

»Daten! Daten! Daten!«, schrie er ungeduldig. »Ohne Lehm kann ich keine Ziegel herstellen.«

– Arthur Conan Doyle

Der Aufstieg der Daten

Wir leben in einer Welt, die in Daten ertrinkt. Webseiten erfassen jeden Klick jedes Benutzers. Ihr Smartphone speichert Ihren Aufenthaltsort und Ihr Tempo jede einzelne Sekunde des Tages. »Quantified Selfer« tragen aufgemotzte Schrittmesser, die Herzfrequenz, Bewegungsgewohnheiten, Ernährung und Schlafzyklen registrieren. Intelligente Autos sammeln Informationen über Fahrgewohnheiten, intelligente Häuser sammeln Informationen über Lebensgewohnheiten, und intelligente Marketingleute sammeln Konsumgewohnheiten. Das Internet selbst stellt ein gewaltiges Netzwerk des Wissens dar, das (unter anderem) eine enorme Enzyklopädie mit Querverweisen darstellt – domänenspezifische Datenbanken über Filme, Musik, Sportergebnisse, Flippergeräte, Memes und Cocktails, außerdem viel zu viele Behördenstatistiken (einige davon sind sogar wahr!) von viel zu vielen Regierungen, bis Ihnen schwindelig wird.

Vergraben in diesen Daten sind die Antworten auf unzählige Fragen, die niemand zuvor zu fragen wagte. In diesem Buch werden Sie lernen, wie man sie findet.

Was ist Data Science?

Ein Witz sagt, dass ein Data Scientist jemand ist, der mehr über Statistik weiß als ein Informatiker und mehr über Informatik als ein Statistiker. (Ich habe nicht gesagt, dass es ein guter Witz ist.) Tatsächlich sind einige Data Scientists – für alle praktischen Belange – Statistiker, während andere kaum von Softwareentwicklern zu unterscheiden sind. Einige sind Experten für maschinelles Lernen, während andere nicht einmal den Weg zum Kühlschrank maschinell lernen könnten. Einige haben Dokortitel und eindrucksvolle Publikationslisten, während andere nie einen

akademischen Fachartikel gelesen haben (Schande über sie). Kurz, egal wie Sie Data Science definieren, Sie werden Praktiker finden, auf die diese Definition überhaupt nicht passt.

Das soll uns aber nicht davon abhalten, es zu versuchen. Wir sagen einfach, dass ein Data Scientist jemand ist, der Erkenntnisse aus chaotischen Daten gewinnt. Die heutige Welt ist voll von Menschen, die Daten in Erkenntnis umwandeln möchten.

Das Datingportal OkCupid beispielsweise bittet seine Mitglieder, Tausende von Fragen zu beantworten, um die passendsten Partner für sie zu finden. Aber es wertet diese Ergebnisse auch aus, um unschuldig klingende Fragen zu entwickeln, mit denen sie herausbekommen können, wie wahrscheinlich es ist, dass jemand beim ersten Date mit Ihnen ins Bett geht (<https://theblog.okcupid.com/the-best-questions-for-a-first-date-dba6adaa9df2>).

Facebook fragt Sie nach Ihrer Heimatstadt und Ihrem gegenwärtigen Aufenthaltsort – vorgeblich, um es Ihren Freunden zu erleichtern, Sie zu finden und sich zu befreunden. Aber Facebook analysiert die Orte auch, um in der globalen Migration (<https://www.facebook.com/notes/facebook-data-science/coordinated-migration/10151930946453859>) und den Wohnorten von Footballfans (<https://www.facebook.com/notes/facebook-data-science/nfl-fans-on-facebook/10151298370823859>) Muster zu erkennen.

Target, ein großes Unternehmen im Einzelhandel, verfolgt Ihre Einkäufe und Interaktionen sowohl online als auch im Geschäft. Sie verwenden diese Daten zur Feststellung (<https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html>), welche ihrer Kundinnen schwanger sind, um ihnen besser Babyprodukte präsentieren zu können.

