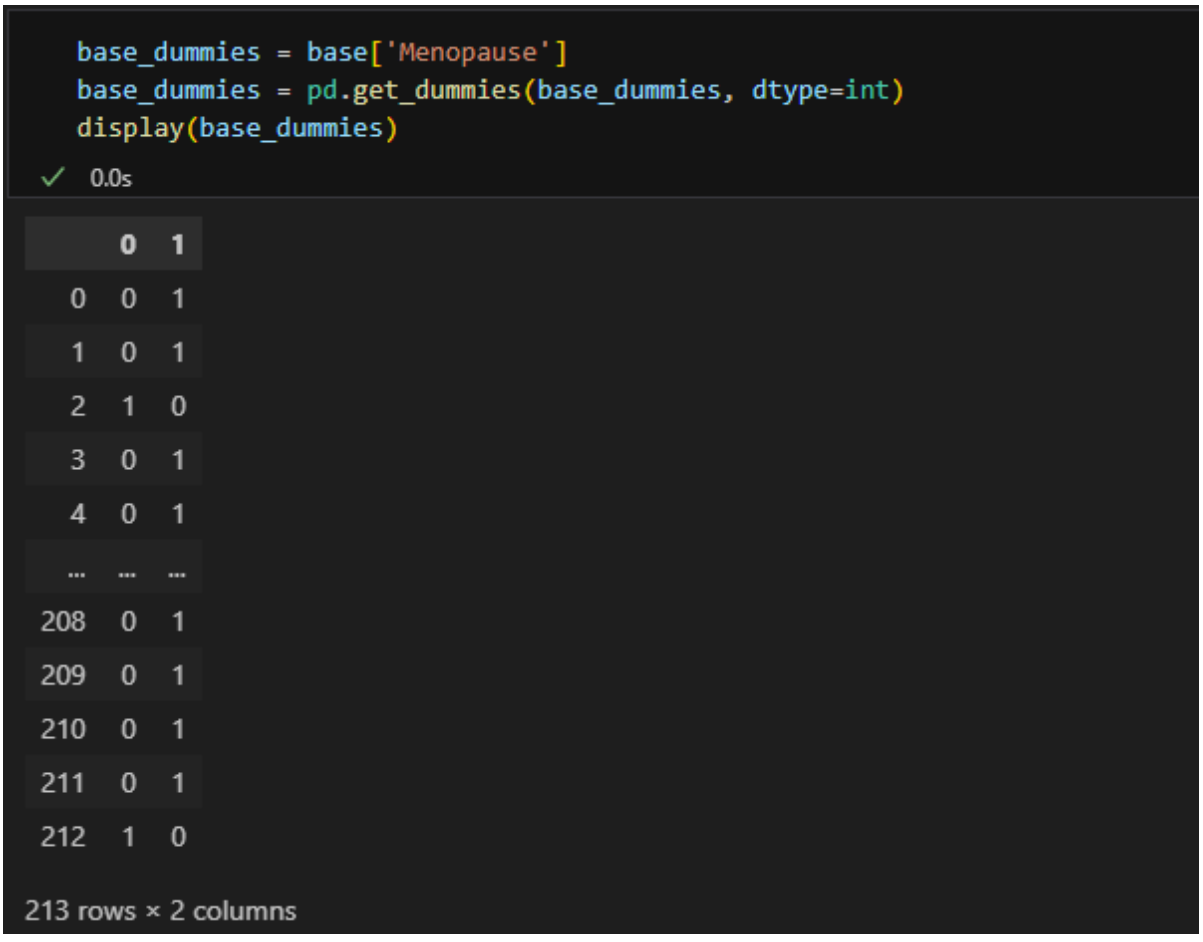

Figura 5

A *base_dummies* foi passada como *Series* porque apenas um colchete ([]) foi usado na sua definição. Isso pode ser conferido usando o comando type, como na figura 6:

```
type(base_dummies)
```
✓ 0.0s

```
pandas.core.series.Series
```

Figura 6

Assim, caso você faça o processo separado usando as bases como *Series*, causaria uma confusão com os nomes das colunas, que seriam os mesmos:

```
base_dummies_menopause = base['Menopause']
base_dummies_menopause = pd.get_dummies(base_dummies_menopause, dtype=int)
base_dummies_metastasis = base['Metastasis']
base_dummies_metastasis = pd.get_dummies(base_dummies_metastasis, dtype=int)
base_dummies = pd.concat([base_dummies_menopause,base_dummies_metastasis], axis = 1)
display(base_dummies)
```
✓ 0.0s

	0	1	#	0	1
0	0	1	0	1	0
1	0	1	0	1	0
2	1	0	0	1	0
3	0	1	0	1	0
4	0	1	0	1	0
...	...	...	...	...	...
208	0	1	0	0	1
209	0	1	0	1	0
210	0	1	0	1	0
211	0	1	0	1	0
212	1	0	0	1	0

213 rows × 5 columns

Figura 7

Isso seria resolvido usando as bases como dataframe, colocando sempre dois colchetes([[]]) nas definições das bases, como na figura 8:

```
base_dummies_menopause = base[['Menopause']]
base_dummies_menopause = pd.get_dummies(base_dummies_menopause, dtype=int)
base_dummies_metastasis = base[['Metastasis']]
base_dummies_metastasis = pd.get_dummies(base_dummies_metastasis, dtype=int)
base_dummies = pd.concat([base_dummies_menopause,base_dummies_metastasis], axis = 1)
display(base_dummies)
```

✓ 0.0s

	Menopause_0	Menopause_1	Metastasis_#	Metastasis_0	Metastasis_1
0	0	1	0	1	0
1	0	1	0	1	0
2	1	0	0	1	0
3	0	1	0	1	0
4	0	1	0	1	0
...	...	...	...	...	...
208	0	1	0	0	1
209	0	1	0	1	0
210	0	1	0	1	0
211	0	1	0	1	0
212	1	0	0	1	0

213 rows × 5 columns

Figura 8