CAPÍTULO 1

# Introdução

*"Dados! Dados! Dados!" ele gritou impacientemente. "Não posso fabricar tijolos sem barro."*
—Arthur Conan Doyle

## A Ascensão dos Dados

Vivemos em um mundo que está soterrado por dados. Os websites rastreiam todos os cliques de todos os usuários. Seu smartphone está fazendo um registro da sua localização e sua velocidade a cada segundo diariamente. Atletas avaliados usam pedômetros com esteroides que estão sempre registrando suas batidas do coração, hábitos de movimentos, dieta e padrões do sono. Carros inteligentes coletam hábitos de direção, casas inteligentes coletam hábitos de moradia e marqueteiros inteligentes coletam hábitos de compra. A própria internet representa um diagrama grande de conhecimento que contém (entre outras coisas) uma enorme enciclopédia de referências cruzadas: bases de dados específicos de domínio sobre filmes, música, resultados de esportes, máquinas de pinball, memes e coquetéis; e muitas estatísticas do governo (algumas delas são verdades!) sobre tantos governos que causariam um nó na sua cabeça.

Soterrados sob esses dados estão as respostas para as inúmeras questões que ninguém nunca pensou em perguntar. Neste livro, aprenderemos como encontrá-las.

## O Que É Data Science?

Há uma piada que diz que um cientista de dados é alguém que sabe mais sobre estatística do que um cientista da computação e mais sobre 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 propósitos práticos — estatísticos, enquanto outros são quase indistinguíveis dos engenheiros de software. Alguns são experts em aprendizado de máquina, enquanto outros não conseguiram aprender muita coisa sobre o assunto. Alguns são PhDs com um impressio-

nante registro de publicações, enquanto outros nunca leram um trabalho acadêmico (apesar de ser uma vergonha). Resumindo, basicamente não importa como você define data science, pois você encontrará praticantes para quem a definição está total e absolutamente errada.

De qualquer forma, não permitiremos que isso nos impeça de tentar. Digamos que um cientista de dados seja alguém que extrai conhecimento de dados desorganizados. O mundo de hoje está cheio de pessoas tentando transformar dados em conhecimento.

Por exemplo, o site de namoro OkCupid pede que seus membros respondam milhares de perguntas a fim de encontrar as combinações mais adequadas para eles. Mas também analisa tais resultados para descobrir perguntas aparentemente inócuas as quais você poderia perguntar para alguém e descobrir qual a possibilidade de essa pessoa dormir com você no primeiro encontro (*http://bit.ly/1EQU0hI*).

O Facebook pede que você adicione sua cidade natal e sua localização atual, supostamente para facilitar que seus amigos o encontrem e se conectem com você. Porém, ele também analisa essas localizações para identificar padrões de migração global (*http://on.fb.me/1EQTq3A*) e onde vivem os fã-clubes dos times de futebol (*http://on.fb.me/1EQTvnO*).

Como uma grande empresa, a Target rastreia suas encomendas e interações, tanto online como na loja física. Ela usa os dados em um modelo preditivo (*http://nyti.ms/1EQTznL*) para saber quais clientes estão grávidas a fim de melhorar sua oferta de artigos relacionados a bebês.

Em 2012, a campanha do Obama empregou muitos cientistas de dados que mineraram os dados e experimentaram uma forma de identificar os eleitores que precisavam de uma atenção extra, otimizar programas e recursos para a captação de fundos de doadores específicos e focando esforços para votos onde provavelmente eles teriam sido úteis. Normalmente, é de comum acordo pensar que esses esforços tiveram um papel importante na reeleição do presidente, o que significa que é seguro apostar que as campanhas políticas do futuro se tornarão cada vez mais dependentes de dados, resultando em uma corrida armamentista sem fim de data science e coleta de dados.

