09

Classificação de variáveis categóricas

Classificação de variáveis categóricas

Até agora, já resolvemos alguns problemas de classficação que havíamos visto, porém os dados que nos deparamos, indicavam características dos elementos que estávamos analisando, por exemplo, se o elemento era um cachorro ou um porco verificávamos se esse elemento tinha perna curta, se era gordinho ou se fazia *auau*.

Para todas as características, marcamos entre 0 e 1 indicando se o elemento tinha a característica ou não, ou seja, se ele tinha perna curta, marcávamos como 1 se não 0 e assim sucessivavemente para as demais características.

Perceba que todas as nossas características tiveram apenas 2 tipos de valores, ou seja, 0 ou 1. Além disso, nossas marcações também tiveram apenas 2 valores, 0 ou 1, ou então, -1 ou 1... Observe que, por enquanto, todas as nossas características e marcações tiveram apenas 2 tipos de valores para distinguir os nossos elementos, porém, nem sempre iremos nos deparar com esses tipos de valores para os nossos dados. Iremos verificar um exemplo um pouco diferente ao qual vimos até agora, vamos analisá-lo?

> No Alura, um site de cursos online, nós temos um cliente que visitou a home.

De acordo com o que vimos até agora, como que classificaríamos essa situação? É fácil, certo? Simplesmente marcamos como 0 se não e 1 se visitou. Por enquanto sem nenhuma novidade... Agora esse cliente fez o seguinte:

> Esse cliente já estava logado

E como fazemos para identificar essa informação? Sem segredo também! Marcamos com 1 para indicar que ele estava logado. Vejamos a próxima ação dele:

> E então, esse cliente comprou um curso

Da mesma forma que fizemos anteriormente, ou seja, marcamos com 1 para indicar que ele comprou... Porém, observe essa próxima situação desse mesmo cliente:

> Por fim, esse cliente buscou sobre 'algoritmos'

Como podemos representar essa situação? Será que podemos também marcar com 1 para dizer que sim e 0 para não? E se ele buscasse outro curso? O que faríamos? Faz sentido marcamos como 0 ou 1? Podemos verificar outro exemplo similar:

- Um cliente que entrou na home (0 ou 1)
- Não estava logado (0 ou 1)
- Não comprou (0 ou 1)
- Porém, buscou sobre 'Java' (E agora?)

Perceba que, cada cliente poderia buscar sobre diversos tipos de cursos, assuntos ou tecnologias, por exemplo, HTML, CSS, Javascript, SQL, Ruby ou qualquer outra informação. A questão é, como podemos representar todas essas informações

distintas para uma mesma característica?

Até agora, vimos apenas como distinguir uma características com apenas 2 valores distintos, ou seja, 0 ou 1, acessou ou não, está logado ou não, comprou ou não, porém, para o valor de busca, podemos ter diversos valores, por exemplo, buscou algoritmos ou buscou Java ou buscou Ruby. Vamos verificar algumas possibilidades para esse caso na tabela abaixo:

os clientes

home	busca	estava logado?	comprou?
1	algoritmos	1	1
1	java	0	0
0	java	1	0
1	algoritmos	1	1
0	java	0	1
1	algoritmos	0	0
1	ruby	1	1

Cada coluna refere-se a:

- **home**: se o cliente visitou a página home (0 ou 1).
- **busca**: o que o cliente buscou (algortimos ou Java ou Ruby ou ...).
- **estava logado?**: se o cliente estava logado (0 ou 1).
- **comprou??**: se o cliente comprou ou não (0 ou 1).

Analisando cada coluna, qual é a novidade dentre esses conjuntos de dados? É a coluna `busca`, pois é um tipo de dado que não trabalhamos ainda. Observe que essa coluna é uma informação extremamente rica sobre o nosso usuário, pois ela não diz apenas se o cliente buscou ou não, ou seja, ela diz também o que ele buscou!

Esse tipo de informação é muito importante, pois um cliente que está buscando, por exemplo, "algoritmos", vai ter um comportamento, um outro cliente que busca sobre "Java" vai ter um outro comportamente diferente, e assim sucessivamente! Perceba que podemos conter um conjunto de diversos cursos e que cada um precisa ter o seu comportamento.

Uma situação similar a essa é quando vamos a uma livraria comprar um livro e você quer, por exemplo, um livro de programação, você vai procurar primeiro a seção de "programação" e, nessa seção, terá apenas um conjunto de livros relacionados a programação. E se agora você quiser algum livro de medicina? Novamente procuraria a seção de "medicina" primeiro...

Esse tipo de informação da coluna `busca`, influencia se o cliente vai ou não comprar o produto, serviço ou conteúdo que ele buscou. Porém, a grande pergunta é:

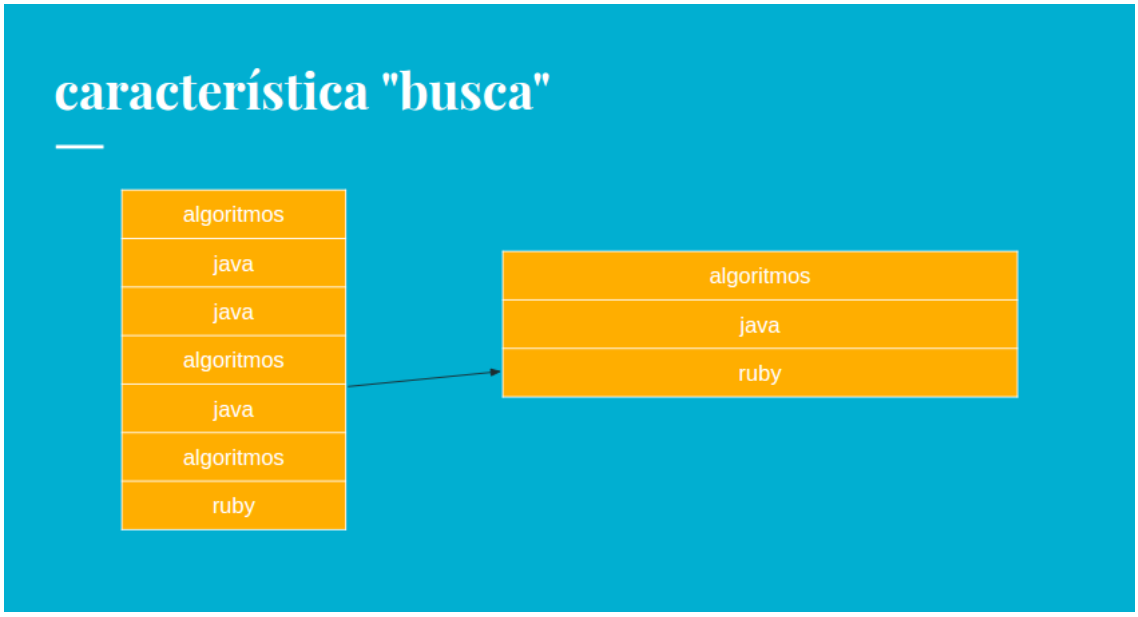
- Como podemos trabalhar com uma coluna que não tenha apenas 2 valores? Por exemplo, diversos valores, ou seja, pode ser "algoritmos" ou "Java" ou "Ruby" e assim por diante.

Como podemos representar esse tipo de informação se até agora trabalhamos com apenas 2 tipos de valores (0 ou 1)? Se nós tentarmos rodar o nosso algoritmo com um texto solto, ele vai ficar doidão e não vai funcionar! E agora? O que podemos fazer?

Precisamos, de alguma forma, transformar essas informações da coluna `busca` em variáveis númericas com valores entre 0 e 1, pois se conseguirmos fazer isso, ou seja, verificar se o cliente só fez busca de "algoritmos" ou "Java" ou "Ruby"... Consegue imaginar como podemos transformar diversos valores diferentes entre 0 e 1?

Mas porque pra 0 e 1? Pois era a forma que estávamos acostumados a trabalhar anteriormente, ou seja, se conseguirmos, resolveremos o nosso problema com um algoritmo que já implementamos!

O nosso desafio agora é reduzir esse nosso problema que contém diversos valores para uma determinada característica em 0 e 1. Será que podemos fazer isso? Vamos começar isolando essa coluna:



Observe que identificamos 3 tipos de características diferentes para essa coluna **busca**:

- Algoritmos
- Java
- Ruby

Cada um dos nossos clientes, fez apenas uma dessas buscas, ou seja, ou ele pesquisou "algoritmos" ou "Java" ou "Ruby". Até agora nenhuma novidade, mas, e se fizermos as seguintes perguntas:

- O cliente buscou algoritmos?
- O cliente buscou Java?
- O cliente buscou Ruby?

Para um cliente que buscou sobre "algoritmos" apenas, como podemos marcar essas perguntas?

- O cliente buscou algoritmos? Sim, acessou (1)
- O cliente buscou Java? Não acessou (0)
- O cliente buscou Ruby? Não acessou (0)

Conseguimos descobrir qual foi a busca que ele fez! Repare que convertemos a pergunta:

- Qual busca o cliente fez?