An der Wahlkampagne von Obama nahmen 2012 Dutzende von Data Scientists teil, die Daten durchwühlten und damit experimentierten, um Wähler mit besonderem Zuwendungsbedarf zu identifizieren, optimale auf Spender zugeschnittene Spendenaufrufe zu starten und Aufrufe zur Wahlbeteiligung auf die vielversprechendsten Gegenden zu fokussieren. Und im Jahr 2016 probierte die Trump-Kampagne eine große Zahl von Online-Ads aus (<https://www.wired.com/2016/11/facebook-won-trump-election-not-just-fake-news/>) und analysierte dann die Daten, um herauszufinden, welche funktionieren und welche nicht.

Bevor Sie sich nun völlig abgeschreckt fühlen: Einige Data Scientists setzen ihre Fähigkeiten gelegentlich ein, um Gutes zu tun – etwa um die öffentliche Verwaltung durch Daten effektiver zu machen (<https://www.marketplace.org/2014/08/22/tech/beyond-ad-clicks-using-big-data-social-good>), Obdachlosen zu helfen (<https://dssg.uchicago.edu/2014/08/20/tracking-the-paths-of-homelessness/>) und die Gesundheitsversorgung zu verbessern (<https://plus.google.com/communities/109572103057302114737>). Es wird Ihrer Karriere allerdings gewiss nicht schaden, wenn Sie mit Vergnügen die beste Möglichkeit austüfteln, Leute zum Anklicken von Werbebannern zu bewegen.

Ein motivierendes Szenario: DataSciencester

Herzlichen Glückwunsch! Sie wurden soeben als Leiter der Abteilung für Data Science bei DataSciencester angeheuert, *dem* sozialen Netzwerk für Data Scientists.



Als ich die erste Auflage dieses Buchs schrieb, dachte ich, dass ein »soziales Netzwerk für Data Scientists« sehr nett, aber auch sehr hypothetisch wäre. Tatsächlich wurden jedoch seitdem soziale Netzwerke für Data Scientists geschaffen, und Risikokapitalgeber sammelten viel mehr Geld ein, als ich mit diesem Buch je verdient habe. Dahinter steckt vermutlich eine wertvolle Lektion über die verrückten Ideen von Data Scientists und das Veröffentlichen von Büchern.

Obwohl für Data Scientists geschaffen, hat DataSciencester bisher keine Mühen in die eigenen Data-Science-Praktiken investiert. (Fairerweise muss gesagt werden, dass DataSciencester als Produkt nicht real existiert.) Dies wird Ihre Aufgabe sein! Im Verlauf dieses Buchs werden Sie Data-Science-Methoden dadurch kennenlernen, dass Sie die Aufgaben an Ihrem neuen Arbeitsplatz bewältigen. Bisweilen werden wir uns direkt von Nutzern eingegebene Daten anschauen, bisweilen durch Interaktionen von Nutzern mit der Webseite generierte Daten und einige Male sogar Daten aus von uns entworfenen Experimenten.

Weil DataSciencester großen Wert auf die »Marke Eigenbau« legt, werden wir unsere Werkzeuge von Grund auf neu entwickeln. Am Ende werden Sie ein sehr solides Verständnis von den Data-Science-Grundlagen bekommen haben. Sie werden bereit sein, Ihre Fähigkeiten in einer Firma mit einer weniger heiklen Leitlinie einzusetzen oder auf eine für Sie interessante Fragestellung anzuwenden.

Willkommen an Bord und viel Glück! (Freitags dürfen Sie hier Jeans tragen, und die Toiletten sind am Ende des Korridors auf der rechten Seite.)

Finden von Schlüsselpersonen

Es ist Ihr erster Arbeitstag bei DataSciencester, und der Vizepräsident für Netzwerkarbeit steckt voller Fragen über Ihre Nutzer. Bisher hatte er niemanden, den er fragen konnte, und daher ist er begeistert, Sie dabeizuhaben.