Agora, antes que você se sinta muito exausto: alguns cientistas de dados também usam suas habilidades para o bem, ocasionalmente — usar os dados para tornar o governo mais eficiente (*http://bit.ly/1EQTGiW*), ajudar os desabrigados (*http://bit.ly/1EQTIYl*), e melhorar a saúde pública (http://bit.ly/1EQTPTv). Mas, certamente, não afetará sua carreira se você gosta de encontrar a melhor maneira de fazer o público clicar em seus anúncios.

## Motivação Hipotética: DataSciencester

Parabéns! Você acabou de ser contratado para liderar os esforços de data science na DataSciencester, *a* rede social para cientistas de dados.

Apesar de ser *para* os cientistas de dados, a DataSciencester nunca investiu em construir sua própria atividade de data science (na verdade, a DataSciencester nunca investiu em construir seu próprio produto). Esse será seu trabalho! No decorrer do livro, aprenderemos sobre os conceitos de data science ao resolver problemas com os quais você se depara no traba-

lho. Algumas vezes, olharemos para os dados explicitamente fornecidos pelo usuário, outras vezes olharemos para os gerados por suas interações com um site e, às vezes, olharemos para os dados dos experimentos que projetaremos.

E, devido à DataSciencester possuir uma forte mentalidade de "não-foi-inventado-aqui", nós construiremos nossas próprias ferramentas do zero. No final, você terá um sólido entendimento dos fundamentos de data science. Você estará pronto para aplicar suas habilidades em sua empresa com uma premissa menos duvidosa, ou em qualquer outro problema que vier a despertar seu interesse.

Bem-vindo a bordo e boa sorte! 'Você pode usar jeans às sextas e o toalete é no final do corredor à direita.'

## Encontrando Conectores-Chave

É seu primeiro dia de trabalho na DataSciencester e o vice-presidente de Rede (networking) está cheio de perguntas sobre seus usuários. Até agora, ele não teve ninguém para perguntar, então ele está muito empolgado em ter você aqui.

Particularmente, ele quer que você identifique quem são os "conectores-chave" entre os cientistas de dados. Para isso, ele lhe dá uma parte de toda a rede da DataSciencester. Na vida real, você geralmente não recebe os dados de que precisa. O Capítulo 9 é voltado para a obtenção de dados.

Com o que se parece essa parte dos dados? Ela consiste em uma lista de usuários, cada um representado por um `dict` que contém um `id` (um número) para cada usuário ou usuária e um `name` (que por uma das grandes coincidências cósmicas que rima com o `id` do usuário):

```
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" }
]
```

Ele também fornece dados "amigáveis", representados por uma lista de pares de IDs:

```
friendships = [(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 a `id` 0 (Hero) e o cientista de dados com a `id` 1 (Dunn) são amigos. A rede é ilustrada na Figura 1-1.

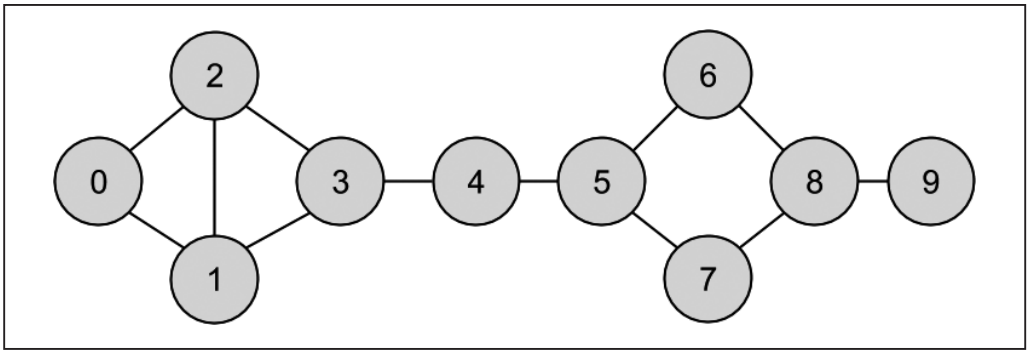


*Figura 1-1. A rede da DataSciencester*

Já que representamos nossos usuários como `dict`s, é fácil de aumentá-los com dados extras.

Não fique preso aos detalhes do código agora. No Capítulo 2, vamos levá-lo a um curso intensivo de Python. Por enquanto, tente pegar o sentido geral do que estamos fazendo.

Por exemplo, talvez nós queiramos adicionar uma lista de amigos para cada usuário. Primeiro nós configuramos a propriedade `friends` de cada usuário em uma lista vazia:

```
for user in users:
    user["friends"] = []
```

Então, nós povoamos a lista com os dados de `friendships`:

```
for i, j in friendships:
    # isso funciona porque users[i] é o usuário cuja id é i
    users[i]["friends"].append(users[j]) # adiciona i como um amigo de j
    users[j]["friends"].append(users[i]) # adiciona j como um amigo de i
```

Uma vez que o `dict` de cada usuário contenha uma lista de amigos, podemos facilmente perguntar sobre nosso gráfico, como "qual é o número médio de conexões?"

Primeiro, encontramos um número *total* de conexões, resumindo os tamanhos de todas as listas de `friends`:

```
def number_of_friends(user):
    """quantos amigos o usuário tem?"""
    return len(user["friends"])                  # tamanho da lista friend_ids

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

Então, apenas dividimos pelo número de usuários:

```
from __future__ import division                  # divisão inteira está incompleta
num_users = len(users)                           # tamanho da lista de usuários
avg_connections = total_connections / num_users   # 2.4
```

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

Como não há muitos usuários, podemos ordená-los de "muito amigos" para "menos amigos":

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

sorted(num_friends_by_id,                                  # é ordenado
       key=lambda (user_id, num_friends): num_friends,     # 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)]
```

Uma maneira de pensar sobre o que nós fizemos é uma maneira de identificar as pessoas que são, de alguma forma, centrais para a rede. Na verdade, o que acabamos de computar é uma rede métrica de *grau de centralidade* (Figura 1-2).

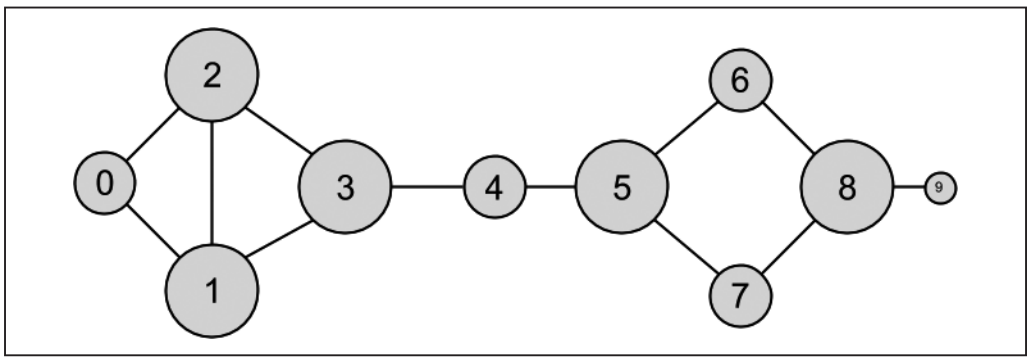


*Figura 1-2. A rede DataSciencester ordenada pelo grau*

Essa figura tem a vantagem de ser fácil de calcular, mas nem sempre lhe dá os resultados que você queria ou esperaria. Por exemplo, a rede Thor da DataSciencester (`id` 4) possui somente duas conexões enquanto que Dunn (`id` 1) possui três. Ainda olhando para a rede, parece que Thor deveria ser mais centralizado. No Capítulo 21, investigaremos as redes com mais detalhe, e veremos noções de centralidade mais complexas que podem ou não corresponder melhor à nossa intuição.

## Cientistas de Dados Que Você Talvez Conheça

Enquanto você está preenchendo os papéis de admissão, a vice-presidente da Fraternidade chega a sua mesa. Ela quer estimular mais conexões entre os seus membros, e pede que você desenvolva sugestões de "Cientistas de Dados Que Você Talvez Conheça".

Seu primeiro instinto é sugerir um usuário que possa conhecer amigos de amigos. São fáceis de computar: para cada amigo de um usuário, itera sobre os amigos daquela pessoa, e coleta todos os resultados:

```
def friends_of_friend_ids_bad(user):
    # "foaf" é abreviação de "friend of a friend"
    return [foaf["id"]
            for friend in user["friends"]     # para cada amigo de usuário
            for foaf in friend["friends"]]    # pega cada _their_friends
```

Quando chamamos `users[0]` (Hero), ele produz:

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

Isso inclui o usuário 0 (duas vezes), uma vez que Hero é, de fato, amigo de ambos os seus amigos. Inclui os usuários 1 e 2, apesar de eles já serem amigos do Hero. E inclui o usuário 3 duas vezes, já que Chi é alcançável por meio de dois amigos diferentes:

```
print [friend["id"] for friend in users[0]["friends"]]  # [1, 2]
print [friend["id"] for friend in users[1]["friends"]]  # [0, 2, 3]
print [friend["id"] for friend in users[2]["friends"]]  # [0, 1, 3]
```

Saber que as pessoas são amigas-de-amigas de diversas maneiras parece uma informação interessante, então talvez nós devêssemos produzir uma *contagem* de amigos em comum. Definitivamente, devemos usar uma função de ajuda para excluir as pessoas que já são conhecidas do usuário:

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

def not_the_same(user, other_user):
    """dois usuários não são os mesmos se possuem ids diferentes"""
    return user["id"] != other_user["id"]

def not_friends(user, other_user):
    """other_user não é um amigo se não está em user["friends"];
    isso é, se é not_the_same com todas as pessoas em user["friends"]"""
    return all(not_the_same(friend, other_user)
               for friend in user["friends"])

def friends_of_friend_ids(user):
    return Counter(foaf["id"]
                   for friend in user["friends"]    # para cada um dos meus amigos
                   for foaf in friend["friends"]    # que contam *their* amigos
                   if not_the_same(user, foaf)      # que não sejam eu
                   and not_friends(user, foaf))     # e que não são meus amigos

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

Isso diz sobre Chi (`id` 3) que ela possui dois amigos em comum com Hero (`id` 0) mas somente um amigo em comum com Clive (`id` 5).

Como um cientista de dados, você sabe que você pode gostar de encontrar usuários com interesses similares (esse é um bom exemplo do aspecto "competência significativa" de data science). Depois de perguntar por aí, você consegue pôr as mãos nesse dado, 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, Thor (`id` 4) não possui amigos em comum com Devin (`id` 7), mas compartilham do interesse em aprendizado de máquina.

É fácil construir uma função que encontre usuários com o mesmo interesse:

```
def data_scientists_who_like(target_interest):
    return [user_id
            for user_id, user_interest in interests
            if user_interest == target_interest]
```

Funciona, mas a lista inteira de interesses deve ser examinada para cada busca. Se tivermos muitos usuários e interesses (ou se quisermos fazer muitas buscas), seria melhor construir um índice de interesses para usuários:

```
from collections import defaultdict

# as chaves são interesses, os valores são listas de user_ids com interests
user_ids_by_interest = defaultdict(list)

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

E outro de usuários para interesses:

```
# as chaves são user_ids, os valores são as listas de interests para aquele user_id
interests_by_user_id = defaultdict(list)
```

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

Agora fica fácil descobrir quem possui os maiores interesses em comum com um certo usuário:

- Itera sobre os interesses do usuário.
- Para cada interesse, itera sobre os outros usuários com aquele interesse.
- Mantém a contagem de quantas vezes vemos cada outro usuário.

```
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"])
```

Poderíamos usar esse exemplo para construir um recurso mais rico de "Cientistas de Dados Que Você Deveria Conhecer" baseado em uma combinação de amigos e interesses em comum. Exploraremos esses tipos de aplicações no Capítulo 22.