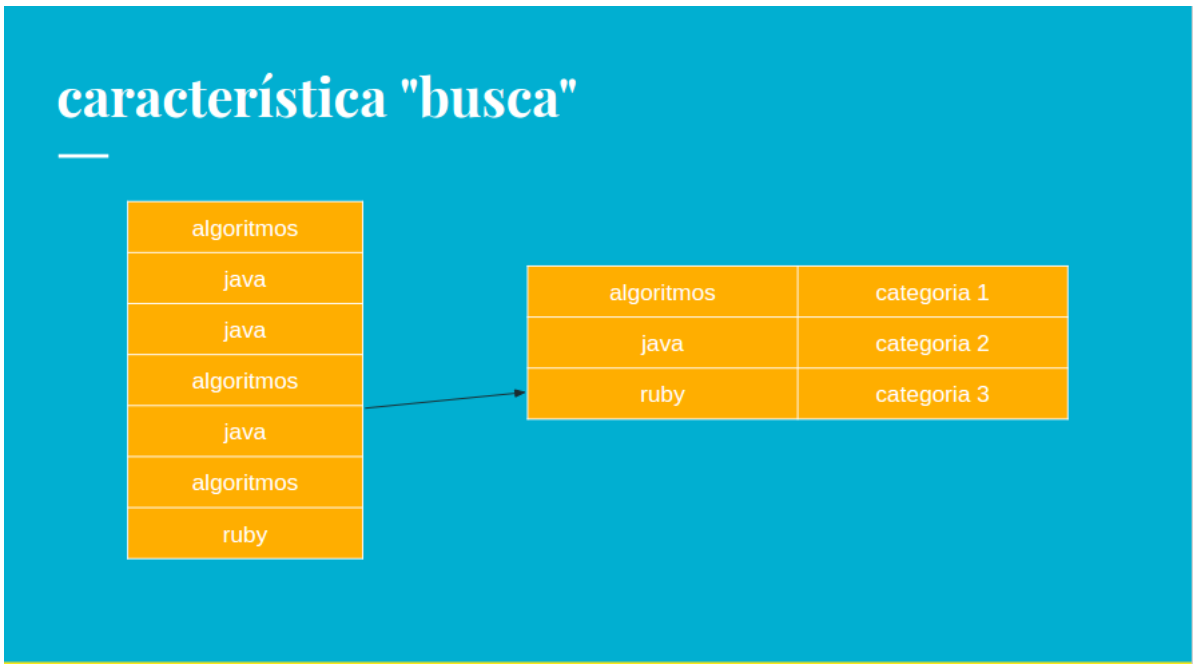
Em 3 perguntas:

- O cliente buscou algoritmos?
- O cliente buscou Java?
- O cliente buscou Ruby?

Vamos considerar agora, um cliente que buscou por "Java" e fazer novamente as 3 perguntas:

- O cliente buscou algoritmos? Não acessou (0)
- O cliente buscou Java? Sim, acessou (1)
- O cliente buscou Ruby? Não acessou (0)

Observe que, novamente conseguimos utilizar as 3 perguntas para classificar o que um outro cliente buscou! Então o que fizemos exatamente? Demos 3 categorias para a coluna `busca`:



Observe que cada característica tem a sua própria categoria:

- Algoritmos - categoria 1
- Java - categoria 2
- Ruby - categoria 3

Isso significa que a nossa variável `busca` não é simplesmente uma variável que atribuímos apenas um valor (0 ou 1), ou seja, podemos atribuir 3 valores para essa mesma variável! Chamamos esse tipo de variável de ***variável categórica***, pois ela possui diversas categorias!

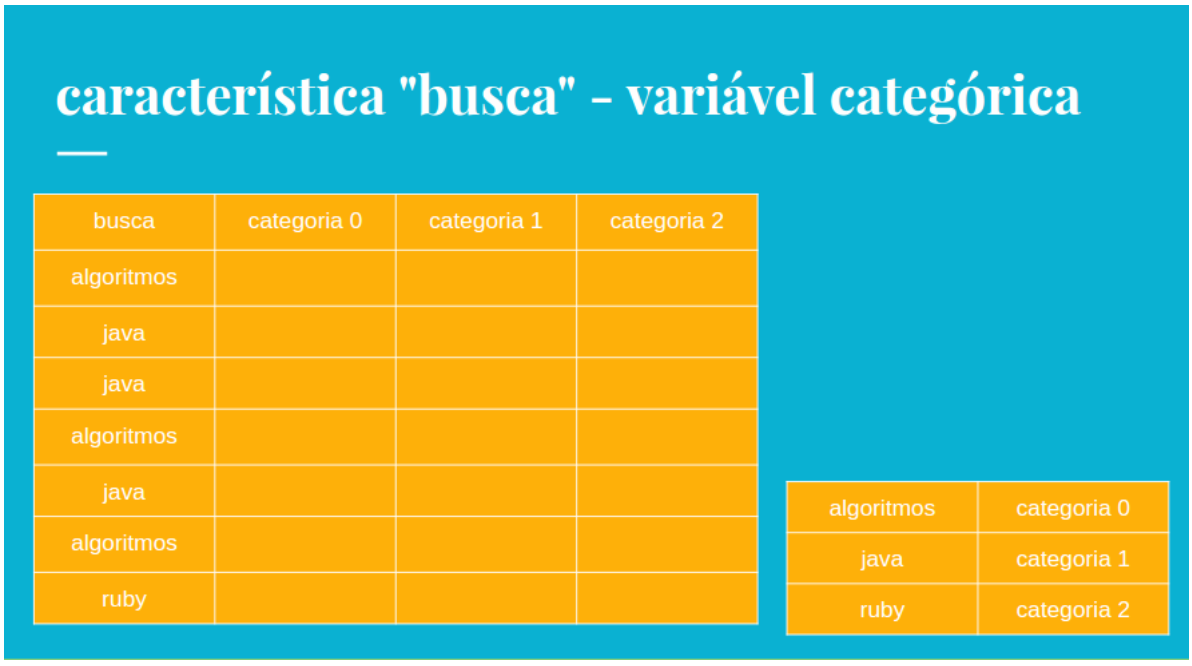
Um outro exemplo que poderíamos utilizar esse tipo de variável seria na seguinte pergunta:

- Em qual estado você está presente nesse instante?

Podemos adicionar todos os estados do Brasil como categorias dessa pergunta, ou seja, São Paulo, Rio de Janeiro, Bahia, Distrito Federal, Pernambuco... Agora que sabemos sobre essa variável categórica, podemos separar a nossa tabela da seguinte forma:

Note que agora adicionamos os valores de 0 a 2 para as categorias, pois utilizaremos arrays para classificar cada uma dessas características.

Mas e na prática? Como poderíamos montar a nossa tabela para cada uma dessas categorias? Vejamos um modelo:



Já tem uma ideia de como podemos preencher essa tabela? Vamos pegar o primeiro exemplo de um cliente que buscou por "algoritmos":

característica "busca" - variável categórica

busca	categoria 0	categoria 1	categoria 2
algoritmos	1	0	0
java			
java			
algoritmos			
java			
algoritmos			
ruby			

algoritmos	categoria 0
java	categoria 1
ruby	categoria 2

Já que ele buscou por algoritmos, significa que ele pesquisou pela "categoria 0", ou seja, marcamos a coluna "categoria 0" com 1 e as demais como 0. E para o cara que pesquisou sobre Java? Como marcamos?

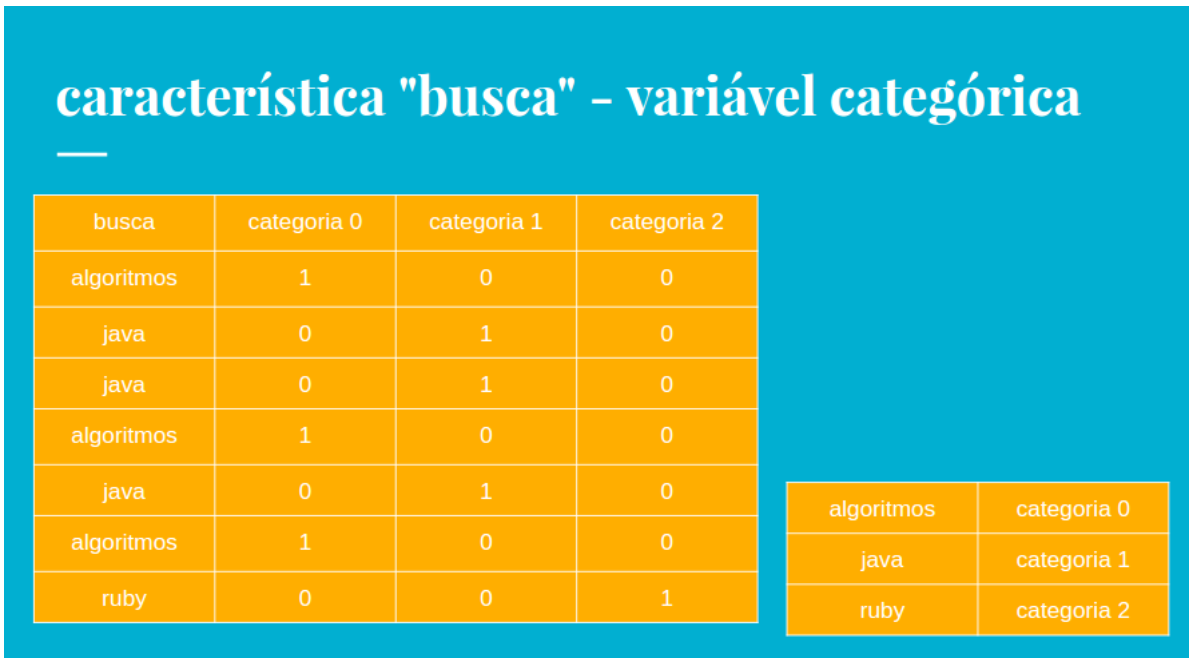
Por enquanto, aprendemos como podemos preencher todos os clientes com categorias 0 e 1, ou seja, categorias para busca de "algoritmos" (0) e "Java" (1). Então vamos preencher todos as buscas de "algoritmos" e "Java":

característica "busca" - variável categórica

busca	categoria 0	categoria 1	categoria 2
algoritmos	1	0	0
java	0	1	0
java	0	1	0
algoritmos	1	0	0
java	0	1	0
algoritmos	1	0	0
ruby			

