Data Science do Zero

Noções Fundamentais com Python

CAPÍTULO 1
Introdução

"Dados! Dados! Dados!", esbravejou, impaciente. "Não posso fazer tijolos sem barro."
—Arthur Conan Doyle

A Ascensão dos Dados

Vivemos em um mundo cada vez mais imerso em dados. Na web, os sites rastreiam todos os cliques dos usuários. Seu smartphone registra sua localização e velocidade a cada segundo. Os mais fanáticos usam aparelhos turbinados que monitoram continuamente seus batimentos cardíacos, movimentos, dieta e sono. Carros inteligentes registram padrões de direção, casas inteligentes registram padrões domésticos e publicitários inteligentes registram padrões de compra. A Internet é um imenso diagrama de conhecimento e contém (entre outras coisas) uma enorme enciclopédia com referências cruzadas: bases com dados específicos sobre filmes, músicas, esportes, máquinas de pinball, memes e coquetéis; e muitas estatísticas (algumas delas são quase exatas!) produzidas por tantos governos que chega a dar vertigem.

Embaixo desses dados, estão as respostas para inúmeras perguntas que nunca foram feitas. Neste livro, aprenderemos a encontrá-las.

O Que É Data Science?

Aí vai uma piada: o cientista de dados entende mais de estatística do que um cientista da computação e mais de ciência da computação do que um estatístico. (Eu não disse que a piada era boa.) Na verdade, alguns cientistas de dados são — para todos os efeitos — estatísticos; outros são, na prática, engenheiros de software. Alguns são experts em aprendizado de máquina; outros acham que isso é um novo fliperama. Alguns são PhDs com um currículo cheio de artigos importantes; outros nunca leram nenhum trabalho acadêmico (o que é pior para eles). Resumindo, qualquer que seja sua definição de data science, você sempre encontrará praticantes que discordarão total e absolutamente dela.

No entanto, isso não nos impedirá de formular uma definição. Digamos que o cientista de dados extrai conhecimento de dados desorganizados. Atualmente, o mundo está cheio de gente empenhada em transformar dados em conhecimento.

Por exemplo, o site de namoro OkCupid faz milhares de perguntas aos seus membros a fim de encontrar os parceiros mais adequados para eles. Mas a página também analisa esses resultados para definir perguntas aparentemente inócuas que podem determinar a possibilidade de sexo no primeiro encontro (*https://theblog.okcupid.com/the-best-questions-for-a-first-date-dba6adaa9df2*).

O Facebook pede que você informe sua cidade natal e sua localização atual para que, supostamente, seus amigos o encontrem e adicionem com mais facilidade. Porém, a página também analisa esses dados para identificar padrões de mobilidade global (*https://www.facebook.com/notes/facebook-data-science/coordinated-migration/10151930946453859*) e a localização das torcidas dos clubes de futebol americano (*https://www.facebook.com/notes/facebook-data-science/nfl-fans-on-facebook/10151298370823859*).

Uma grande empresa varejista, a Target registra as compras e interações dos clientes nas lojas online e físicas e usa esses dados em um modelo preditivo (*https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html*) para encontrar clientes grávidas e otimizar suas ofertas de artigos infantis.

Em 2012, a campanha de Barack Obama contratou dezenas de cientistas de dados para analisar a situação e encontrar formas de identificar os eleitores que precisavam de mais atenção, otimizar os programas de captação de recursos junto a doadores específicos e direcionar as iniciativas de incentivo ao voto para os pontos mais críticos. Em 2016, a campanha de Donald Trump testou uma incrível variedade de anúncios online (*https://www.wired.com/2016/11/facebook-won-trump-election-not-just-fake-news/*) e analisou os dados para determinar os mais eficientes.

