Capítulo 5

COMBINANDO DATAFRAMES

A etapa de **seleção de dados** é quase sempre necessária em qualquer processo de ciência de dados. É nela que todos os dados relevantes para a resolução do problema são coletados e **combinados** em um único DataFrame. Trata-se de uma tarefa desafiadora, uma vez que os dados relevantes costumam estar estruturados em formatos heterogêneos: tabelas de bancos de dados, arquivos texto (CSV, JSON etc.), planilhas Excel, páginas da internet, arquivos de log, entre outras possibilidades.

Felizmente a pandas simplifica muito o processo de coleta de dados, pois permite a importação de virtualmente qualquer tipo de arquivo para DataFrames (como foi visto no capítulo 3). A biblioteca vai ainda mais além, oferecendo ferramentas para possibilitar a combinação de DataFrames. Este capítulo tem como objetivo principal apresentar as três diferentes abordagens que podem ser empregadas para combinar DataFrames:

- Concatenação
- Operações de conjunto
- Junção

5.1 CONCATENAÇÃO

Concatenar dois ou mais DataFrames pandas consiste na operação de empilhar um "no topo" do outro, gerando um DataFrame único como resultado. Vamos apresentar essa operação de forma prática. Suponha uma empresa que possua três lojas, digamos A, B e C. Considere que em todas as segundas-feiras, cada uma das lojas deve enviar para a matriz um arquivo padronizado contendo o consolidado das vendas do final de semana. Por exemplo, considere que os arquivos seguintes foram os últimos enviados por cada loja:

Arquivo da Loja A:

```
A sex 7500
A sab 9500
A dom 8200
```

Arquivo da Loja B:

```
B sex 5100
B sab 8250
B dom 9900
```

Arquivo da Loja C:

```
C sab 7500
C dom 11800
```

Veja que todos os arquivos possuem os mesmos três atributos, especificados na mesma ordem: identificação da loja, dia da semana e valor consolidado das vendas. Observe ainda que o arquivo da loja C possui apenas duas linhas (imagine que a loja não abriu na sexta-feira). No programa a seguir, os dados apresentados serão inicialmente estruturados em três DataFrames distintos que, em seguida, serão concatenados para um único

DataFrame maior através do método `concat()`:

```
#P39: Concatenação de DataFrames
import pandas as pd

#(1)-Cria os DataFrames com as vendas de cada loja
lojaA = pd.DataFrame({"loja":["A", "A", "A"],
                      "dia": ["sex", "sab", "dom"],
                      "valor": [7500, 9500, 8200]}
                      )

lojaB = pd.DataFrame({"loja":["B", "B", "B"],
                      "dia": ["sex", "sab", "dom"],
                      "valor": [5100, 8250, 9900]}
                      )

lojaC = pd.DataFrame({"loja":["C", "C"],
                      "dia": ["sab", "dom"],
                      "valor": [7500, 11800]}
                      )

#(2)-Concatena tudo em um único DataFrame
lojasABC = pd.concat([lojaA,lojaB,lojaC], ignore_index=True)

print(lojasABC)
```

Saída:

```
  loja  dia  valor
0    A  sex   7500
1    A  sab   9500
2    A  dom   8200
3    B  sex   5100
4    B  sab   8250
5    B  dom   9900
6    C  sab   7500
7    C  dom  11800
```

Neste exemplo, os DataFrames `lojaA` (com 3 linhas), `lojaB` (com 3 linhas) e `lojaC` (com 2 linhas) foram concatenados para um único DataFrame `lojasABC` contendo 3 + 3 + 2 = 8 linhas. A utilização do método `concat()` para efetuar esta operação é

bastante simples: basta especificar os arquivos a serem concatenados em uma lista (`[lojaA,lojaB,lojaC]`) para que isso seja feito na ordem indicada. Em nosso exemplo, também fizemos uso do parâmetro `ignore_index=True` , que, apesar de opcional, é muito útil. Ele foi utilizado para forçar a geração de novos índices, variando de 0 a 7, no DataFrame `lojasABC` . Se o parâmetro não for especificado, a pandas trabalhará apenas com os índices dos DataFrames originais (0, 1 e 2), associando mais de uma linha a cada um destes três índices.