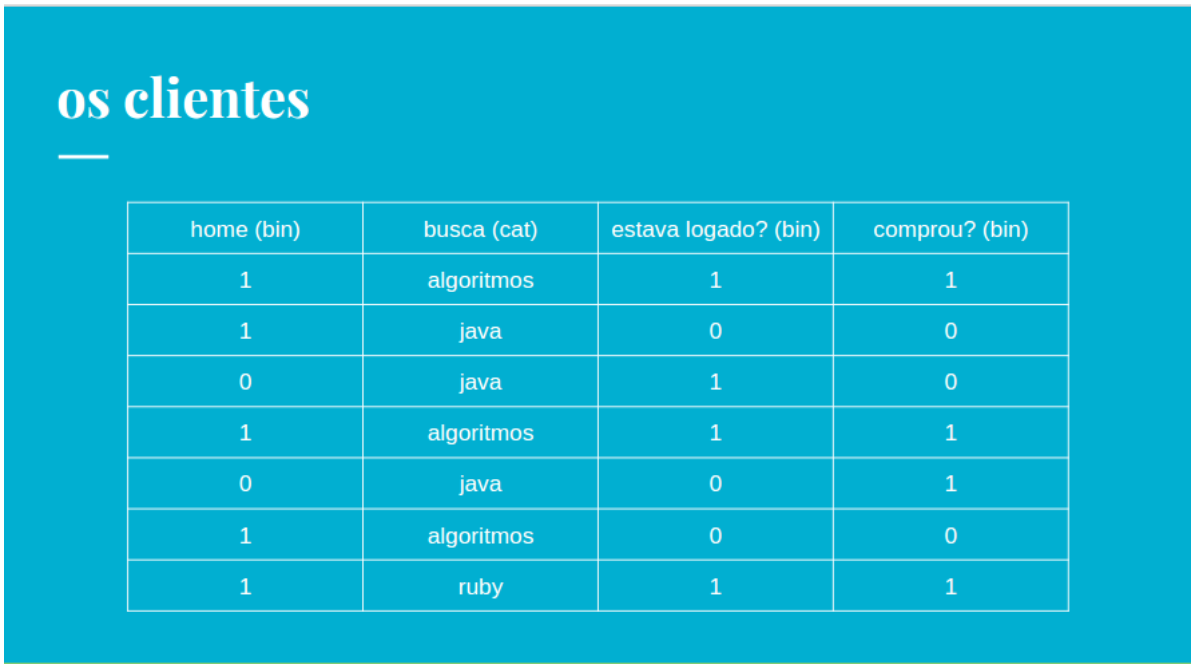
algoritmos	categoria 0
java	categoria 1
ruby	categoria 2

Agora está faltando o último cliente. Esse cliente ele pesquisou sobre "Ruby", e agora? Categoria 0 ou categoria 1? Lembre-se que as categorias 0 e 1 são referentes a "algoritmos" e "Java", ou seja, precisamos marcar apenas a "categoria 2" que refere-se a busca de "Ruby":



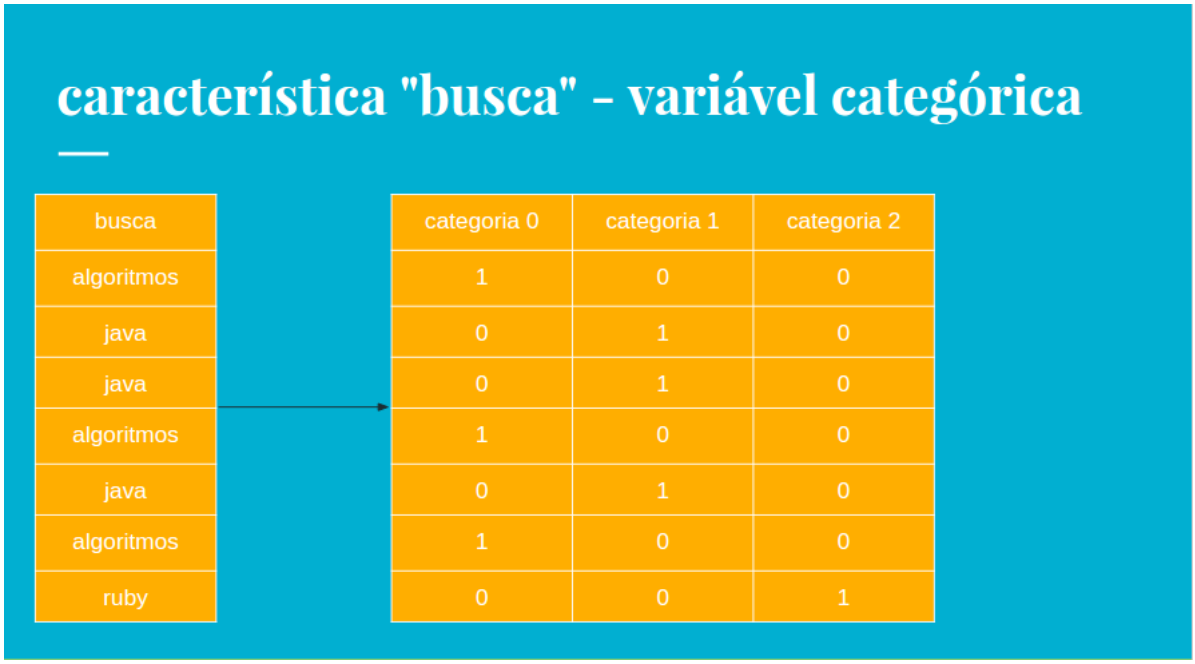
Veja como é fácil traduzir uma coluna categórica em **um conjunto de categorias**, ou seja, novas características baseadas nos valores distintos da variável categórica. Isso significa que, a coluna `busca` é equivalente às 3 colunas das categorias (0 a 2).

A princípio, a implementação parece difícil, porém, veremos que será bem mais fácil do que parece! Mas antes de mexermos com o nosso código, vamos modificar a nossa tabela original. Vejamos novamente a nossa tabela original, porém, dessa vez, vamos indicar quais são os tipos de variáveis de cada coluna:



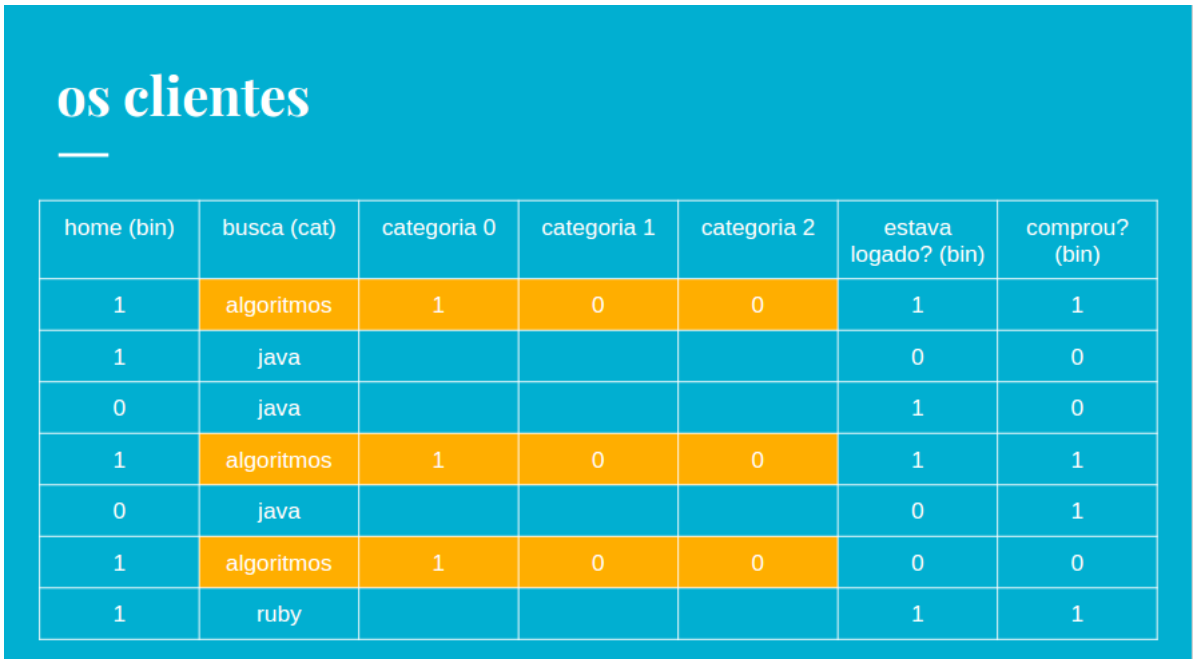
Observe que todas as categorias, exceto a `busca`, são do tipo `bin` que significa binário(0 ou 1) e a coluna `busca` é do tipo `cat` que significa uma variável categórica, ou seja, que possui categorias!

Esses dados que nós temos atualmente, já estão prontos para passarmos para um algoritmo de machine learning? Ou existe alguma dessas colunas que precisamos traduzir? Nesse caso, a variável categórica precisa ser trabalhada! O que devemos fazer então? Primeiro precisamos converter essa única coluna para as 3 possíveis categorias:



Então encaixamos essas novas colunas dentro da nossa tabela original:

Agora nós precisamos preencher cada uma dessas categorias. Lembra que para busca de algoritmos adicionávamos na categoria 0? Então fica 1, 0, 0:



E para Java? categoria 1! Então 0, 1, 0:

os clientes

home (bin)	busca (cat)	categoria 0	categoria 1	categoria 2	estava logado? (bin)	comprou? (bin)
1	algoritmos	1	0	0	1	1
1	java	0	1	0	0	0
0	java	0	1	0	1	0
1	algoritmos	1	0	0	1	1
0	java	0	1	0	0	1
1	algoritmos	1	0	0	0	0
1	ruby				1	1

E a categoria do Ruby? Categoria 2, ou seja, 0, 0, 1:

os clientes

home (bin)	busca (cat)	categoria 0	categoria 1	categoria 2	estava logado? (bin)	comprou? (bin)
1	algoritmos	1	0	0	1	1
1	java	0	1	0	0	0
0	java	0	1	0	1	0
1	algoritmos	1	0	0	1	1
0	java	0	1	0	0	1
1	algoritmos	1	0	0	0	0
1	ruby	0	0	1	1	1

Agora que conseguimos transformar todas as colunas em tipos binários, não precisamos mais da coluna `busca`, ou seja, a variável categórica. Então jogamos fora essa coluna e a nossa tabela atual mantém os seguintes dados:

os clientes: com variáveis de mentira

home (bin)	categoria 0 (bin)	categoria 1 (bin)	categoria 2 (bin)	estava logado? (bin)	comprou? (bin)
1	1	0	0	1	1
1	0	1	0	0	0
0	0	1	0	1	0
1	1	0	0	1	1
0	0	1	0	0	1
1	1	0	0	0	0
1	0	0	1	1	1

Agora que todas as nossas colunas são perguntas que esperam apenas 0 ou 1, conseguimos levar para o nosso algoritmo, pois ele sabe lidar com esses valores! Aprendendo a partir desses dados e tentando prevê-los para o futuro. Note também que a partir das 5 variáveis iniciais, iremos prever a sexta variável que é saber se o cliente comprou ou não.

Criamos essa tabela com variáveis de "mentira", ou seja, variáveis que possuem um valor real, porém elas não estavam presentes na nossa pesquisa original, pois elas estavam presentes de uma maneira diferente!