Agora leia isto antes de largar o livro: às vezes, os cientistas de dados também usam suas habilidades para o bem — por exemplo, para aumentar a eficiência das ações governamentais (*https://www.marketplace.org/2014/08/22/tech/beyond-ad-clicks-using-big-data-social-good*), ajudar os sem-teto (*https://dssg.uchicago.edu/2014/08/20/tracking-the-paths-of-homelessness/*) e melhorar a saúde pública (*https://plus.google.com/communities/109572103057302114737*). Entretanto sua carreira não sofrerá nenhum abalo se você só quiser fazer os usuários clicarem nos anúncios.

Motivação Hipotética: DataSciencester

Parabéns! Você acabou de ser contratado para coordenar o setor de data science da DataSciencester, *a* rede social dos cientistas de dados.

Na primeira edição, eu achava que "uma rede social para cientistas de dados" era só uma hipótese divertida e meio ingênua. De lá para cá, essas redes foram mesmo criadas e captaram muitos recursos de empresas de capital de risco, bem mais dinheiro do que eu ganhei com o livro. Essa talvez seja uma lição valiosa sobre a relação entre hipóteses divertidas e meio ingênuas e o mercado editorial.

Apesar do seu *público-alvo*, a DataSciencester nunca investiu no seu setor de data science. (Na verdade, a DataSciencester também nunca investiu no seu produto principal.) Esse será seu trabalho! Ao longo do livro, aprenderemos os conceitos do data science resolvendo problemas comuns à prática profissional. Analisaremos os dados fornecidos expressamente pelo usuário, os gerados em suas interações com um site e os obtidos nos experimentos que desenvolveremos.

E, como a DataSciencester sofre da síndrome do "não inventado aqui", construiremos as ferramentas do zero. Ao final, você terá compreendido bem os fundamentos do data science e poderá aplicar essas habilidades em uma empresa com uma proposta menos duvidosa ou em outros problemas do seu interesse.

Seja bem-vindo e boa sorte! (Calças jeans estão liberadas às sextas, e o banheiro fica no final do corredor, à direita.)

Encontrando Conectores-Chave

No seu primeiro dia na DataSciencester, o vice-presidente de Networking lhe faz várias perguntas sobre os usuários. Agora que você está aqui, ele está muito empolgado.

Especificamente, ele quer que você identifique os "conectores-chave" entre os cientistas de dados e, para isso, lhe dá um data dump da rede da DataSciencester. (Na vida real, você geralmente não recebe os dados necessários. O Capítulo 9 explica como obtê-los.)

Como é esse data dump? Trata-se de uma lista em que cada usuário é representado por um `dict`, que contém o seu `id` (um número) e seu `name` (os nomes foram extraídos do catálogo internacional de nomes aleatórios):

```
users = [
   { “id”: 0, “name”: “Hero” },
   { “id”: 1, “name”: “Dunn” },
   { “id”: 2, “name”: “Sue” },
   { “id”: 3, “name”: “Chi” },
   { “id”: 4, “name”: “Thor” },
   { “id”: 5, “name”: “Clive” },
   { “id”: 6, “name”: “Hicks” },
   { “id”: 7, “name”: “Devin” },
   { “id”: 8, “name”: “Kate” },
   { “id”: 9, “name”: “Klein” }
]
```