A concatenação é uma operação que faz mais sentido quando os DataFrames de origem possuem a mesma definição ou **esquema**, isto é, contêm os mesmos atributos, com os mesmos nomes e tipos, especificados na mesma ordem. No entanto, nada impede que DataFrames não inteiramente compatíveis sejam concatenados. No exemplo a seguir, mostramos o que acontece quando se realiza a concatenação de dois DataFrames que não possuem nenhum atributo em comum:

```
#P40: Concatenação de DataFrames Incompatíveis
import pandas as pd

#(1)-Cria dois DataFrames com definição diferente
d1 = pd.DataFrame({"carro":["Hyundai", "Renault", "Fiat"]})
d2 = pd.DataFrame({"animal":["Capivara", "Bem-te-vi"]})

#(2)-Concatena os DataFrames
d3 = pd.concat([d1,d2], ignore_index=True, sort=False)

print(d3)
```

A seguir, o resultado da concatenação:

```
     carro     animal
0  Hyundai        NaN
1  Renault        NaN
2     Fiat        NaN
```

```
3      NaN   Capivara
4      NaN  Bem-te-vi
```

Neste exemplo, primeiro criamos dois DataFrames, `d1` com uma única coluna chamada `carro` e `d2` com uma única coluna chamada `animal`. Como resultado da concatenação, o DataFrame produzido fica definido com essas mesmas duas colunas, fazendo uso do valor `NaN` para permitir o alinhamento dos dados. O parâmetro `sort=False` foi utilizado para fazer com que, no DataFrame resultante, a pandas não tente reordenar os atributos (isto é, para definir o primeiro atributo de `d3` como `carro`, vindo `d1`, e o segundo como `animal`, vindo de `d2`).

5.2 OPERAÇÕES DE CONJUNTO

União, interseção e diferença

Em algumas situações práticas, pode ser interessante tratar um DataFrame como um **conjunto matemático**, ou seja, como uma coleção de elementos do mesmo tipo. Neste caso, cada linha do DataFrame passa a ser considerada um elemento do conjunto. Mas por que isso é útil? A explicação é simples. Como sabemos, problemas de ciência de dados tipicamente exigem a análise de diversas bases de dados, muitas vezes oriundas de diferentes fontes. Em alguma etapa da análise, pode ser necessário comparar o conteúdo de dois ou mais DataFrames para determinar se eles possuem elementos (linhas) em comum.

Por exemplo, suponha que você recebeu dois enormes arquivos contendo e-mails de alunos de cursos EAD. Um dos arquivos contém os e-mails dos alunos que realizaram o curso de SQL e o outro dos alunos do curso de Python. Imagine que você precise de

respostas para as seguintes questões:

1. Quais são os alunos distintos, considerando ambas as listas?
2. Quais são os alunos que fizeram ambos os cursos?
3. Quais os alunos que realizaram o curso de SQL, mas não o de Python (e vice-versa)?

Neste caso, as respostas podem ser obtidas de maneira trivial, se você importar os arquivos para dois diferentes DataFrames e, então, considerar que estes DataFrames são conjuntos matemáticos. Bastará então aplicar as operações básicas de união, interseção e diferença para obter as respostas para as questões (1), (2) e (3), respectivamente. Na pandas, estas operações de conjunto são disponibilizadas através dos métodos relacionados na tabela seguinte:

Operação	Método(s)
união	`concat() + drop_duplicates()`
interseção	`merge()`
diferença	`isin()` + indexação booleana