Insbesondere möchte er herausbekommen, welche die »Schlüsselpersonen« unter den Data Scientists sind. Dazu stellt er Ihnen eine vollständige Kopie des Netzwerks von DataSciencester zur Verfügung. (Im wirklichen Leben übergibt man Ihnen die benötigten Daten eher selten. Kapitel 9 beschäftigt sich mit dem Beschaffen von Daten.)

Wie sieht diese Kopie der Netzwerkdaten aus? Sie besteht aus einer Liste von Nutzern, in der für jeden Nutzer ein `dict` mit einer `id` (eine Zahl) und einem Namen (`name`) angelegt ist. Aufgrund eines großen kosmischen Zufalls reimen sich die Namen mit der englisch ausgesprochenen `id` des Nutzers:

```

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" }
]

```

Der Vizepräsident gibt Ihnen auch noch Daten über »Freundschaften« als eine Liste von id-Paaren:

```

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)]

```

Zum Beispiel zeigt das Tupel (0, 1) an, dass der Data Scientist mit der id 0 (Hero) und der Data Scientist mit der id 1 (Dunn) befreundet sind. Das komplette Netzwerk ist in Abbildung 1-1 dargestellt.

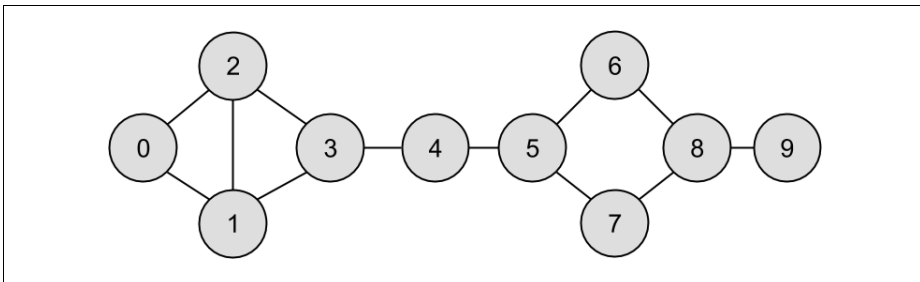


Abbildung 1-1: Das DataSciencester-Netzwerk

Wir haben die Freundschaften als Liste aus Paaren abgelegt. Damit lässt sich allerdings nicht sehr gut arbeiten. Um alle Freundschaften von Nutzer 1 zu finden, müssen Sie über jedes Paar iterieren und nach solchen mit der 1 suchen. Haben Sie viele Paare, würde das sehr lange dauern.

Lassen Sie uns stattdessen ein dict erstellen, in dem die Schlüssel die ids der Nutzer sind, während es sich bei den Werten um Listen mit ids der Freunde handelt. (Das Nachschlagen von Dingen in einem dict geht sehr schnell.)



Machen Sie sich bitte zunächst nicht zu viele Gedanken über die Details des Programmcodes. In Kapitel 2 werden Sie einen Crashkurs in Python absolvieren. Versuchen Sie erst einmal, einen Eindruck davon zu bekommen, was wir tun.

Wir müssen zwar immer noch jedes Paar anschauen, um das dict zu erstellen, aber nur einmal, und danach sind die Lookups sehr günstig:

```
# Das dict mit einer leeren Liste für jede Nutzer-id initialisieren:
friendships = {user["id"]: [] for user in users}

# Und über die Freundschaftspaare laufen lassen, um sie zu befüllen:
for i, j in friendship_pairs:
    friendships[i].append(j) # j als Freund von Nutzer i hinzufügen
    friendships[j].append(i) # i als Freund von Nutzer j hinzufügen
```

Nachdem wir jetzt die Freundschaften in einem dict abgelegt haben, können wir leicht Fragen an unseren Graphen formulieren, etwa: »Was ist die durchschnittliche Anzahl von Verbindungen eines Nutzers?«

Als Erstes ermitteln wir die *gesamte* Anzahl an Verbindungen, indem wir die Längen sämtlicher Listen in friends aufsummieren:

```
def number_of_friends(user):
    """Wie viele Freunde hat _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
```

Anschließend teilen wir einfach durch die Anzahl der Nutzer:

```
num_users = len(users) # Länge der Liste users
avg_connections = total_connections / num_users # 24/10 == 2.4
```

Es ist ebenfalls leicht, die am besten vernetzten Personen zu finden – sie sind diejenigen mit der größten Anzahl Freunde.