Você também recebe os dados de “amizades ”, reunidos em uma lista de pares de IDs:

```
friendship_pairs = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
                    (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

Por exemplo, a tupla `(0, 1)` indica que o cientista de dados com o `id` 0 (Hero) e o cientista de dados com o `id` 1 (Dunn) são amigos. A Figura 1-1 representa a rede.

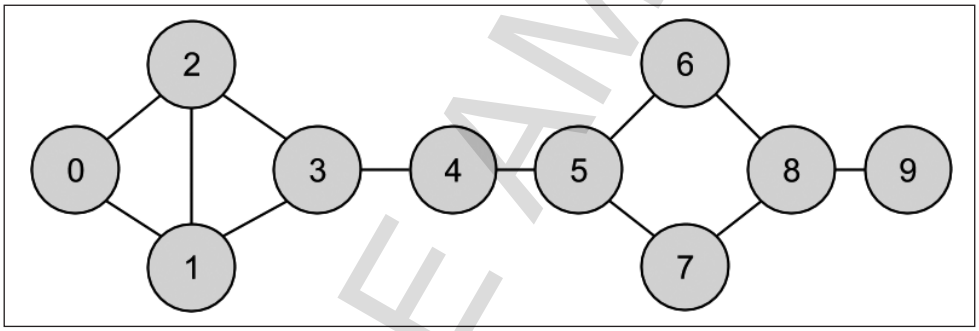

Figura 1-1. A rede da DataSciencester

Representar as amizades em uma lista de pares não é muito eficiente. Para encontrar todas as amizades do usuário 1, você precisa iterar todos os pares em busca dos que contêm o 1. Se houver um grande número de pares, isso levará muito tempo.

Em vez disso, criaremos um `dict` no qual as chaves serão os `id`s dos usuários e os valores serão listas com os `id`s dos seus amigos. (É muito rápido pesquisar em um `dict`.)

Não se apegue demais aos detalhes do código agora. No Capítulo 2, teremos um curso intensivo de Python. Por enquanto, tente captar a ideia central do exemplo.

Ainda precisamos conferir todos os pares para criar o `dict`, mas só uma vez; depois, teremos pesquisas eficientes:

```
# Inicialize o dict com uma lista vazia para cada id de usuário:
friendships = {user["id"]: [] for user in users}

# Em seguida, execute um loop pelos pares de amigos para preenchê-la:
for i, j in friendship_pairs:

friendships[i].append(j) # Adicione j como amigo do usuário i
friendships[j].append(i) # Adicione i como amigo do usuário j
```

Agora que colocamos as amizades em um `dict`, podemos facilmente fazer perguntas ao nosso grafo, como: "Qual é o número médio de conexões?"

Primeiro, determinamos o número *total* de conexões somando os tamanhos de todas as listas de `friends`:

```
def number_of_friends(user):
   """Quantos amigos tem o _user_?"""
   user_id = user["id"]
   friend_ids = friendships[user_id]
   return len(friend_ids)

total_connections = sum(number_of_friends(user)
                        for user in users)          # 24
```

Em seguida, basta dividir pelo número de usuários:

```
num_users = len(users)                              # tamanho da lista de usuários
avg_connections = total_connections / num_users     # 24 / 10 == 2.4
```

Também é fácil encontrar as pessoas mais conectadas — as que possuem o maior número de amigos.

Como o número de usuários não é muito grande, podemos colocá-los em ordem decrescente, dos que têm "mais amigos" para os que têm "menos amigos":

```
# Crie uma lista (user_id, number_of_friends).
num_friends_by_id = [(user["id"], number_of_friends(user))
                     for user in users]

num_friends_by_id.sort(                             # Classifique a lista
      key=lambda id_and_friends: id_and_friends[1],    # por num_friends
      reverse=True)                                 # do maior para o menor