No próximo programa, a utilização destes métodos é demonstrada. Uma explicação detalhada é apresentada logo após a listagem do código.

```
#P41: Operações de conjunto
import pandas as pd

#(1)-Cria dois DataFrames com e-mails
df_sql = pd.DataFrame({"email":["rakesh@xyz.com",
                                "ecg@acmecorpus.com"]})

df_python = pd.DataFrame({"email":["ana@xyz.com",
                                   "jonas@acmecorpus.com",
```

```
                                    "rakesh@xyz.com"]})

#(2)-Efetua as operações de conjunto

#2.1 União (relação de alunos distintos)
alunos = pd.concat([df_sql, df_python], ignore_index=True)
alunos = alunos.drop_duplicates()

#2.2 Interseção (quem fez ambos os cursos)
sql_e_python = df_sql.merge(df_python)

#2.3 Diferença (quem fez só SQL e quem fez só Python)
so_sql = df_sql[df_sql.email.isin(df_python.email)==False]
so_python = df_python[df_python.email.isin(df_sql.email)==False]

#(3)-Imprime os resultados
print('------------------------------')
print('Alunos Distintos:')
print(alunos)
print('------------------------------')
print('Alunos cursaram SQL e Python:')
print(sql_e_python)
print('------------------------------')
print('Alunos cursaram apenas SQL:')
print(so_sql)
print('------------------------------')
print('Alunos cursaram apenas Python:')
print(so_python)
```

Veja o resultado:

```
Alunos Distintos:
                  email
0        rakesh@xyz.com
1    ecg@acmecorpus.com
2           ana@xyz.com
3  jonas@acmecorpus.com
------------------------------
Alunos cursaram SQL e Python:
           email
0  rakesh@xyz.com
------------------------------
Alunos cursaram apenas SQL:
               email
```

```
1  ecg@acmecorpus.com
------------------------------
Alunos cursaram apenas Python:
                  email
0           ana@xyz.com
1  jonas@acmecorpus.com
```

Nesse programa, inicialmente criamos dois DataFrames, `df_sql` (e-mails dos alunos do curso de SQL) e `df_python` (e-mails dos alunos do curso de Python). Após terem sido criados, estes DataFrames são submetidos às operações de conjunto na segunda parte do programa.

Para realizar a operação de união (parte 2.1 do programa), utilizamos nosso conhecido método `concat()` seguido do método `drop_duplicates()`. Como sabemos, o método `concat()` serve para concatenar DataFrames. Sendo assim, o comando `alunos = pd.concat([df_sql, df_python], ignore_index=True)` simplesmente gera um DataFrame com 2 + 3 = 5 linhas:

```
0        rakesh@xyz.com
1    ecg@acmecorpus.com
2           ana@xyz.com
3  jonas@acmecorpus.com
4        rakesh@xyz.com
```

Veja que existem duas linhas referentes ao e-mail "rakesh@xyz.com", pois esse é o único aluno que realizou os dois cursos (seu e-mail está contido nos dois DataFrames). Para manter apenas uma ocorrência deste e-mail, basta aplicar o método `drop_duplicates()` - que, como o seu próprio nome sugere, serve para remover linhas duplicadas. Desta forma, o comando `alunos = alunos.drop_duplicates()` produz como resultado o conjunto de todos os e-mails distintos:

```
0        rakesh@xyz.com
1    ecg@acmecorpus.com
2           ana@xyz.com
3  jonas@acmecorpus.com
```

Para realizar a operação de interseção (parte 2.2 do programa), utilizamos o método `merge()`. Trata-se de um poderoso método da pandas que é empregado principalmente para a implementar a operação de junção de DataFrames, como veremos em detalhes ainda neste capítulo. Por ora, basta saber que, quando este método é utilizado sem a especificação de nenhum parâmetro, em uma operação que envolva dois DataFrames compatíveis (com a mesma definição), o resultado produzido consiste na interseção dos conjuntos:

```
0  rakesh@xyz.com
```

Por fim, vamos falar da operação de diferença, que tem o código um pouco mais complicado. A diferença entre dois conjuntos `R` e `S` é a operação que tem por objetivo é determinar os elementos de `R` que não fazem parte de `S`. Na pandas, é preciso utilizar o método `isin()` (apresentado no capítulo 2) em conjunto com a indexação booleana (também introduzida no capítulo 2) para que seja possível realizar essa operação. Tudo começa com o comando seguinte:

```
df_sql.email.isin(df_python.email)
```

Este comando produz uma Series booleana contendo o valor `False` associado a todos os e-mails do DataFrame `df_sql` que não façam parte de `df_python`. Isto é: ele vai associar `True` ao **índice** do e-mail "rakesh@xyz.com" e `False` para o índice de "ecg@acmecorpus.com".

```
0     True
```

```
1    False
```