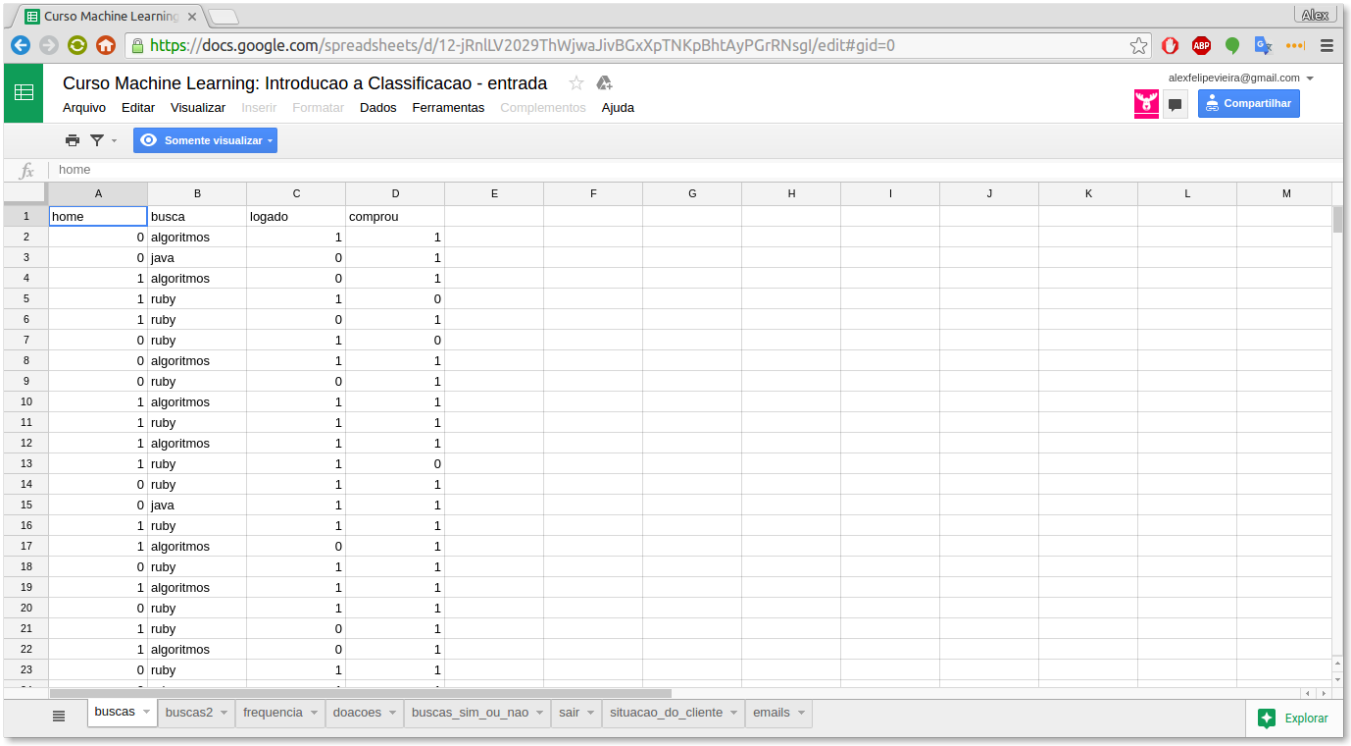
Isso significa que elas foram abordadas e questionadas de uma maneira, porém preenchemos de outra! Chamamos esse tipo de variáveis de *dummies*.

os clientes: com dummies

home (bin)	categoria 0	categoria 1	categoria 2	estava logado? (bin)	comprou? (bin)
1	1	0	0	1	1
1	0	1	0	0	0
0	0	1	0	1	0
1	1	0	0	1	1
0	0	1	0	0	1
1	1	0	0	0	0
1	0	0	1	1	1

O objeto desse tipo de variável é justamente preencher um espaço de uma outra pergunta que fizemos, ou seja, as variáveis de categorias são os nossos *dummies* do tipo binário. Agora poderemos adicionar todos esses dados em um arquivo e utilizá-los no nosso algoritmo para ele prever quem vai ou não comprar.

Temos aqui uma planilha do Google Spreadsheets (http://bit.ly/alura_analise_entrada) que contém diversos dados:



Curso Machine Learning: Introducao a Classificacao - entrada

	A	B	C	D
1	home	busca	logado	comprou
2	0	algoritmos	1	1
3	0	java	0	1
4	1	algoritmos	0	1
5	1	ruby	1	0
6	1	ruby	0	1
7	0	ruby	1	0
8	0	algoritmos	1	1
9	0	ruby	0	1
10	1	algoritmos	1	1
11	1	ruby	1	1
12	1	algoritmos	1	1
13	1	ruby	1	0
14	0	ruby	1	1
15	0	java	1	1
16	1	ruby	1	1
17	1	algoritmos	0	1
18	0	ruby	1	1
19	1	algoritmos	1	1
20	0	ruby	1	1
21	1	ruby	0	1
22	1	algoritmos	0	1
23	0	ruby	1	1

Cada coluna tem o seguinte significado:

- **home**: se o usuário acessou a home ou não (0 ou 1).
- **busca**: qual foi o curso que o usuário buscou (algoritmos ou java ou ruby).
- **logado**: se o usuário estava logado ou não (0 ou 1).
- **comprou**: se o usuário comprou (0 ou 1).

Observe que esses dados são informações que já conhecemos, ou seja, sabemos que as colunas, **home**, **logado** e **comprou** são variáveis binárias e a coluna **busca** uma variável categórica. Mas o que queremos fazer com esses dados? Trabalhar com eles no python, certo? Porém, essas informações estão em uma planilha! Precisamos, de alguma forma, transportar esses dados para um formato que o nosso algoritmo saiba trabalhar.

Todas as planilhas eletrônicas, seja o Google Spreadsheets ou Excel ou Open Office ou outras variações, vão disponibilizar alguma opção para exportarmos esses dados em algum outro formato que nos possibilite trabalhar com esses dados.

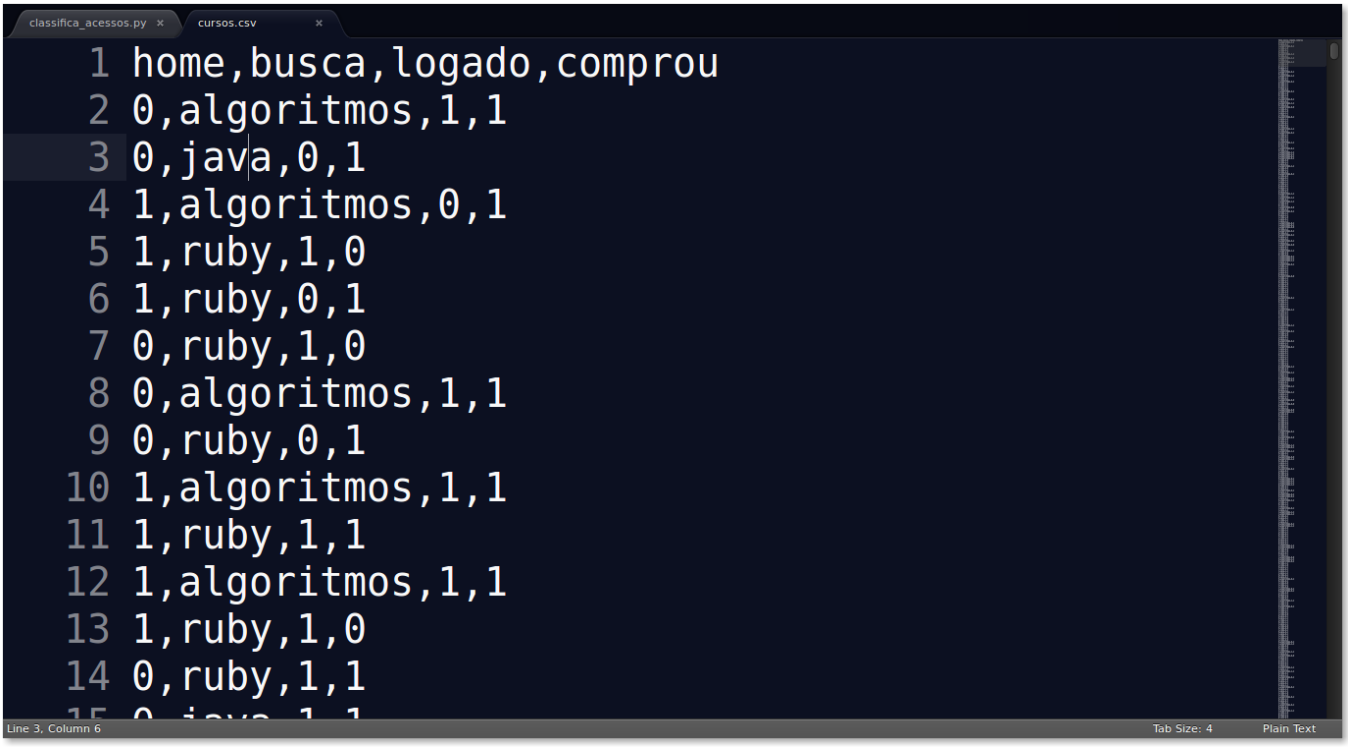
Se entrarmos no menu "Arquivo > Fazer download como" do Google Spreadsheets, veremos que ele nos fornece diversas formas para exportarmos esses arquivos, como por exemplo, no formato .xlsx para trabalharmos com o Excel, porém nós não estamos trabalhando com o Excel! Estamos trabalhando com o python.

Então qual era o tipo de arquivo que trabalhamos no python? Lembra que era o arquivo CSV? Aquele que separava cada valor por vírgula. Então, iremos importar a nossa planilha para esse formato utilizando a mesma opção "Arquivo > Fazer download como > Valores separados por vírgula (.csv, página atual)".

Observe que nessa opção ele informa que irá fazer isso para a página atual, mas o que isso significa exatamente? Se você der uma olhada na planilha:

Observará que contém outras abas, isso significa que contém para cada aba há uma página! Ou seja, quando escolhemos essa opção, exportamos um arquivo só com o conteúdo da aba (página) que estamos atualmente. Ao clicar nessa opção é realizado o download od arquivo .csv.