# Cada par é (user_id, num_friends):
# [(1, 3), (2, 3), (3, 3), (5, 3), (8, 3),
#  (0, 2), (4, 2), (6, 2), (7, 2), (9, 1)]
```

Pense nisso como um modo de identificar as pessoas que são, de certa forma, centrais para a rede. Na verdade, acabamos de computar uma métrica da rede chamada *centralidade de grau* (Figura 1-2).

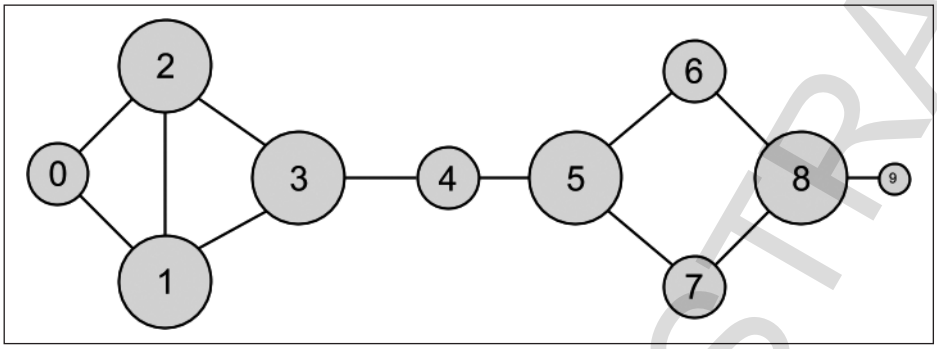

Figura 1-2. A rede DataSciencester dimensionada de acordo com o grau

Apesar de ser fácil de calcular, essa operação nem sempre gera os resultados desejados ou esperados. Por exemplo, Thor (`id` 4) tem duas conexões e Dunn (`id` 1), três. Mas, se observarmos a rede, temos a impressão de que Thor deveria estar mais centralizado. No Capítulo 22, analisaremos minuciosamente as redes e veremos noções de centralidade mais complexas, que podem ou não seguir nossa intuição.

Cientistas de Dados Que Você Talvez Conheça

Enquanto você está preenchendo os papéis de admissão, a vice-presidente de Relações Internas vai à sua mesa. Ela quer incrementar as conexões entre os membros do site e pede que você desenvolva um sugestor do tipo "Cientistas de Dados Que Você Talvez Conheça".

Sua primeira ideia é sugerir que os usuários podem conhecer os amigos dos seus amigos. Então, você escreve o código para iterar os amigos e coletar os amigos dos amigos:

```
def foaf_ids_bad(user):
    """foaf significa "friend of a friend" [amigo de um amigo] """
    return [foaf_id
            for friend_id in friendships[user["id"]]
            for foaf_id in friendships[friend_id]]
```

Quando chamamos `users[0]` (Hero), obtemos o seguinte:

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

O resultado traz o usuário 0 duas vezes, pois Hero também é amigo dos seus amigos. Também traz os usuários 1 e 2, apesar de eles já serem amigos de Hero. E traz o usuário 3 duas vezes, pois Chi é acessível por meio de dois amigos:

```
print(friendships[0])  # [1, 2]
print(friendships[1])  # [0, 2, 3]
print(friendships[2])  # [0, 1, 3]
```

Como essas informações sobre quem é amigo de quem parecem ser interessantes, vamos gerar uma contagem de amigos em comum, porém excluindo as pessoas que o usuário já conhece:

```
from collections import Counter                          # não é carregado por padrão

def friends_of_friends(user):
    user_id = user["id"]
    return Counter(
            foaf_id
            for friend_id in friendships[user_id]        # Para cada amigo meu,
            for foaf_id in friendships[friend_id]        # encontre os amigos deles
            if foaf_id != user_id                        # que não sejam eu
            and foaf_id not in friendships[user_id]      # e não sejam meus amigos.
    )


print(friends_of_friends(users[3]))                      # Counter({0: 2, 5: 1})
```

Essa operação informa corretamente a Chi (`id` 3) que ela possui dois amigos em comum com Hero (`id` 0), mas só um amigo em comum com Clive (`id` 5).

Como cientista de dados, talvez você queira encontrar usuários com interesses similares. (Esse é um bom exemplo do fator "experiência substancial" do data science). Então, depois de suar um pouco a camisa, você obtém os seguintes dados, reunidos como uma lista de pares (`user_id, interest`):