De posse desse resultado, basta aplicar a indexação booleana sobre `df_sql` para que seja possível obter apenas os e-mails que foram associados ao valor `False` (ou seja, quem fez o curso de SQL, mas não fez o de Python). É exatamente isto que é feito no comando:

```
so_sql = df_sql[df_sql.email.isin(df_python.email)==False]`
```

Este comando gera o seguinte resultado:

```
1  ecg@acmecorpus.com
```

A obtenção dos alunos que cursaram Python, mas não SQL, é realizada de forma análoga, através do comando `so_python = df_python[df_python.email.isin(df_sql.email)==False]` . Ele produz o seguinte resultado:

```
0           ana@xyz.com
1  jonas@acmecorpus.com
```

Um detalhe muito importante sobre as operações de conjunto é que elas só fazem sentido quando utilizadas sobre DataFrames que possuam a mesma definição. Como sabemos, só podemos fazer a união, interseção e diferença de conjuntos matemáticos que possuam o mesmo tipo de elemento (não faz sentido fazer a interseção, união ou diferença de um conjunto de e-mails com um conjunto de carros, por exemplo).

Comparando dois DataFrames

Uma outra operação simples - porém muitas vezes necessária, especialmente quando estamos lidando com bases de dados muito volumosas - consiste na comparação de dois DataFrames para

determinar se eles são idênticos (armazenam o mesmo conteúdo) ou não. Neste caso, você deve utilizar o método `equals()`, conforme apresentado no exemplo a seguir. Este método retorna `True` quando os dois DataFrames envolvidos no teste forem iguais ou `False` se eles tiverem qualquer diferença, por mínima que seja.

```
#P42: Comparação de DataFrames
import pandas as pd

#(1)-Cria três DataFrames de Filmes
filmes1 = pd.DataFrame({"titulo":["O Filho da Noiva",
                                  "La La Land"],
                        "ano":[2001,
                               2017]})

filmes2 = pd.DataFrame({"titulo":["Noel, Poeta da Vila",
                                  "La La Land"],
                        "ano":[2007,
                               2017]})

filmes3 = pd.DataFrame({"titulo":["O Filho da Noiva",
                                  "La La Land"],
                        "ano":[2001,
                               2017]})

#(2)-Verifica quais DataFrames são iguais
#   (possuem o mesmo conteúdo)
print('filmes1 é igual à filmes2 -> ', filmes1.equals(filmes2))
print('filmes1 é igual à filmes3 -> ', filmes1.equals(filmes3))
```

Saída:

```
Filmes1 é igual à Filmes2 ->  False
Filmes1 é igual à Filmes3 ->  True
```

5.3 JUNÇÃO

A operação de junção (*join* ou *merge*) é talvez a mais

importante dentre as operações para combinação de DataFrames. As próximas seções introduzem os três tipos de junção diretamente disponibilizados pela pandas: junção natural, interna e externa.

Junção natural

A operação de junção natural realiza o **casamento** (*match*) de linhas de um DataFrame `R` com as linhas de outro DataFrame `S`. No entanto, a combinação é feita de forma **seletiva**. Isso significa que, por padrão, a junção natural combina apenas as linhas de `R` e `S` que coincidem em quaisquer atributos que são comuns a ambos os DataFrames. Para que o conceito fique claro, um exemplo é apresentado na figura a seguir.

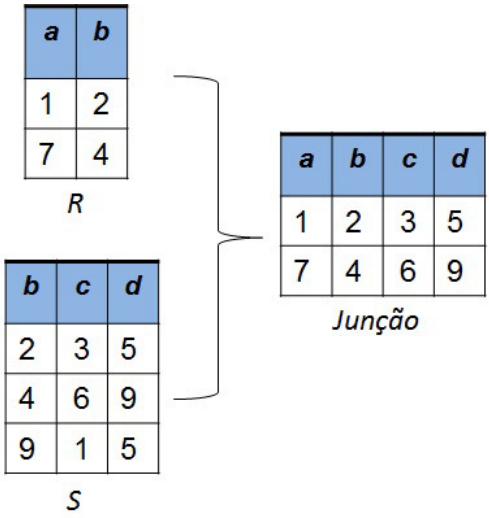

Figura 5.1: Operação de junção natural entre os DataFrames R e S.