## Salários e Experiência

Na hora em que você está saindo para o almoço, o vice-presidente de Relações Públicas pergunta se você pode fornecer alguns fatos curiosos sobre quanto os cientistas de dados recebem. Dados de salário é, de fato, um tópico sensível, mas ele consegue fornecer um conjunto de dados anônimos contendo o `salary` (salário) de cada usuário (em dólares) e `tenure` (experiência) como um 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 traçar os dados (veremos como fazê-lo no Capítulo 3). Os resultados se encontram na Figura 1-3.

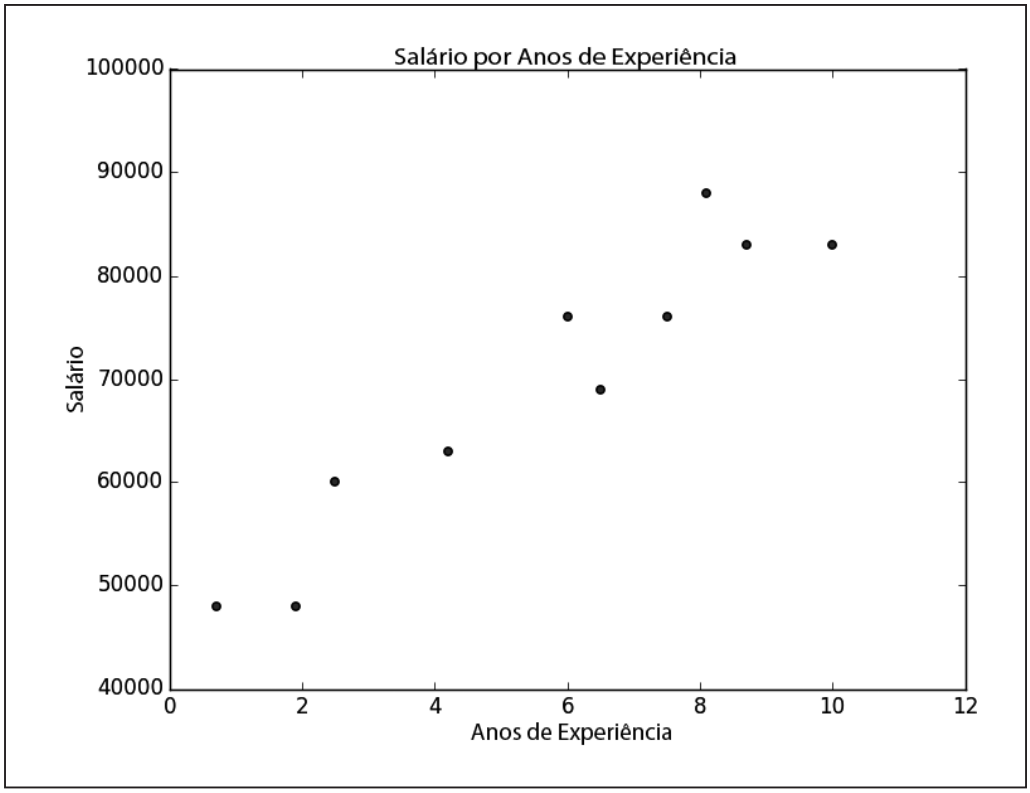


*Figura 1-3. Salário por anos de experiência*

Fica bem claro que os que possuem mais experiência tendem a receber mais. Como você pode transformar isso em um fato curioso? A primeira ideia é analisar a média salarial para cada ano:

```
# as chaves são os anos, os valores são as listas dos salários para cada ano
salary_by_tenure = defaultdict(list)

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

# as chaves são os anos, cada valor é a média salarial para aquele ano
average_salary_by_tenure = {
    tenure : sum(salaries) / len(salaries)
    for tenure, salaries in salary_by_tenure.items()
}
```