```
interests = [
    (0, "Hadoop"), (0, "Big Data"), (0, "HBase"), (0, "Java"),
    (0, "Spark"), (0, "Storm"), (0, "Cassandra"),
    (1, "NoSQL"), (1, "MongoDB"), (1, "Cassandra"), (1, "HBase"),
    (1, "Postgres"), (2, "Python"), (2, "scikit-learn"), (2, "scipy"),
    (2, "numpy"), (2, "statsmodels"), (2, "pandas"), (3, "R"), (3, "Python"),
    (3, "statistics"), (3, "regression"), (3, "probability"),
    (4, "machine learning"), (4, "regression"), (4, "decision trees"),
    (4, "libsvm"), (5, "Python"), (5, "R"), (5, "Java"), (5, "C++"),
    (5, "Haskell"), (5, "programming languages"), (6, "statistics"),
    (6, "probability"), (6, "mathematics"), (6, "theory"),
    (7, "machine learning"), (7, "scikit-learn"), (7, "Mahout"),
    (7, "neural networks"), (8, "neural networks"), (8, "deep learning"),
    (8, "Big Data"), (8, "artificial intelligence"), (9, "Hadoop"),
    (9, "Java"), (9, "MapReduce"), (9, "Big Data")
]
```

Por exemplo, Hero (`id` 0) não possui amigos em comum com Klein (`id` 9), mas os dois se interessam por Java e Big Data.

É fácil construir uma função para encontrar usuários com o mesmo interesse:

```
def data_scientists_who_like(target_interest):
    """Encontre os ids dos usuários com o mesmo interesse."""
    return [user_id
```

```
            for user_id, user_interest in interests
            if user_interest == target_interest]
```

A operação funciona, porém tem que examinar a lista de interesses inteira a cada busca. Quando há muitos usuários e interesses (ou para fazer muitas buscas), é melhor construir um índice de interesses para usuários:

```
from collections import defaultdict

# As chaves são interesses, os valores são listas de user_ids com o interesse em questão
user_ids_by_interest = defaultdict(list)

for user_id, interest in interests:
    user_ids_by_interest[interest].append(user_id)
```

E outro índice, de usuários para interesses:

```
# As chaves são user_ids, os valores são listas de interesses do user_id em questão.
interests_by_user_id = defaultdict(list)

for user_id, interest in interests:
    interests_by_user_id[user_id].append(interest)
```

Agora é fácil determinar quem tem mais interesses em comum com um usuário específico:

- Faça a iteração dos interesses do usuário;
- Para cada interesse, faça a iteração dos outros usuários com o mesmo interesse;
- Conte quantas vezes cada usuário aparece.

No código:

```
def most_common_interests_with(user):
   return Counter(
       interested_user_id
       for interest in interests_by_user_id[user[“id”]]
       for interested_user_id in user_ids_by_interest[interest]
       if interested_user_id != user[“id”]
   )
```

Podemos usar esse exemplo para construir um recurso mais sofisticado do tipo “Cientistas de Dados Que Você Talvez Conheça”, combinando amigos e interesses em comum. Abordaremos essas aplicações no Capítulo 23.

Salários e Experiência

Você já está saindo para o almoço quando o vice-presidente de Relações Públicas lhe pede alguns fatos curiosos sobre os salários dos cientistas de dados. Esse tópico é sensível, no

entanto, ele lhe fornece um conjunto de dados anônimos contendo o `salary` (o salário de cada usuário, em dólares) e o `tenure` (a experiência como cientista de dados, em anos):