Ao observar a figura, podemos constatar que o único atributo

comum a `R` e `S` é o atributo `b`. Portanto, para uma linha de `R` poder casar com uma linha de `S`, é preciso que ambas possuam o mesmo valor para o atributo `b`. Em nosso exemplo, a primeira linha de `R` casa com a primeira de `S`, pois ambas compartilham o mesmo valor (valor 2) para o seu atributo em comum `b`. Esse casamento gera a primeira linha do resultado da junção: (1, 2, 3, 5). O processo é ilustrado na figura seguinte.

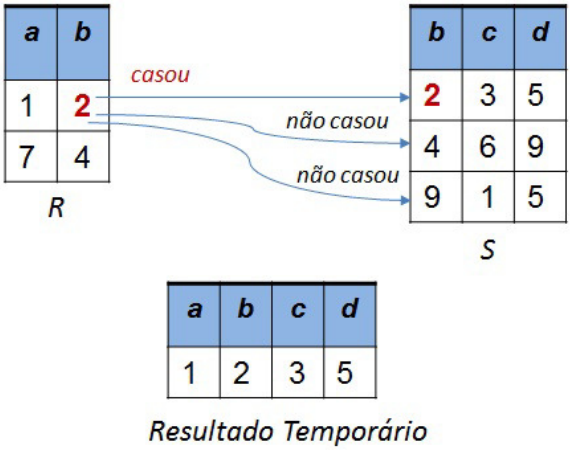

Figura 5.2: Etapa 1 da junção natural entre os DataFrames R e S: casamento da primeira linha de R.

Já a segunda linha de `R` casa com a segunda linha de `S`, uma vez que elas compartilham o mesmo valor (valor 4) para o atributo em comum `b`. Esse casamento gera a segunda linha do resultado, (7, 4, 6, 9), conforme ilustrado na figura seguinte. Este é o resultado final da operação de junção, uma vez que todas as linhas de `R` já foram comparadas com todas as de `S`.

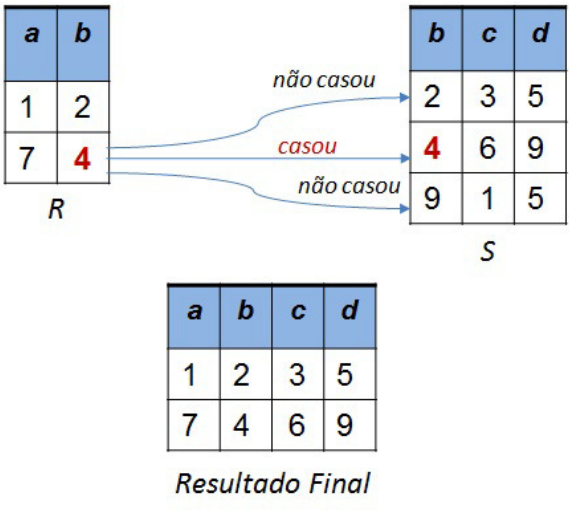

Figura 5.3: Etapa 2 da junção natural entre os DataFrames R e S: casamento da segunda e última linha de R).

Observações:

- No exemplo apresentado, a terceira linha de `S` não casa com nenhuma linha de `R`, pois não há nenhuma linha em `R` com o valor 9 para o atributo em comum `b`. Desta forma, a terceira linha de `S` não tem nenhum efeito no resultado final da operação de junção (ela não é levada para o resultado).
- O atributo comum aos DataFrames é chamado de **atributo de ligação** ou **chave de ligação** ou, simplesmente, **chave** (*key*).
- Enquanto a concatenação produz como resultado um DataFrame mais "comprido" (com linhas empilhadas) a junção produz um DataFrame mais "largo" (que possui atributos de dois DataFrames).