Renomeie o arquivo para `cursos.csv` para ficar mais fácil de ler. Por fim, coloque o arquivo dentro da pasta aonde você salvou os seus arquivos do python que criamos durante o curso. Dentro do seu editor de texto, abra o arquivo `cursos.csv` :



Veja que o arquivo foi importado sem nenhum problema. Agora já podemos fazer a leitura desse arquivo CSV que representa os acessos e buscas dos clientes aos nossos cursos.

Vamos criar um novo arquivo, porém qual nome colocaremos? Se estamos classificando cursos, que tal `classifica_acessos.py` ? Porém, já existe um arquivo com esse nome:

E agora? Qual nome daremos? Perceba que precisamos criar um padrão para que os nossos arquivos façam sentido. Podemos utilizar o padrão `classifica_nome_do_arquivo.py` , por exemplo, se o nosso arquivo python ler o arquivo `acesso.csv` ele se chamará `classifica_acessos.py` , ou seja, se está lendo `cursos.csv` , ele se chamará `classifica_cursos.py` .

Porém, é valido pensar se o nome dos nossos arquivos são coesos, pois o arquivo `cursos.csv` representa as buscas que o nosso cliente realizou, ou seja, é mais coerente chamarmos esse arquivo de `buscas.csv` , pois não estamos classificando cursos! Consequentemente, o arquivo `classifica_cursos.py` se chamará `classifica_buscas.py` .

Vejamos agora o que precisamos ler do nosso arquivo `buscas.csv` :

```
home,busca,logado,comprou
0,algoritmos,1,1
0,java,0,1
1,algoritmos,0,1
1,ruby,1,0
# restante dos dados
```

Observe que precisamos ler as 4 colunas: `home` , `busca` , `logado` e `comprou` . Já fizemos algo bem similar no arquivo `dados.py` :

```
import csv
```

```
def carregar_acessos():

    X = []
    Y = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)

    leitor.next()

    for home,planos_de_cursos,contato,comprou in leitor:

        dado = ([int(home),int(planos_de_cursos)
            ,int(contato)])
        X.append(dado)
        Y.append(int(comprou))

    return X, Y
```

Se analisarmos a função `carregar_acessos()`, podemos até pensar em reutilizá-lo, pois os procedimentos serão identicos aos dessa função, mesmo que os nomes das colunas sejam diferentes. Porém, existe um pequeno detalhe que impede que reutilizemos essa função para o arquivo `buscas.csv`, que é justamente o tipo dos dados que estão sendo processados:

```
dado = ([int(home),int(planos_de_cursos)
        ,int(contato)])
X.append(dado)
Y.append(int(comprou))
```

Note que **todos os dados** estão sendo convertidos para inteiros e os dados do nosso arquivo `buscas.csv`:

```
home,busca,logado,comprou
0,algoritmos,1,1
0,java,0,1
1,algoritmos,0,1
1,ruby,1,0
# restante dos dados
```

Além de números, possuem também palavras! Ou seja, não podemos reutilizar essa mesma função. Então iremos criar uma nova função chamada `carregar_buscas`:

```
def carregar_buscas():
```

Essa função será muito similar ao `carregar_acessos`, porém ela será utilizada para ler e tratar os dados do arquivo `buscas.csv` da forma correta:

```
def carregar_buscas():

    X = []
    Y = []

    arquivo = open('buscas.csv', 'rb')
```

```
    leitor = csv.reader(arquivo)

    leitor.next()
```

Agora nós precisamos fazer as iterações, porém, vamos mudar os nomes para as colunas do nosso arquivos `buscas.csv`:

```
def carregar_buscas():

    X = []
    Y = []

    arquivo = open('buscas.csv', 'rb')
    leitor = csv.reader(arquivo)

    leitor.next()

    for home,busca,logado,comprou in leitor:
```

Agora precisamos adicionar esses valores para o nosso `X` e `Y`. Começaremos pelo `X`:

```
# restante do código
for home,busca,logado,comprou in leitor:

dado = ([int(home), busca, int(logado)])
X.append(dado)
```

Repare que apenas a coluna `busca` será utilizada como *string*, pois ela é uma palavra! Agora precisamos adicionar o valor do `Y`:

```
# restante do código
for home,busca,logado,comprou in leitor:

dado = ([int(home), busca, int(logado)])
X.append(dado)
Y.append(int(comprou))
```

Por fim, retornamos o `X` e `Y` e a nossa função `carregar_buscas()` fica da seguinte maneira:

```
def carregar_buscas():

    X = []
    Y = []

    arquivo = open('buscas.csv', 'rb')
    leitor = csv.reader(arquivo)

    leitor.next()

    for home,busca,logado,comprou in leitor:

        dado = ([int(home), busca, int(logado)])
```

```
            X.append(dado)
            Y.append(int(comprou))

    return X, Y
```

Agora que temos a nossa função que carrega as nossas buscas, podemos ler esses dados a partir do arquivo `classifica_buscas.py`. Lembra que fizemos a mesma coisa no `classifica_acessos.py`?

```
from dados import carregar_acessos
X,Y = carregar_acessos()
```

Faremos o mesmo para o arquivo `classifica_buscas.py`, porém com a função `carregar_buscas`:

```
from dados import carregar_buscas
X,Y = carregar_buscas()
```

Vamos verificar se o `X` e `Y` estão corretos? Começaremos imprimindo o `X`:

```
from dados import carregar_buscas
X,Y = carregar_buscas()
print(X)
```

Se rodarmos o nosso `classifica_buscas.py`:

```
> python classifica_buscas.py
[[0, 'algoritmos', 1], [0, 'java', 0], [1, 'algoritmos', 0], [1, 'ruby', 1], ...]
```

Verificando no arquivo `buscas.csv`:

```
home,busca,logado,comprou
0,algoritmos,1,1
0,java,0,1
1,algoritmos,0,1
1,ruby,1,0
# Restante dos dados
```

Funcionou conforme o esperado! E o nosso `Y`? Será que está correto também? Vejamos:

```
from dados import carregar_buscas
X,Y = carregar_buscas()
print(Y)
```

Rodando o código novamente:

```
> python classifica_buscas.py
[1, 1, 1, 0, ...]
```

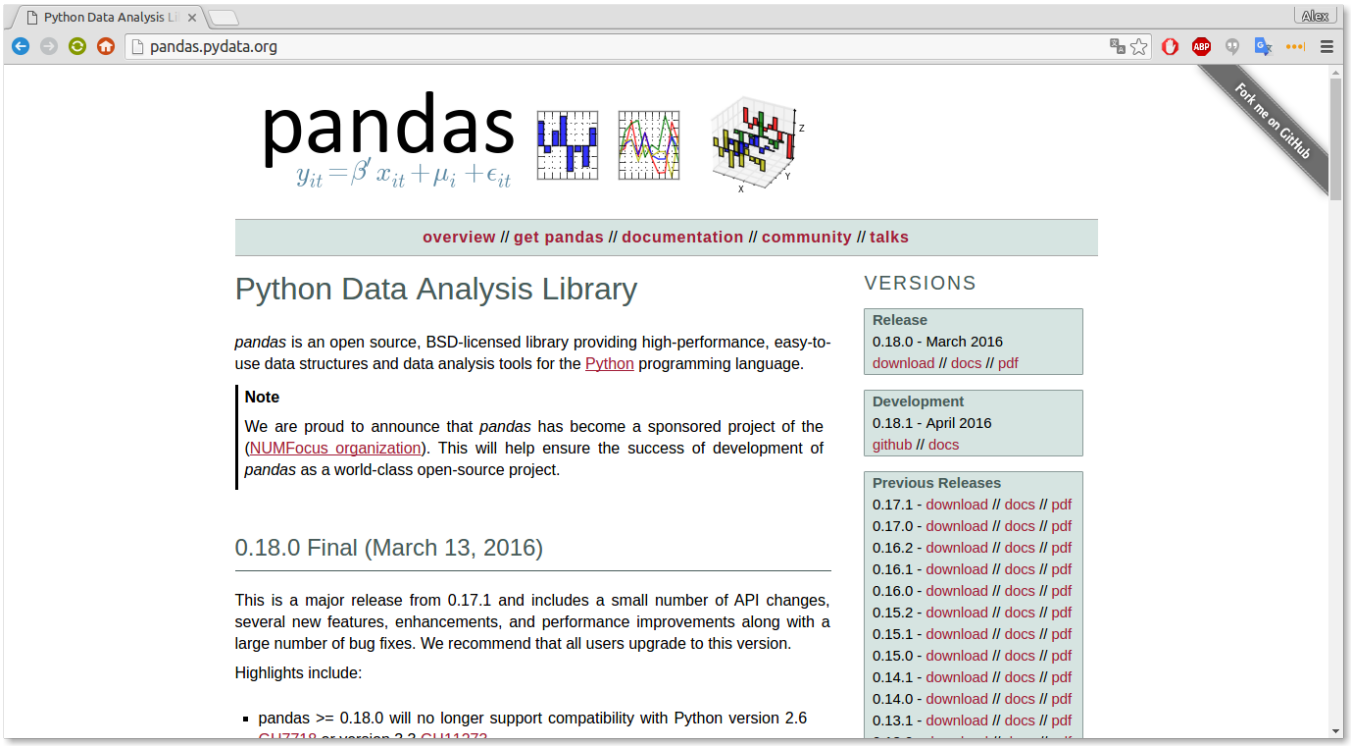
Os valores do `Y` também estão corretos! Porém, todos os nossos dados estão corretos? Ou seja, valores binários (0 ou 1). Observe que ainda estamos lidando com uma variável do tipo *string* e o nosso algoritmo não sabe lidar com esse tipo de variável!

Lembra que a coluna `busca` é uma variável categórica? Isso significa que precisamos convertê-la para 3 colunas, ou seja, as 3 categorias possíveis para essa coluna.

Mas como faremos isso no python? Para fazer essa conversão, traduzação de uma variável categórica, utilizaremos a biblioteca do python chamada pandas (*Python Data Analysis Library*).

Instalando o pandas

Acessando a página do pandas (http://pandas.pydata.org/):



Para instalar o pandas, poderíamos entrar no link "get pandas", e seguir as instruções, porém, já que temos o pip, iremos utilizá-lo para instalar o pandas por meio do comando:

```
> sudo pip install pandas
```

Caso você já tenha o pandas, atualize a biblioteca utilizando o comando:

```
> sudo pip install pandas --upgrade
```

É importante manter a versão mais atualizada, pois, dependendo da *API* que utilizaremos, pode existir variações que podem causar incompatibilidade ou simplesmente não funcionar conforme o esperado por ser depreciado. **É muito importante se atentar a isso**.