```
salaries_and_tenures = [(83000, 8.7), (88000, 8.1),
                        (48000, 0.7), (76000, 6),
                        (69000, 6.5), (76000, 7.5),
                        (60000, 2.5), (83000, 10),
                        (48000, 1.9), (63000, 4.2)]
```

Naturalmente, o primeiro passo é plotar os dados (veremos essa operação no Capítulo 3). Os resultados estão na Figura 1-3.

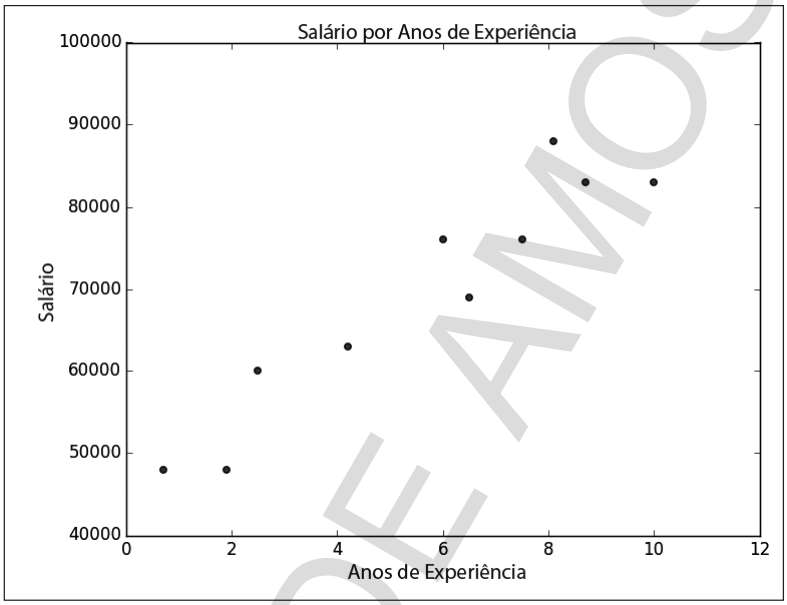

Figura 1-3. Salário por anos de experiência

Certamente, os mais experientes tendem a ganhar melhor, entretanto como isso pode se tornar um fato curioso? Primeiro, você analisa a média salarial por anos de experiência:

```
# As chaves são anos, os valores são listas de salários por anos de experiência.
salary_by_tenure = defaultdict(list)

for salary, tenure in salaries_and_tenures:
    salary_by_tenure[tenure].append(salary)

# As chaves são anos, cada valor é o salário médio associado ao número de anos de experiência.
average_salary_by_tenure = {
    tenure: sum(salaries) / len(salaries)
    for tenure, salaries in salary_by_tenure.items()
}
```

Essa informação não parece muito útil, já que os usuários não têm os mesmos anos de experiência, ou seja, só os seus salários individuais são indicados:

```
{0.7: 48000.0,
 1.9: 48000.0,
 2.5: 60000.0,
 4.2: 63000.0,
 6: 76000.0,
 6.5: 69000.0,
 7.5: 76000.0,
 8.1: 88000.0,
 8.7: 83000.0,
 10: 83000.0}
```

Talvez seja melhor fazer buckets de experiências:

```
def tenure_bucket(tenure):
    if tenure < 2:
       return “less than two”
    elif tenure < 5:
       return “between two and five”
    else:
       return “more than five”
```

Em seguida, agrupamos os salários correspondentes a cada bucket:

```
# As chaves são buckets de anos de experiência, os valores são as listas de salários
associadas ao bucket em questão.
salary_by_tenure_bucket = defaultdict(list)

for salary, tenure in salaries_and_tenures:
    bucket = tenure_bucket(tenure)
    salary_by_tenure_bucket[bucket].append(salary)
```

E, finalmente, computamos a média salarial de cada grupo:

```
# As chaves são buckets de anos de experiência, os valores são a média salarial do
bucket em questão.
average_salary_by_bucket = {
  tenure_bucket: sum(salaries) / len(salaries)
  for tenure_bucket, salaries in salary_by_tenure_bucket.items()
}
```

O resultado é mais interessante:

```
{‘between two and five’: 61500.0,
 ‘less than two’: 48000.0,
 ‘more than five’: 79166.66666666667}
```

Temos uma boa sacada: “Os cientistas de dados com mais de cinco anos de experiência ganham 65% mais do que os colegas com pouca ou nenhuma experiência!”

No entanto, escolhemos os buckets de forma bem arbitrária. Nosso objetivo, na verdade, é indicar o efeito salarial — em média — de cada ano de experiência. Além de deixar o fato mais curioso, podemos *fazer previsões* sobre salários que não conhecemos a partir dessa informação. Exploraremos mais essa ideia no Capítulo 14.

Contas Pagas

Quando você volta para sua mesa, a vice-presidente de Receita está lhe esperando, pois quer saber quais usuários pagam por suas contas e quais não pagam. (Ela tem os nomes deles, mas essa informação não é muito útil.)

Você logo percebe uma relação aparente entre os anos de experiência e as contas pagas:

```
0.7  paid
1.9  unpaid
2.5  paid
4.2  unpaid
6.0  unpaid
6.5  unpaid
7.5  unpaid
8.1  unpaid
8.7  paid
10.0 paid
```

Os usuários com pouquíssimos e muitos anos de experiência tendem a pagar; os usuários com experiência mediana, não. Logo, para criar um modelo — mesmo não havendo dados suficientes para isso —, você deveria prever "paid" para os usuários com pouquíssimos, e muitos, anos de experiência e "unpaid" para os usuários com experiência mediana:

```
def predict_paid_or_unpaid(years_experience):
  if years_experience < 3.0:
    return "paid"
  elif years_experience < 8.5:
    return "unpaid"
  else:
    return "paid"
```

Claro, definimos os referenciais no olhômetro.

Com mais dados (e mais cálculos), podemos construir um modelo para prever a probabilidade de pagamento dos usuários com base nos seus anos de experiência. Investigaremos esse tipo de problema no Capítulo 16.

Tópicos de Interesse

Já no final do seu primeiro dia, a vice-presidente de Estratégia de Conteúdo solicita dados sobre os tópicos que mais interessam aos usuários para planejar o calendário do blog dela. Você já tem os dados brutos do projeto do sugestor de amigos:

```
interests = [
    (0, “Hadoop”), (0, “Big Data”), (0, “HBase”), (0, “Java”),
    (0, “Spark”), (0, “Storm”), (0, “Cassandra”),
    (1, “NoSQL”), (1, “MongoDB”), (1, “Cassandra”), (1, “HBase”),
    (1, “Postgres”), (2, “Python”), (2, “scikit-learn”), (2, “scipy”),
    (2, “numpy”), (2, “statsmodels”), (2, “pandas”), (3, “R”), (3, “Python”),
    (3, “statistics”), (3, “regression”), (3, “probability”),
    (4, “machine learning”), (4, “regression”), (4, “decision trees”),
    (4, “libsvm”), (5, “Python”), (5, “R”), (5, “Java”), (5, “C++”),
    (5, “Haskell”), (5, “programming languages”), (6, “statistics”),
    (6, “probability”), (6, “mathematics”), (6, “theory”),
    (7, “machine learning”), (7, “scikit-learn”), (7, “Mahout”),
    (7, “neural networks”), (8, “neural networks”), (8, “deep learning”),
    (8, “Big Data”), (8, “artificial intelligence”), (9, “Hadoop”),
    (9, “Java”), (9, “MapReduce”), (9, “Big Data”)
]
```

Um modo simples (mas razoavelmente entediante) de encontrar os interesses mais populares é contar as palavras:

1. Escreva os interesses em letras minúsculas (talvez os usuários tenham usado letras maiúsculas);
2. Divida-os em palavras;
3. Conte os resultados.

No código:

```
words_and_counts = Counter(word
                           for user, interest in interests
                           for word in interest.lower().split())
```

Essa operação facilita a listagem de palavras que ocorrem mais de uma vez:

```
for word, count in words_and_counts.most_common():
    if count > 1:
        print(word, count)
```

Os resultados saem como esperado (a menos que você queira dividir "scikit-learn" em duas palavras; nesse caso, o resultado não sai como esperado):

```
learning 3
java 3
python 3
big 3
data 3
hbase 2
regression 2
cassandra 2
statistics 2
probability 2
hadoop 2
networks 2
machine 2
neural 2
scikit-learn 2
r 2
```

Veremos formas mais sofisticadas de extrair tópicos dos dados no Capítulo 21.

Em Frente

Foi um ótimo primeiro dia! Exausto, você sai do prédio antes de ouvir outro pedido. Durma bem, porque amanhã você participará do curso para novos funcionários. (Sim, você trabalhou um dia inteiro sem *nenhuma* formação inicial. Culpa do RH.)