O método `merge()` é utilizado na pandas para implementar todos os tipos de operação de junção. O programa a seguir reproduz o exemplo da junção natural entre os DataFrames `R` e `S` que acabamos de apresentar.

```
#P43: Junção Natural
import pandas as pd

#(1)-Cria os DataFrames R e S
R = pd.DataFrame({"a":[1,7],
                  "b":[2,4]})

S = pd.DataFrame({"b":[2,4,9],
                  "c":[3,6,1],
                  "d":[5,9,5]})


#(2)-Efetua a operação de junção
juncao_natural = pd.merge(R, S)

print('------------------------------')
print("R:")
print(R)
print('------------------------------')
print("S:")
print(S)
print('------------------------------')
print("junção natural entre R e S:")
print(juncao_natural)
```

Resultado:

```
R:
   a  b
0  1  2
1  7  4
------------------------------
S:
   b  c  d
0  2  3  5
1  4  6  9
2  9  1  5
------------------------------
```

```
junção natural entre R e S:
   a  b  c  d
0  1  2  3  5
1  7  4  6  9
```

Conforme mostrado no programa, a sintaxe do método `merge()` para realizar a junção interna de `R` e `S` é trivial. Basta especificar um par de DataFrames como parâmetros, que o método `merge()` se encarrega de identificar, sozinho, qual é o atributo comum a ambos e, assim, processar a junção. Veja que no resultado final o atributo `b` (o único comum aos dois esquemas) aparece apenas uma vez.

Entretanto, também é possível indicar explicitamente qual é o atributo de ligação, bastando para isso utilizar o parâmetro `on`.

```
pd.merge(R, S, on="b")
```

Esse parâmetro é útil em situações em que os DataFrames que são alvo da operação de junção possuam mais de um atributo de mesmo nome, sendo que apenas um deles representa a chave de ligação. Além disso, o parâmetro `on` torna o programa mais legível, pois deixa o atributo de ligação explícito para qualquer pessoa que venha a examinar o código.

Junção interna

Na junção natural, para um atributo ser considerado comum a `R` e `S`, ele precisa ter o **mesmo nome** nestes dois DataFrames. No entanto, isso nem sempre acontece quando estamos lidando com bases de dados reais. Na figura a seguir, um exemplo que ilustra esse tipo situação é apresentado. Temos dois DataFrames chamados `depto` (departamento) e `emp` (empregados), que possuem o **id do departamento** como atributo de ligação. No

entanto, este atributo de ligação possui o nome diferente em cada DataFrame. Veja que ele se chama `id` em `depto` e `idDepto` no DataFrame `emp`.

Id	*NomDepto*	*Local*
D1	Compras	SP
D2	RH	RJ
D3	TI	RJ
D4	Vendas	SP

depto

Num	*Nome*	*Salario*	*IdDepto*
3199	Ana	1600	D2
3269	David	2975	D3
3555	José	1500	
3788	Marina	5000	D2
3844	Luís	3000	D4

emp

Figura 5.4: DataFrames com dados de Departamentos (depto) e Empregados (emp).

Felizmente, a pandas também permite com que seja efetuada a **junção interna** (*inner join*) de DataFrames, um tipo que não exige que a chave de ligação possua nome igual nos DataFrames cuja junção será feita.

Na junção interna, você deve indicar explicitamente a **condição de junção** para o método `merge()`. Isso significa especificar quais são os atributos de ligação de cada DataFrame, utilizando para tal os parâmetros `left_on` e `right_on`. A seguir, o uso destes parâmetros é demonstrado em um programa que implementa a junção interna com o objetivo de combinar os

dados dos funcionários e com os dados de seus departamentos.

```
#P44: Junção Interna
import pandas as pd

#(1)-Cria os DataFrames depto e emp
dic_depto = {"id":["D1","D2","D3","D4"],
             "nomDepto": ["Compras","RH","TI","Vendas"],
             "local":["SP","RJ","RJ","SP"]
             }

dic_emp = {"num":[3199,3269,3555,3788,3844],
           "nome": ["Ana","David","José","Marina","Luís"],
           "salario":[1600,2975,1500,5000,3000],
           "idDepto": ["D2","D3",None,"D2","D4"]
           }


depto = pd.DataFrame(dic_depto)
emp = pd.DataFrame(dic_emp)


#(2)-Efetua a operação de junção
juncao_interna = pd.merge(emp, depto, left_on="idDepto", right_on=
"id")

print('------------------------------')
print("depto:")
print(depto)
print('------------------------------')
print("emp:")
print(emp)
print('------------------------------')
print("junção interna:")
print(juncao_interna)
```