Usaremos o pandas para fazer a leitura dos nossos dados, pois ele é uma biblioteca para leitura e analise de dados. Se observamos as nossas funções que realizam a leitura dos nossos dados:

```
import csv

def carregar_acessos():

    X = []
    Y = []

    arquivo = open('acesso.csv', 'rb')
    leitor = csv.reader(arquivo)

    leitor.next()

    for home,planos_de_cursos,contato,comprou in leitor:

        dado = ([int(home),int(planos_de_cursos)
            ,int(contato)])
        X.append(dado)
        Y.append(int(comprou))

    return X, Y


import csv

def carregar_buscas():

    X = []
    Y = []

    arquivo = open('buscas.csv', 'rb')
    leitor = csv.reader(arquivo)

    leitor.next()

    for home,busca,logado,comprou in leitor:

        dado = ([int(home), busca, int(logado)])
        X.append(dado)
        Y.append(int(comprou))

    return X, Y
```

Podemos notar que, todas as vezes que queremos ler um arquivo, precisamos criar uma função nova e informar qual o tipo de valor que eu estou lendo para cada coluna. Parece repetitivo e, se nos atentarmos mais, perceberemos que isso faz parte de analise de dados. Porém, para analisarmos os dados, a primeira coisa que precisamos fazer é justamente a leitura dos dados.

Temos diversas formas para ler dados, atualmente estavamos utilizando a biblioteca padrão `csv` ou então, podemos utilizar uma biblioteca específica para analise de dados, ou seja, podemos usar o pandas para ler arquivos CSV também! Como fazemos isso? Simplesmente importando o pandas no nosso arquivo:

```
from dados import carregar_buscas
X,Y = carregar_buscas()
print(Y)
```

```
import pandas
```

Podemos também abreviar o pandas como *python data analysis* utilizando `pd` :

```
from dados import carregar_buscas
X,Y = carregar_buscas()
print(Y)

import pandas as pd
```

E como fazemos para ler um arquivo utilizando o pandas? Podemos utilizar a função `read_csv()` informando o arquivo que queremos ler por parâmetro:

```
from dados import carregar_buscas
X,Y = carregar_buscas()
print(Y)

import pandas as pd
pd.read_csv('buscas.csv')
```

E como pegamos os dados? Retornando para uma variável chamada `dados` !

```
from dados import carregar_buscas
X,Y = carregar_buscas()
print(Y)

import pandas as pd
dados = pd.read_csv('buscas.csv')
```

Vamos verificar como estão esses dados? Primeiro adicionaremos um `print` :

```
from dados import carregar_buscas
X,Y = carregar_buscas()
print(Y)

import pandas as pd
dados = pd.read_csv('buscas.csv')
print(dados)
```

E agora removeremos o nosso código anterior que fazia a leitura por meio da nossa função `carregar_buscas` :

```
import pandas as pd
dados = pd.read_csv('buscas.csv')
print(dados)
```

Rodando o nosso algoritmo:

```
python classifica_buscas.py
     home       busca  logado  comprou
0       0  algoritmos       1        1
1       0        java       0        1
2       1  algoritmos       0        1
3       1        ruby       1        0
4       1        ruby       0        1
5       0        ruby       1        0
6       0  algoritmos       1        1
...
999     0        ruby       1        0
```

Observe que o pandas conseguiu ler sem nenhum problema! Além disso, ele conseguiu identificar que a primeira linha refere-se ao cabeçalho. Veja também, que ele mostrou de uma forma bem clara o que é cada informação. E mais, ele nos informou quantas linhas de dados nós temos, nesse caso, 1000 linhas.

Na realidade, ao invés de devolver apenas um array puro e bem básico e sem uma estrutura bem elaborada, assim como fizemos, o pandas nos devolveu um objeto bem completo com todas as informações bem estruturadas. Todos esses objetos que o pandas nos devolveu, é chamado de *data frame*, ou seja, ao invés de chamarmos o retorno de `dados`, podemos identificá-lo como o nosso *data frame*, ou na abreviação, `df`:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
print(df)
```

É válido lembrar que na prática, podemos utilizar soluções mais básicas e padrões do próprio python, porém, na maioria das situações, é mais interessante utilizarmos bibliotecas específicas que já resolvem a mesma situação, porém, de uma forma mais inteligente.

Perceba que começamos com uma maneira manual justamente para entender como o processo de leitura de um arquivo de dados, no formato CSV, funciona por de trás dos panos, porém, agora que já aprendemos todo o processo para a leitura desses tipos de dados, podemos delegar toda essa tarefa para alguém que já faça tudo isso por nós e de uma maneira bem mais inteligente, nesse caso o pandas!

Lemos o arquivo utilizando o pandas, porém ainda não resolvemos o grande desafio que é solucionar a coluna `busca`. Lembra que, quando nós fazíamos a leitura dos nossos dados, separávamos todos os dados que representavam o `X` e o `Y`. Mas e agora? Como faremos isso utilizando o pandas? Será que se pedirmos para ele a primeira coluna, como se fosse um array, ele funcionaria? Vejamos:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
print(df[0])
```

Vamos testar o nosso código:

```
> python classifica_buscas.py
Traceback (most recent call last):
  File "classifica_buscas.py", line 3, in <module>
    print(df[0])
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/frame.py", line 1992, in __getitem__
```

```
    return self._getitem_column(key)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/frame.py", line 1999, in _getitem_column
    return self._get_item_cache(key)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/generic.py", line 1345, in _get_item_cach
    values = self._data.get(item)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/internals.py", line 3225, in get
    loc = self.items.get_loc(item)
  File "/usr/local/lib/python2.7/dist-packages/pandas/indexes/base.py", line 1878, in get_loc
    return self._engine.get_loc(self._maybe_cast_indexer(key))
  File "pandas/index.pyx", line 137, in pandas.index.IndexEngine.get_loc (pandas/index.c:4027)
  File "pandas/index.pyx", line 157, in pandas.index.IndexEngine.get_loc (pandas/index.c:3891)

  File "pandas/hashtable.pyx", line 675, in pandas.hashtable.PyObjectHashTable.get_item (pandas/hash
  File "pandas/hashtable.pyx", line 683, in pandas.hashtable.PyObjectHashTable.get_item (pandas/hash
KeyError: 0
```

Opa, aconteceu uma caquinha aí! Que tal pedirmos para ele utilizando o nome da coluna? Vamos tentar pedindo a coluna `home`:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
print(df['home'])
```

Rodando o nosso código:

```
> python classifica_buscas.py
0      0
1      0
2      1
3      1
4      1
5      0
6      0
...
999    0
Name: home, dtype: int64
```

Agora sim ele imprimiu todos os dados da coluna, perceba que no final, o data frame nos informa qual é o nome desses dados e o tipo dele, nesse caso estamos pegando a coluna **home** e o tipo é **int64**.

Mas, o valor do nosso `X` é mais de uma coluna, ou seja, são as colunas: `home`, `busca`, `logado`. Vamos tentar separar cada coluna por vírgula:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
print(df['home', 'busca', 'logado'])
```

Se testarmos o nosso código:

```
> python classifica_buscas.py
Traceback (most recent call last):
  File "classifica_buscas.py", line 3, in <module>
```

```
  File "classifica_buscas.py", line 3, in <module>
    print(df['home', 'busca', 'logado'])
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/frame.py", line 1992, in __getitem__
    return self._getitem_column(key)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/frame.py", line 1999, in _getitem_column
    return self._get_item_cache(key)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/generic.py", line 1345, in _get_item_cach
    values = self._data.get(item)

  File "/usr/local/lib/python2.7/dist-packages/pandas/core/internals.py", line 3225, in get
    loc = self.items.get_loc(item)
  File "/usr/local/lib/python2.7/dist-packages/pandas/indexes/base.py", line 1878, in get_loc
    return self._engine.get_loc(self._maybe_cast_indexer(key))
  File "pandas/index.pyx", line 137, in pandas.index.IndexEngine.get_loc (pandas/index.c:4027)
  File "pandas/index.pyx", line 157, in pandas.index.IndexEngine.get_loc (pandas/index.c:3891)
  File "pandas/hashtable.pyx", line 675, in pandas.hashtable.PyObjectHashTable.get_item (pandas/hash
  File "pandas/hashtable.pyx", line 683, in pandas.hashtable.PyObjectHashTable.get_item (pandas/hash
KeyError: ('home', 'busca', 'logado')
```

Eita, mais uma caca... O que será que aconteceu? Será que ele não consegue imprimir mais de uma coluna? Um detalhe importante quando queremos imprimir mais de uma coluna utilizando o pandas, é que devemos colocar todas as colunas dentro de um array:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
print(df[['home', 'busca', 'logado']])
```

Agora se tentarmos novamente rodar o código:

```
> python classifica_buscas.py
     home       busca  logado
0       0  algoritmos       1
1       0        java       0
2       1  algoritmos       0
3       1        ruby       1
4       1        ruby       0
5       0        ruby       1
6       0  algoritmos       1
...
999     0        ruby       1

[1000 rows x 3 columns]
```

As colunas são impressas, porém, ainda existe um pequeno detalhe... Analisando a coluna `busca`, vemos que os valores ainda são as variáveis categóricas! Lembra que precisamos utilizar as categorias da coluna `busca`? Ou seja, precisamos pegar os *dummies* dessa coluna. Como faremos isso? Podemos pedir para pandas devolver os *dummies* do nosso `X`:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
X = df[['home', 'busca', 'logado']]
Xdummies = pd.get_dummies(X)
print(Xdummies)
```

Testando o nosso código:

```
> python classifica_buscas.py
     home  logado  busca_algoritmos  busca_java  busca_ruby
0       0       1               1.0         0.0         0.0
1       0       0               0.0         1.0         0.0
2       1       0               1.0         0.0         0.0
3       1       1               0.0         0.0         1.0
4       1       0               0.0         0.0         1.0
5       0       1               0.0         0.0         1.0
6       0       1               1.0         0.0         0.0
...

999     0       1               0.0         0.0         1.0

[1000 rows x 5 columns]
```