Weil es nicht sehr viele Nutzer gibt, können wir sie einfach von »hat die meisten Freunde« bis »hat die wenigsten Freunde« sortieren:

```
# Erstelle eine Liste (user_id, number_of_friends)
num_friends_by_id = [(user["id"], number_of_friends(user))
                    for user in users]

num_friends_by_id.sort( # nach num_friends
    key=lambda id_and_friends: id_and_friends[1], # sortieren
    reverse=True) # (absteigend)

# Jedes Paar entspricht (user_id, num_friends)
# [(1, 3), (2, 3), (3, 3), (5, 3), (8, 3),
# (0, 2), (4, 2), (6, 2), (7, 2), (9, 1)]
```

Wir können die obige Prozedur anwenden, um die zentralen Personen im Netzwerk zu identifizieren. Tatsächlich nennt man das von uns berechnete Maß für die Zentralität des Netzwerks *Grad* (siehe Abbildung 1-2).

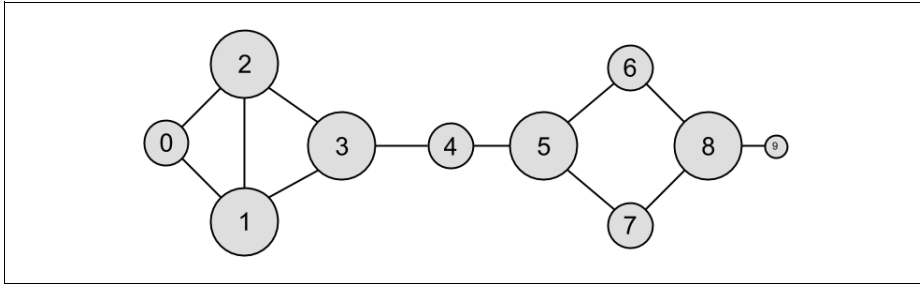


Abbildung 1-2: Das Netzwerk von DataSciencester, aufgeschlüsselt nach Grad

Dieses Maß hat den Charme, leicht berechenbar zu sein, aber es liefert nicht immer das Ergebnis, das Sie möchten oder erwarten. Im Netzwerk von DataSciencester zum Beispiel hat Thor (id 4) nur zwei Verbindungen, Dunn (id 1) dagegen besitzt drei. Aber schaut man sich einmal das Netzwerk an, sieht es eigentlich so aus, als wäre Thor zentraler. In Kapitel 22 werden wir Netzwerke genauer untersuchen, und wir werden uns auch kompliziertere Maße für Zentralität ansehen, die teilweise mit unserer Intuition übereinstimmen, teilweise aber auch nicht.

Data Scientists, die Sie kennen könnten

Während Sie noch den Papierkram für neue Mitarbeiter ausfüllen, kommt die Vizepräsidentin für Seilschaften an Ihrem Schreibtisch vorbei. Sie möchte Ihre Mitglieder zu mehr Verbindungen ermuntern und bittet Sie, ein Empfehlungssystem für »Data Scientists, die Sie kennen könnten« zu entwerfen.

Ihr erster Einfall ist, Nutzern die Freunde ihrer Freunde vorzuschlagen. Also schreiben Sie ein paar Codezeilen, um über die Freunde zu iterieren und deren Freunde einzusammeln:

```
def foaf_ids_bad(user):
    # "foaf" steht für "friend of a friend"
    return [foaf_id
            for friend_id in friendships[user["id"]]
            for foaf_id in friendships[friend_id]]
```

Rufen wir diese Prozedur mit `users[0]` (Hero) auf, erhalten wir:

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

Das Ergebnis enthält zweimal Nutzer 0, da Hero in der Tat mit beiden seiner Freunde befreundet ist. Es enthält auch Nutzer 1 und 2, obwohl beide bereits mit Hero befreundet sind. Und es enthält Nutzer 3 doppelt, weil Chi über zwei unterschiedliche Freunde erreicht werden kann:

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

Zu wissen, dass Personen auf vielerlei Art und Weise Freunde von Freunden sein können, scheint eine interessante Information zu sein. Vielleicht sollten wir stattdessen die *Anzahl* gemeinsamer Freunde berechnen. Und wir sollten vielleicht dem Nutzer bekannte Personen ausschließen:

```

from collections import Counter                                # wird nicht automatisch
                                                            # geladen

def friends_of_friends(user):
    user_id = user["id"]
    return Counter(
        foaf_id
        for friend_id in friendships[user_id]                # für jeden meiner Freunde
        for foaf_id in friendships[friend_id]                # zähle *deren* Freunde,
        if foaf_id != user_id                                # die nicht ich sind und
        and foaf_id not in friendships[user_id]              # noch nicht meine Freunde
    )

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

```

Dies sagt uns korrekterweise, dass Chi (id 3) zwei gemeinsame Freunde mit Hero (id 0), aber nur einen gemeinsamen Freund mit Clive (id 5) hat.

Als Data Scientist wissen Sie, dass Sie gern Menschen mit ähnlichen Interessen kennenlernen würden. (Dies ist ein gutes Beispiel für den Aspekt »substanzielles Expertenwissen« in Data Science.) Nach etwas Herumfragerei bekommen Sie die Interessen der Nutzer als Liste von Paaren (user_id, interest) in die Finger:

```

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")
]

```

Hero (id 0) hat zum Beispiel keine gemeinsamen Freunde mit Klein (id 9), aber beide interessieren sich für Java und Big Data.

Eine Funktion zum Finden von Nutzern mit einem bestimmten Interessengebiet ist schnell geschrieben:

```

def data_scientists_who_like(target_interest):
    """Finde die ids aller Nutzer, die das Ziel interessiert."""
    return [user_id

```

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

Das funktioniert zwar, aber bei jeder Suche muss die gesamte Interessenliste abgearbeitet werden. Wenn wir eine Menge Nutzer und Interessen haben (oder einfach viele Suchen durchführen möchten), sind wir mit einer Indexstruktur von Interessen zu Nutzern besser dran ...

```
from collections import defaultdict

# Die Interessen sind Schlüssel, die Werte sind Listen von user_ids mit diesem
# Interesse
user_ids_by_interest = defaultdict(list)

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

... sowie einem zweiten Index von Nutzern zu Interessen:

```
# Die Schlüssel sind user_ids, die Werte sind Listen von Interessen dieser user_id
interests_by_user_id = defaultdict(list)

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

Nun können wir leicht herausbekommen, wer mit einem bestimmten Nutzer die meisten gemeinsamen Interessen hat:

- Iteriere über die Interessen eines Nutzers.
- Iteriere für jedes Interessengebiet über die anderen Nutzer mit dem gleichen Interesse.
- Zähle mit, wie oft wir jedem der anderen Nutzer begegnen.

Als Code:

```
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"]
    )
```

Wir könnten ein umfangreicheres Feature »Data Scientists, die Sie kennen könnten« entwickeln, indem wir gemeinsame Freunde und gemeinsame Interessen kombinieren. Wir werden diese Art von Anwendung in Kapitel 23 untersuchen.

Gehälter und Erfahrung

Als Sie gerade zur Mittagspause aufbrechen möchten, fragt Sie der Vizepräsident für Öffentlichkeitsarbeit, ob Sie kernige Fakten über das Einkommen von Data Scientists finden könnten. Natürlich sind Daten über das Gehalt vertraulich, aber

er gibt Ihnen einen anonymisierten Datensatz mit dem Gehalt in Dollar (salary) und der Dienstzeit als Data Scientist in Jahren (tenure) für jeden Nutzer:

```
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)]
```

Der natürliche erste Schritt ist, diese Daten zu plotten (wie das geht, erfahren wir in Kapitel 3). Sie können das Resultat in Abbildung 1-3 sehen.