Aqui está a saída gerada. Todos os quatro empregados com departamento cadastrado tiveram os seus dados corretamente combinados com os dados de seus respectivos departamentos:

```
------------------------------
depto:
   id nomDepto local
```

```
0  D1  Compras    SP
1  D2       RH    RJ
2  D3       TI    RJ
3  D4   Vendas    SP
------------------------------
emp:
    num    nome  salario idDepto
0  3199     Ana     1600      D2
1  3269   David     2975      D3
2  3555    José     1500    None
3  3788  Marina     5000      D2
4  3844    Luís     3000      D4
------------------------------
junção interna:
    num    nome  salario idDepto  id nomDepto local
0  3199     Ana     1600      D2  D2       RH    RJ
1  3788  Marina     5000      D2  D2       RH    RJ
2  3269   David     2975      D3  D3       TI    RJ
3  3844    Luís     3000      D4  D4   Vendas    SP
```

Conceitualmente, a junção interna funciona da mesma forma que a junção natural, ou seja, ela combina uma linha de um DataFrame `R` com uma linha de um DataFrame `S` apenas quando há coincidência nos valores das colunas especificadas na condição de junção. Para realizar a junção interna entre dois DataFrames, basta seguir sempre a mesma receita:

- Especificar os nomes dos DataFrames envolvidos na junção separados por vírgula, dentro do método `merge()` (por exemplo, `depto,emp`).
- Utilizar o parâmetro `left_on` indicando o nome do atributo de ligação no DataFrame que foi especificado à esquerda (em nosso exemplo, `emp` foi especificado antes de `depto` , logo `emp` é considerado o DataFrame à esquerda).
- Utilizar o parâmetro `right_on` indicando o nome do atributo de ligação no DataFrame que foi especificado à

direita.

Conforme mostra a saída do programa, cada linha produzida como resultado na junção interna entre `emp` e `depto` contém todos os atributos de `emp` e todos os de `dept`, incluindo as chaves de ligação `idDepto` e `id`. Para selecionar apenas os atributos de interesse, basta aplicar a técnica de fatiamento, introduzida no capítulo 3. No exemplo a seguir, mostra-se como empregar o fatiamento para manter apenas os atributos `num`, `nome` e `nomDepto` (esta técnica também é chamada de **projeção de atributos**, como veremos no próximo capítulo):

```
fatia = juncao_interna[['num','nome','nomDepto']]
print(fatia)
```

```
   num    nome nomDepto
0  3199     Ana       RH
1  3788  Marina       RH
2  3269   David       TI
3  3844    Luís   Vendas
```

Junção externa

Você deve ter percebido que em nossa base de dados exemplo existe um empregado sem departamento cadastrado ("José") e também um departamento que não possui nenhum funcionário ("D1"). As linhas referentes ao empregado e departamento citados acabaram não fazendo parte do resultado produzido pela pela junção interna.

Mas e se houvesse a necessidade de efetuar um estudo envolvendo qualquer empregado, possuindo ele departamento ou não? Ou um relatório que abrangesse qualquer departamento, mesmo aqueles que ainda não tenham nenhum funcionário

alocado? Nesse caso, seria preciso utilizar outro tipo de junção, conhecida como **junção externa**.

As junções externas conectam linhas de dois DataFrames de uma forma mais inclusiva: elas produzem os mesmos resultados da junção interna **acrescidos** de linhas não casadas de um ou ambos os DataFrames. Existem três tipos de junção externa: *left join* (a mais usada), *right join* e *full join*.