Conseguimos os *dummies* do `X`, mas perceba que os valores vieram como ponto flutuante, não queremos números flutuantes e sim inteiros! Para convertemos os valores do data frame para inteiro, basta utilizar o método `astype()` passando `int` por parâmetro:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
X = df[['home', 'busca', 'logado']]
Xdummies = pd.get_dummies(X).astype(int)
print(Xdummies)
```

Rodando novamente o nosso código:

```
python classifica_buscas.py
     home  logado  busca_algoritmos  busca_java  busca_ruby
0       0       1                 1           0           0
1       0       0                 0           1           0
2       1       0                 1           0           0
3       1       1                 0           0           1
4       1       0                 0           0           1
5       0       1                 0           0           1
6       0       1                 1           0           0
...
999     0       1                 0           0           1

[1000 rows x 5 columns]
```

Mas e o `Y`? O que fazemos com ele? Precisamos pegar os *dummies* do `Y` sendo que a coluna `comprou` é uma coluna com valores binários? A resposta é não, pois o próprio `Y` já possui os valores que precisamos para o nosso algoritmo, ou seja, ele por si só já é um *dummie*:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
X = df[['home', 'busca', 'logado']]
Y = df['comprou']
```

```
Xdummies = pd.get_dummies(X).astype(int)
Ydummies = Y

print(Xdummies)
```

Vejamos o resultado:

```
> python classifica_buscas.py
0      1
1      1
2      1
3      0
4      1
5      0
6      1
...
999    0
Name: comprou, dtype: int64
```

Ele é impresso conforme o esperado! Porém, vamos dar uma olhada, novamente, como o algoritmo naive bayes `MultinomialNB` funciona:

```
treino_dados = X[:90]
treino_marcacoes = Y[:90]

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
modelo.fit(treino_dados, treino_marcacoes)

# restante do código
```

Observe que o algoritmo `MultinomialNB` recebe 2 arrays do python, ou seja, ele não recebe data frames e sim lista de dados! Então teremos que, de alguma forma, transforma os nossos data frames e arrays de dados. Se analisarmos melhor as nossas variáveis que representam os data frames:

```
import pandas as pd
df = pd.read_csv('buscas.csv')

X = df[['home', 'busca', 'logado']]
Y = df['comprou']
Xdummies_df = pd.get_dummies(X).astype(int)
Ydummies_df = Y
```

Note que `X`, `Xdummies` e `Y`, `Ydummies` são nomes ruins, pois não deixa claro se estamos trabalhando com data frames ou arrays! É sempre importante deixarmos bem claro o significado das nossas variáveis, por isso, alteraremos o nomes de ambas para indicar que são data frames:

```
import pandas as pd
df = pd.read_csv('buscas.csv')

X_df = df[['home', 'busca', 'logado']]
```

```
Y_df = df['comprou']
Xdummies_df = pd.get_dummies(X).astype(int)
Ydummies_df = Y
```

Agora nós precisamos pegar os 2 data frames e transformá-los em arrays. Para isso, usaremos o `values` do data frame, ou seja, pegaremos os valores dos nossos data frames:

```
Xdummies_df.values
Ydummies_df.values
```

Quem são esses valores? Eles são os valores reais do `X` e do `Y`, ou seja, podemos atribuir esses valores para variáveis `X` e `Y`:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
X_df = df[['home', 'busca', 'logado']]
Y_df = df['comprou']

Xdummies_df = pd.get_dummies(X_df).astype(int)
Ydummies_df = Y_df

X = Xdummies_df.values
Y = Ydummies_df.values
```

Vamos imprimir os valores de `X` e `Y` e verificar o resultado:

Primeiro imprimeros o `X`:

```
# restante do código
print(X)
```

Resultado:

```
> python classifica_buscas.py
[[0 1 1 0 0]
 [0 0 0 1 0]
 [1 0 1 0 0]
 ...,
 [0 1 0 1 0]
 [1 0 1 0 0]
 [0 1 0 0 1]]
```

Agora o `Y`:

```
# restante do código
print(Y)
```

Resultado:

```
python classifica_buscas.py
[1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 0 0 1 0 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1
 1 0 1 1 1 1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1
 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0
 1 0 0 1 0 1 1 1 1 0 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0
 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1
 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0
 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0
 0 1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1
 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1
 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 0 1 1 1 1 1 1 0 1 0 0 1 1 1
 0]
```

Conseguimos pegar os valores do nosso `X` e `Y`, ou seja, podemos utilizá-los para treinar e testar o nosso algoritmo. Se verificarmos o código do `classifica_acessos.py`:

```
from dados import carregar_acessos
X,Y = carregar_acessos()

treino_dados = X[:90]
treino_marcacoes = Y[:90]

teste_dados = X[-9:]
teste_marcacoes = Y[-9:]

# restante do código
```

Inicialmente separamos os dados de treino e os dados de teste, nesse exemplo utilizamos 90 primeiros dados para treino e 9 últimos para teste. Faremos o mesmo com os novos dados, porém, vamos verificar a quantidade de dados que temos, por exemplo, no nosso `Y`:

```
>>> import pandas as pd
>>> df = pd.read_csv('buscas.csv')
>>> X_df = df[['home', 'busca', 'logado']]
>>> Y_df = df['comprou']
>>>
```

```
>>> Xdummies_df = pd.get_dummies(X_df).astype(int)
>>> Ydummies_df = Y_df
>>>
>>> X = Xdummies_df.values
>>> Y = Ydummies_df.values
>>>
>>> len(Y)
1000
```

Espera um pouco! Não temos apenas 99 linhas e sim 1000 linhas... Isso significa que não queremos treinar com as 90 primeiras linhas assim como fizemos anteriormente, e sim, treinar com os primeiros 90% dos nossos dados e testar com o restante. Então, o que devemos fazer é criar uma variável que representará a quantidade de dados para treino:

```
# restante do código
tamanho_de_treino = 0.9 * len(Y)
```

Vamos verificar o valor dessa variável:

```
>>> import pandas as pd
>>> df = pd.read_csv('buscas.csv')
>>> X_df = df[['home', 'busca', 'logado']]
>>> Y_df = df['comprou']
>>>
>>> Xdummies_df = pd.get_dummies(X_df).astype(int)
>>> Ydummies_df = Y_df
>>>
>>> X = Xdummies_df.values
>>> Y = Ydummies_df.values
>>>
>>> tamanho_de_treino = 0.9 * len(Y)
>>>
>>> tamanho_de_treino
900.0
>>>
```

Conseguimos os nossos primeiros 90% dados, ou seja, os primeiros 900. A partir desse parâmetro, vamos pegar agora os primeiros 90% de X que são os nossos dados:

```
# restante do código
tamanho_de_treino = 0.9 * len(Y)

treino_dados = X[:tamanho_de_treino]
```

Agora precisamos pegar as marcações de treino, ou seja, os primeiros 90% das linhas:

```
# restante do código
tamanho_de_treino = 0.9 * len(Y)

treino_dados = X[:tamanho_de_treino]
treino_marcacoes = Y[:tamanho_de_treino]
```

Pegamos tanto os dados, quanto as marcações de treino. Precisamos apenas pegar os dados e marcações de teste, como pegamos os dados de teste? Olhando novamente o código do `classifica_acessos.py`:

```
# restante do código

teste_dados = X[-9:]
teste_marcacoes = Y[-9:]
```

Percebemos que são os últimos 10% dos dados, então precisamos definir o tamanho de teste como 10%. Poderíamos simplesmente fazer

```
tamanho_de_teste = 0.1 * len(Y)
```

Porém, já que obtivemos o tamanho de treino, podemos simplesmente pegar o tamanho do `Y` e subtrair pela variável `tamanho_de_treino`, pois 100% - 90% (`tamanho_de_treino`) resultará nos 10% que precisamos! Além disso, se um dia alterarmos o tamanho de treino para, por exemplo, 80% ele calculará automaticamente o tamanho de teste, ou seja, modificaremos em apenas um ponto do código!

```
# restante do código

tamanho_de_treino = 0.9 * len(Y)
tamanho_de_teste = len(Y) - tamanho_de_treino

treino_dados = X[:tamanho_de_treino]
treino_marcacoes = Y[:tamanho_de_treino]
```

Para melhorar mais ainda o nosso código, podemos extrair o valor `0.9` para uma variável indicando que refere-se a nossa porcentagem:

```
porcentagem_treino = 0.9
```

Dessa forma nós podemos reutilizar essa porcentagem em qualquer lugar do código caso seja necessário. Vamos verificar se está funcionando?

```
>>> import pandas as pd
>>> df = pd.read_csv('buscas.csv')
>>> X_df = df[['home', 'busca', 'logado']]
>>> Y_df = df['comprou']
>>>
>>> Xdummies_df = pd.get_dummies(X_df).astype(int)
>>> Ydummies_df = Y_df
>>>
>>> X = Xdummies_df.values
>>> Y = Ydummies_df.values
>>>
>>> porcentagem_treino = 0.9
>>>
>>> tamanho_de_treino = porcentagem_treino * len(Y)
>>> tamanho_de_teste = len(Y) - tamanho_de_treino
```

```
>>>
>>> tamanho_de_teste
100.0
```

Ele retorna os 10% que precisamos! Podemos retornar os nossos dados de teste. Como fazemos isso? Lembra que precisamos dos últimos 10% dos dados? Então faremos o mesmo, porém utilizando a variável `tamanho_de_teste` :

```
# restante do código

porcentagem_treino = 0.9

tamanho_de_treino = porcentagem_treino * len(Y)
tamanho_de_teste = len(Y) - tamanho_de_treino

treino_dados = X[:tamanho_de_treino]
treino_marcacoes = Y[:tamanho_de_treino]

teste_dados = X[-tamanho_de_teste:]
teste_marcacoes = Y[-tamanho_de_teste:]
```

Vamos verificar os nossos dados e marcações. Primeiro começaremos pelos dados e marcações de treino:

```
>>> import pandas as pd
>>> df = pd.read_csv('buscas.csv')
>>> X_df = df[['home', 'busca', 'logado']]
>>> Y_df = df['comprou']
>>>
>>> Xdummies_df = pd.get_dummies(X_df).astype(int)
>>> Ydummies_df = Y_df
>>>
>>> X = Xdummies_df.values
>>> Y = Ydummies_df.values
>>>
>>> porcentagem_treino = 0.9
>>>
>>> tamanho_de_treino = porcentagem_treino * len(Y)
>>> tamanho_de_teste = len(Y) - tamanho_de_treino
>>>
>>> treino_dados = X[:tamanho_de_treino]
>>> treino_marcacoes = Y[:tamanho_de_treino]
>>>
>>> teste_dados = X[-tamanho_de_teste:]
>>> teste_marcacoes = Y[-tamanho_de_teste:]
>>>
>>> treino_dados
array([[0, 1, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [1, 0, 1, 0, 0],
       ...,
       [0, 0, 0, 0, 1],
       [1, 1, 1, 0, 0],
       [1, 1, 0, 1, 0]])
>>> treino_marcacoes
array([1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
       1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
      0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1])
>>>
```

Agora os dados de teste:

```
>>> teste_dados
array([[0, 0, 0, 1, 0],
       [1, 0, 0, 1, 0],
       [1, 1, 1, 0, 0],
       [1, 0, 0, 1, 0],
       [1, 1, 1, 0, 0],
       [1, 1, 1, 0, 0],
       [0, 1, 0, 0, 1],
       [1, 1, 0, 1, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 0, 1, 0],
       [1, 1, 1, 0, 0],
       [1, 0, 1, 0, 0],
```

```
[0, 1, 1, 0, 0],
[1, 0, 1, 0, 0],
[0, 1, 1, 0, 0],
[0, 0, 0, 0, 1],
[1, 1, 1, 0, 0],
[1, 0, 0, 0, 1],
[0, 0, 0, 1, 0],
[1, 0, 0, 0, 1],
[0, 1, 0, 1, 0],
[0, 1, 0, 0, 1],
[1, 1, 1, 0, 0],
[0, 1, 0, 0, 1],
[1, 1, 1, 0, 0],
[1, 1, 0, 0, 1],
[0, 0, 1, 0, 0],
[0, 1, 0, 1, 0],
[1, 0, 0, 1, 0],
[0, 1, 0, 1, 0],
[1, 0, 0, 0, 1],
[1, 0, 0, 0, 1],
[0, 0, 1, 0, 0],
[1, 0, 1, 0, 0],
[1, 0, 0, 0, 1],
[1, 0, 0, 0, 1],
[1, 0, 0, 0, 1],
[1, 1, 0, 1, 0],
[1, 1, 1, 0, 0],
[1, 0, 1, 0, 0],
[0, 1, 0, 1, 0],
[0, 0, 0, 1, 0],
[1, 1, 1, 0, 0],
[1, 1, 0, 0, 1],
[0, 1, 0, 1, 0],
[1, 1, 0, 1, 0],
[1, 0, 1, 0, 0],
[0, 1, 1, 0, 0],
[0, 1, 1, 0, 0],
[0, 0, 0, 1, 0],
[0, 1, 1, 0, 0],
[1, 0, 0, 1, 0],
[0, 1, 0, 1, 0],
[1, 1, 0, 0, 1],
[0, 0, 0, 0, 1],
[0, 0, 0, 1, 0],
[1, 0, 0, 1, 0],
[0, 0, 0, 1, 0],
[1, 0, 1, 0, 0],
[0, 1, 1, 0, 0],
[0, 0, 0, 0, 1],
[1, 0, 0, 0, 1],
[0, 0, 1, 0, 0],
[0, 0, 0, 1, 0],
[0, 1, 1, 0, 0],
[0, 0, 1, 0, 0],
[1, 1, 0, 0, 1],
[1, 1, 1, 0, 0],
[0, 1, 0, 0, 1],
[0, 1, 0, 1, 0],
```

```
        [1, 1, 0, 0, 1],
        [1, 1, 1, 0, 0],
        [0, 1, 0, 1, 0],
        [1, 1, 0, 0, 1],
        [1, 0, 1, 0, 0],
        [0, 1, 0, 1, 0],
        [1, 0, 0, 0, 1],
        [1, 0, 0, 0, 1],
        [0, 0, 0, 0, 1],
        [1, 0, 0, 1, 0],
        [1, 0, 0, 0, 1],
        [0, 0, 1, 0, 0],
        [1, 1, 0, 0, 1],
        [1, 1, 0, 0, 1],
        [0, 0, 0, 1, 0],
        [1, 1, 0, 0, 1],
        [0, 0, 1, 0, 0],
        [0, 1, 0, 1, 0],
        [1, 0, 0, 1, 0],
        [0, 0, 1, 0, 0],
        [0, 0, 0, 1, 0],
        [1, 0, 1, 0, 0],
        [1, 0, 1, 0, 0],
        [1, 0, 0, 1, 0],
        [0, 1, 0, 0, 1],
        [0, 0, 0, 0, 1],
        [0, 0, 0, 0, 1],
        [0, 1, 0, 1, 0],
        [1, 0, 1, 0, 0],
        [0, 1, 0, 0, 1]])
>>>
>>> teste_marcacoes
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 1, 1, 0])
>>>
```

Conseguimos pegar tanto os dados de treino quantos os de teste, qual que é o próximo passo? De acordo com os nossos dados de treino e de teste, queremos criar o nosso modelo! Lembra como criamos o modelo? Se verificarmos o código `classifica_acessos.py`:

```
from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
modelo.fit(treino_dados, treino_marcacoes)
```

Observe que poderemos reutilizar esse código, ou seja, faremos um *copy and paste*. E o resultado? Será que poderemos fazer o mesmo? Vejamos no `classifica_acessos.py`:

```
resultado = modelo.predict(teste_dados)
```

Também não mudou nada, ou seja, podemos reutilizá-lo também! Por fim, vamos verificar como estão sendo feitos os calculos para a taxa de acerto:

```
diferencas = resultado - teste_marcacoes

acertos = [d for d in diferencas if d == 0]
total_de_acertos = len(acertos)
total_de_elementos = len(teste_dados)

taxa_de_acerto = 100.0 * total_de_acertos / total_de_elementos

print(taxa_de_acerto)
print(total_de_elementos)
```

Observe que **todo o código** utilizado no arquivo `classifica_acessos.py` para treinar e testar o modelo, calcular o resultado e a taxa de acerto não mudam! Ou seja, poderemos copiar todo o código e adicioná-lo no nosso arquivo `classifica_buscas.py`. O nosso código final fica assim:

```
import pandas as pd
df = pd.read_csv('buscas.csv')
X_df = df[['home', 'busca', 'logado']]
Y_df = df['comprou']

Xdummies_df = pd.get_dummies(X_df).astype(int)
Ydummies_df = Y_df

X = Xdummies_df.values
Y = Ydummies_df.values

porcentagem_treino = 0.9

tamanho_de_treino = porcentagem_treino * len(Y)
tamanho_de_teste = len(Y) - tamanho_de_treino

treino_dados = X[:tamanho_de_treino]
treino_marcacoes = Y[:tamanho_de_treino]

teste_dados = X[-tamanho_de_teste:]
teste_marcacoes = Y[-tamanho_de_teste:]

from sklearn.naive_bayes import MultinomialNB
modelo = MultinomialNB()
modelo.fit(treino_dados, treino_marcacoes)

resultado = modelo.predict(teste_dados)

diferencas = resultado - teste_marcacoes

acertos = [d for d in diferencas if d == 0]
total_de_acertos = len(acertos)
total_de_elementos = len(teste_dados)

taxa_de_acerto = 100.0 * total_de_acertos / total_de_elementos
```

```
print(taxa_de_acerto)
print(total_de_elementos)
```

Vamos testar e verificar se está funcionando:

```
> python classifica_buscas.py
82.0
100
```

O que o nosso algoritmo fez? Dessa vez ele realizou um treino com 900 elementos e treinou com 100 elementos que ele nunca viu. Dentre esses 100 elementos ele conseguiu prever 82%!

É válido ressaltar que, de acordo com esses dados, que são de um sistema real, o nosso algoritmo conseguiu acertar 82% das vezes utilizando apenas 3 características, variáveis que descrevem o que o usuário fez, poderíamos também adicionar mais variáveis, porém, apenas com essa quantidade de informação o nosso algoritmo foi capaz de acertar 82%. Dessa vez os dados são reais, então vem aquela mesma pergunta, 82% é bom ou ruim?

Note que essa pergunta é bem difícil de responder, pois não temos nenhum método de classificação **diferente** que possa comparar com o nosso código. Se o resultado fosse 100%, com certeza não restaria dúvida que o algoritmo é perfeito, porém, o que precisamos saber por agora é quão longe o nosso algoritmo consegue chegar com dados reais.

Temos que levar em consideração que em cada área, haverão dados diferentes, ou seja, os resultados, provavelmente, serão diferentes, mas, o que precisamos focar é justamente em saber o quão bom é o nosso algoritmo para os determinados números que estamos trabalhando atualmente.

Resumindo

Nesse capítulo nós realizamos um teste para verificar quais clientes iriam ou não comprar no meu site. Vimos que o nosso algoritmo acertou 82% utilizando dados reais, a partir desses dados podemos entrar em contato com os clientes que não irão comprar para tentar entender a necessidade deles ou propor alguma coisa que seja de seu interesse. Então podemos concluir que 82% das vezes acertamos se o cliente iria ou não comprar.

Além disso, nesse mesmo capítulo, fizemos alguns ajustes no nosso algoritmo, dessa vez utilizamos uma biblioteca nova, o pandas, para realizarmos a leitura dos nossos arquivos CSV que continham os nossos dados de uma forma bem simplificada e, além de realizar a leitura, ele nos devolveu um conjunto de informações bem mais completo e inteligente que o array bobo que estávamos fazendo na unha conhecido como *data frame*, que nos permite rodar algoritmos de *data analysis* como por exemplo, machine learning.

Esses data frames são um conjunto de dados que nos fornece diversos recursos, como por exemplo, pegar uma coluna ou diversas colunas, por meio de um(ns) cabeçalho(s), facilitando, e muito, a nossa vida.

Um outro ponto importante visto nesse capítulo, é que mudamos o método de leitura dos nossos dados, porém, o método de criação e treino do modelo utilizando o algoritmo `MultinomialNB` e o cálculo da taxa de acerto, foram os mesmos que utilizamos no arquivo `classifica_acessos.py` , ou seja, reutilizamos o mesmo código! Por fim, nós chegamos a mais uma questão muito importante que é verificar se o resultado do nosso algoritmo, para esse cenário, ou seja, esses dados, foi bom ou ruim. Veremos com mais detalhes essa questão no próximo capítulo.