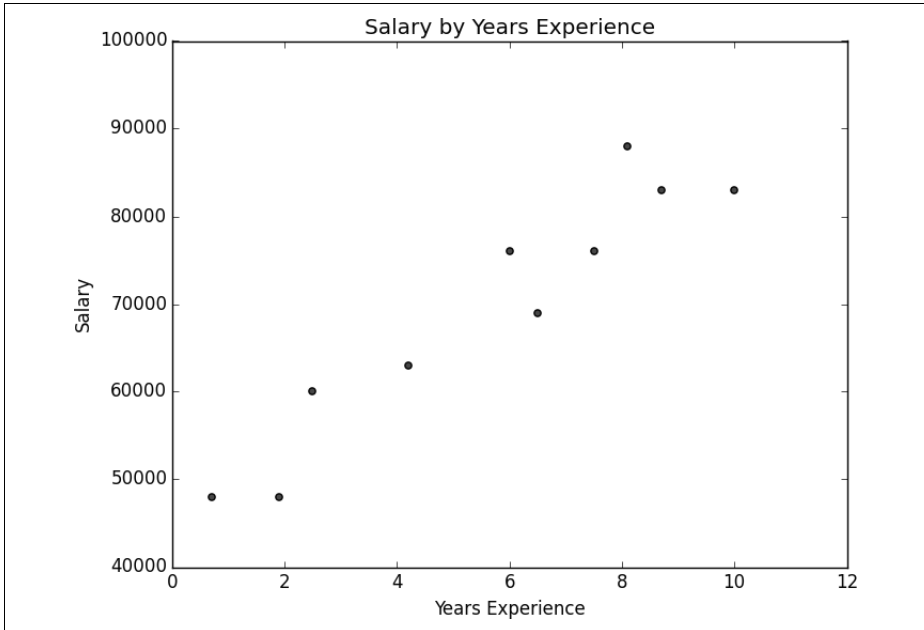


Abbildung 1-3: Gehalt nach Dienstzeit in Jahren

Es scheint recht deutlich, dass Leute mit mehr Erfahrung tendenziell mehr verdienen. Wie könnten Sie diese Erkenntnis interessant verpacken? Ihre erste Idee ist, das durchschnittliche Gehalt für jede Dienstzeit zu betrachten:

```
# Schlüssel sind Jahre, Werte sind Listen von Gehältern für jede Dienstzeit  
salary_by_tenure = defaultdict(list)  
  
for salary, tenure in salaries_and_tenures:  
    salary_by_tenure[tenure].append(salary)  
  
# Schlüssel sind Jahre, Werte sind die Durchschnittsgehälter für diese Dienstzeit  
average_salary_by_tenure = {  
    tenure : sum(salaries) / len(salaries)  
    for tenure, salaries in salary_by_tenure.items()  
}
```

Es stellt sich heraus, dass dies nicht besonders nützlich ist, denn keine Nutzer haben gleiche Dienstzeiten. Damit würden wir lediglich die Gehälter einzelner Nutzer wiedergeben:

```
{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}
```

Hier hilft es, die Dienstzeit in Klassen einzuteilen:

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

Damit lassen sich die Gehälter für jede Klasse gruppieren:

```
# Schlüssel sind die Klassen nach Dienstzeit, Werte sind Listen der Gehälter für  
# diese Klasse  
salary_by_tenure_bucket = defaultdict(list)  
  
for salary, tenure in salaries_and_tenures:  
    bucket = tenure_bucket(tenure)  
    salary_by_tenure_bucket[bucket].append(salary)
```

Und schließlich kann man für jede Gruppe das Durchschnittsgehalt berechnen:

```
# Schlüssel sind die Klassen nach Dienstzeit, Werte das Durchschnittsgehalt für  
# diese Klasse  
average_salary_by_bucket = {  
    tenure_bucket : sum(salaries) / len(salaries)  
    for tenure_bucket, salaries in salary_by_tenure_bucket.items()  
}
```