A operação de *left join* retorna todas as linhas do DataFrame especificado à esquerda no comando `merge()`, mesmo que não exista casamento (valor equivalente) no DataFrame à direita. Para que o conceito fique claro, vamos retornar para o nosso exemplo envolvendo os DataFrames `emp` e `depto`. O empregado "José" não possui um departamento cadastrado (o atributo `idDepto` possui valor `None` para a linha referente a este empregado em `emp`). Se desejarmos produzir um DataFrame contendo todos os empregados combinados com os dados de seus respectivos departamentos, mas que também inclua os empregados sem departamento cadastrado, é preciso fazer uso do *left join* como mostrado a seguir:

```
j_esq = pd.merge(emp, depto, how="left", left_on="idDepto", right
_on="id")
```

O resultado da operação é apresentado adiante:

```
    num    nome  salario idDepto   id nomDepto local
0  3199     Ana     1600      D2   D2       RH    RJ
1  3269   David     2975      D3   D3       TI    RJ
2  3555    José     1500    None  NaN      NaN   NaN
3  3788  Marina     5000      D2   D2       RH    RJ
4  3844    Luís     3000      D4   D4   Vendas    SP
```

- O parâmetro `how="left"` é usado para indicar à pandas

que se deseja realizar o *left join*.

- No resultado, observe que existem quatro linhas idênticas às que foram geradas pela junção interna, referentes aos empregados "Ana", "David", "Marina" e "Luís". A única linha extra é a do empregado "José", onde os dados deste (dados vindos de `emp`) foram combinados com `NaN` (dados que viriam de `depto` caso José tivesse algum departamento cadastrado).

A junção do tipo *right join* retorna todas as linhas do DataFrame à direita, mesmo que não exista casamento (valor equivalente) no DataFrame à esquerda. Desta forma, para produzir um DataFrame com todos os empregados combinados com os dados de seus departamentos, mas incluindo os empregados sem departamento cadastrado, seria necessário montar o `merge()` da forma indicada a seguir, invertendo a posição de `emp` e `depto`.

```
j_dir = pd.merge(depto, emp, how="right", left_on="id", right_on=
"idDepto")
```

Resultado do *right join*:

```
    id nomDepto local   num    nome  salario idDepto
0   D2       RH    RJ  3199     Ana     1600      D2
1   D2       RH    RJ  3788  Marina     5000      D2
2   D3       TI    RJ  3269   David     2975      D3
3   D4   Vendas    SP  3844    Luís     3000      D4
4  NaN      NaN   NaN  3555    José     1500    None
```

- O parâmetro `how="right"` é usado para indicar à pandas que se deseja realizar o *right join*.
- Embora o resultado seja conceitualmente equivalente ao do *left join*, uma diferença é que no DataFrame resultante os dados vindos de `depto` são colocados antes daqueles vindos de `emp`, uma vez que `depto` foi especificado antes

de `emp` no `merge()`.

Para terminar, vamos apresentar o *full join*, que junta em uma única operação os resultados do *left join* e do *right join*. Esse tipo de junção externa retorna todas as linhas do DataFrame à esquerda e todas as linhas do DataFrame à direita devidamente combinadas, de acordo com a especificação da condição de junção. Caso existam linhas no DataFrame à esquerda que não casem com qualquer linha do DataFrame à direita, elas serão levadas para o resultado final. E caso existam linhas no DataFrame à direita que não casem com qualquer linha do DataFrame à esquerda, elas também serão levadas para o resultado final.

```
j_full = pd.merge(emp, depto, how="outer", left_on="idDepto", rig
ht_on="id")
```

Aqui está o resultado produzido:

```
      num    nome  salario idDepto   id nomDepto local
0  3199.0     Ana   1600.0      D2   D2       RH    RJ
1  3788.0  Marina   5000.0      D2   D2       RH    RJ
2  3269.0   David   2975.0      D3   D3       TI    RJ
3  3555.0    José   1500.0    None  NaN      NaN   NaN
4  3844.0    Luís   3000.0      D4   D4   Vendas    SP
5     NaN     NaN      NaN     NaN   D1  Compras    SP
```

- O parâmetro `how="outer"` é usado para indicar à pandas que se deseja realizar o *full join* (esse tipo de junção é também conhecida como *full outer join*).
- No resultado gerado, veja que foram incluídos tanto o empregado sem departamento ("José"), como o departamento que não está associado a nenhum empregado ("D1").

JOIN() *VERSUS* MERGE()

Além do `merge()`, a pandas oferece outro método para junção de DataFrames, denominado `join()`. Entretanto, esse método é um pouco mais simples (possui menos parâmetros) e costuma ser mais utilizado para realizar a junção através de **índices** em vez de colunas. Para maiores informações consulte a documentação oficial da pandas: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.join.html.