Não é muito útil, já que nenhum dos usuários possui o mesmo caso, o que significa que estamos reportando apenas os salários individuais dos usuários:

```
{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 fosse mais proveitoso agrupar os casos:

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

Então, o grupo junta os salários correspondentes para cada agrupamento:

```
# as chaves são agrupamentos dos casos, os valores são as listas
# dos salários para aquele agrupamento
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, computar a média salarial para cada grupo:

```
# as chaves são agrupamentos dos casos, os valores são
# a média salarial para aquele agrupamento
average_salary_by_bucket = {
  tenure_bucket : sum(salaries) / len(salaries)
  for tenure_bucket, salaries in salary_by_tenure_bucket.iteritems()
}
```

que é mais interessante:

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

E você tem um clichê: "os cientistas de dados com mais de cinco anos de experiência recebem 65% a mais do que os que possuem pouca ou nenhuma experiência!"

No entanto, nós escolhemos os casos de forma aleatória. O que realmente queríamos fazer era organizar um tipo de afirmação sobre o efeito do salário — em média — de ter um ano adicional de experiência. Além de tornar o fato mais intrigante, ainda permite que *façamos previsões* sobre salários que não conhecemos. Exploraremos mais essa ideia no Capítulo 14.

## Contas Pagas

Ao voltar para a sua mesa, a vice-presidente da Receita está esperando por você. Ela quer entender melhor quais são os usuários que pagam por contas e quais que não pagam (ela sabe seus nomes, mas essa informação não é essencial).

Você percebe que parece haver uma correspondência entre os anos de experiência e as contas pagas:

```
0.7 paid
1.9 unpaid
2.5 paid
4.2 unpaid
6   unpaid
6.5 unpaid
7.5 unpaid
8.1 unpaid
8.7 paid
10  paid
```

Os usuários com poucos e muitos anos de experiência tendem a pagar; os usuários com uma quantidade mediana de experiência não.

Logo, se você quisesse criar um modelo — apesar de não haver dados o suficiente para servir de base para um — você talvez tentasse prever "paid" para os usuários com poucos e muitos anos de experiência, e "unpaid" para os usuários com quantidade mediana de experiência:

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

Certamente, nós definimos visualmente os cortes.

Com mais dados (e mais matemática), nós poderíamos construir um modelo prevendo a probabilidade de que um usuário pagaria, baseado em seus anos de experiência. Investigaremos esse tipo de problema no Capítulo 16.

## Tópicos de Interesse

Quando seu dia está terminando, a vice-presidente da Estratégia de Conteúdo pede dados sobre em quais tópicos os usuários estão mais interessados, para que ela possa planejar o calendário do seu blog de acordo. Você já possui os dados brutos para o projeto sugerido:

```
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")
]
```

Uma simples forma (e também fascinante) de encontrar os interesses mais populares é fazer uma simples contagem de palavras:

1. Coloque cada um em letras minúsculas (já que usuários diferentes podem ou não escrever seus interesses em letras maiúsculas).
2. Divida em palavras.
3. Conte os resultados.

No código:

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

Isso facilita listar as palavras que ocorrem mais de uma vez:

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

o que fornece o resultado esperado (a menos que você espere que "scikit-learn" possa ser dividido em duas palavras, o que não fornecerá o resultado 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 aprimoradas de extrair tópicos dos dados no Capítulo 20.

## Em Diante

Foi um primeiro dia bem proveitoso! Exausto, você sai do prédio antes que alguém peça algo mais. Tenha uma boa noite de sono, porque amanhã será dia de treinamento para novos funcionários. Sim, você trabalhou um dia inteiro *antes* do treinamento. Culpa do RH.