Nun ist das Ergebnis interessanter:

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

Und damit haben Sie Ihre knackige Aussage: »Data Scientists mit mehr als fünf Jahren Erfahrung verdienen 65 % mehr als Data Scientists mit weniger oder keiner Erfahrung!«

Allerdings haben wir unsere Klassen recht willkürlich ausgewählt. Was wir wirklich bräuchten, ist eine Aussage über die – durchschnittlichen – Auswirkungen

eines zusätzlichen Jahres an Erfahrung auf das Gehalt. Das macht die Aussage noch schwungvoller und erlaubt außerdem, *Vorhersagen* zu uns unbekanntem Gehältern zu treffen. Wir werden uns mit dieser Idee in Kapitel 14 beschäftigen.

Bezahlte Nutzerkonten

Als Sie zu Ihrem Schreibtisch zurückkehren, wartet die Vizepräsidentin für Einkünfte bereits auf Sie. Sie möchte besser verstehen, welche Nutzer für ihre Mitgliedschaft bezahlen und welche nicht. (Sie kennt deren Namen, das allein ist aber keine praktisch umsetzbare Information.)

Ihnen fällt schnell auf, dass die bezahlten Nutzerkonten anscheinend mit der Erfahrung in Jahren zusammenhängen:

```
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
```

Nutzer mit sehr wenigen und sehr vielen Jahren Erfahrung neigen zum Bezahlen, Nutzer mit durchschnittlicher Erfahrungsdauer nicht.

Wenn Sie hierfür ein Modell entwickeln möchten – auch wenn es definitiv noch nicht genug Daten für ein Modell sind –, könnten Sie versuchen, für Nutzer mit sehr wenigen und sehr vielen Jahren Erfahrung »bezahlt« vorherzusagen, hingegen »nicht bezahlt« für Nutzer mit mittlerer Erfahrung:

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

Die Grenzwerte sind natürlich völlig aus der Luft gegriffen.

Mit mehr Daten (und mehr Mathematik) könnten wir ein Modell konstruieren, das aus der Erfahrung eines Nutzers in Jahren die Wahrscheinlichkeit berechnet, mit der dieser bezahlt. Diese Art von Aufgabe werden wir in Kapitel 16 untersuchen.

Interessante Themen

Während Sie Ihren ersten Tag Revue passieren lassen, fragt Sie die Vizepräsidentin für Content-Strategie, an welchen Themen Nutzer am ehesten interessiert sind,

damit sie ihre Blogbeiträge dementsprechend planen können. Sie verfügen bereits über die Rohdaten aus dem Projekt zur Empfehlung von Freunden:

```
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")  
]
```

Eine einfache (wenn auch nicht besonders aufregende) Methode zum Ermitteln der beliebtesten Interessengebiete ist, die Wörter auszuzählen:

1. Verwandle jedes Interessengebiet in Kleinbuchstaben (da bei den Interessen nicht alle Nutzer Großbuchstaben in gleicher Weise verwenden).
2. Teile die Interessengebiete in einzelne Wörter auf.
3. Zähle das Ergebnis aus.

Oder als Programmcode geschrieben:

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

Damit können wir mehr als einmal auftretende Wörter leicht auflisten:

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

Dies ergibt das gewünschte Ergebnis (es sei denn, Sie erwarten, dass »scikit-learn« in zwei Wörter aufgeteilt wird – dann wäre das Ergebnis natürlich nicht das gewünschte):

```
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

Wir werden in Kapitel 21 anspruchsvollere Möglichkeiten der Datengewinnung betrachten.

Weiter geht's!

Es war ein erfolgreicher erster Tag! Erschöpft huschen Sie aus dem Gebäude, bevor Sie noch irgendjemand um etwas bitten kann. Schlafen Sie sich gut aus, denn morgen ist der Orientierungstag für neue Mitarbeiter. (Ja, Sie haben bereits einen ganzen Tag *vor* dem Orientierungstag gearbeitet. Beschweren Sie sich bei der Personalabteilung.)