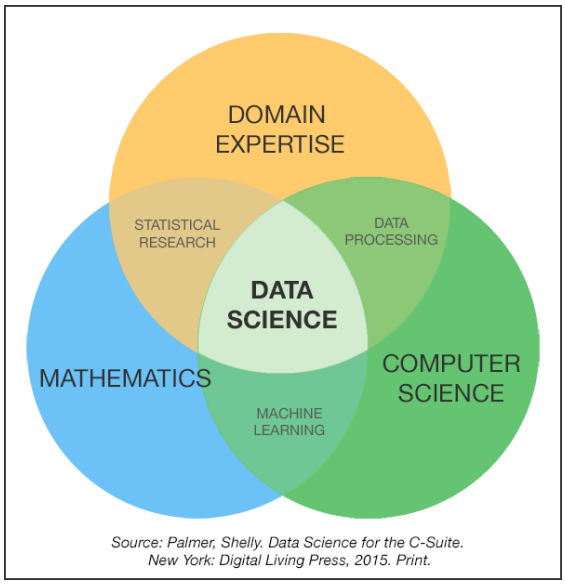
Data Science

Data science é um campo em evidência e está em alta; não requer muita investigação para encontrar prognósticos de analistas de que, nos próximos dez anos, precisaremos de bilhões e bilhões de cienti dados a mais do que possuímos atualmente.

Mas o que é data science?

Segundo um famoso diagrama de Venn, data science se encontra na interseção de:

- Habilidades de Engenharia de software
- Conhecimento de estatística e matemática
- Domínio de negócio



Metodologia de ensino

Existem muitas bibliotecas, estruturas, módulos e kits de ferramentas de data science que implemer modo eficiente os mais comuns (e também os menos comuns) algoritmos e técnicas. Se você se torna cientista de dados, será íntimo de **NumPy, scikit-learn, pandas** e de diversas outras bibliotecas. Elas ótimas para praticar data science e também ótimas para começar a **praticar sem entender de fato o data science**.

Nesta disciplina construiremos ferramentas e implementaremos algoritmos à mão, a fim de entendê-lo melhor.

Python é a escolha evidente quando o assunto é linguagem de programação pois possui diversos recu que o tornam mais adequado para o aprendizado (e prática) de data science:

- É gratuito.
- É relativamente simples de codificar (e, o principal, de entender).
- Possui muitas bibliotecas úteis relacionadas ao data science.

A Ascensão dos Dados

Vivemos em um mundo que está soterrado por dados. Os websites rastreiam todos os cliques de todo usuários. Seu smartphone está fazendo um registro da sua localização e sua velocidade a cada segun diariamente. Atletas avaliados usam pedômetros com esteroides que estão sempre registrando suas b do coração, hábitos de movimentos, dieta e padrões do sono. Carros inteligentes coletam hábitos de casas inteligentes coletam hábitos de moradia e marqueteiros inteligentes coletam hábitos de compra

Soterrados sob esses dados estão as respostas para as inúmeras questões que ninguém nunca em perguntar.

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. Digamos que um cientista de dados seja alguém que extrai conhecimento de dados desorganizados. O mundo de hoje cheio de pessoas tentando transformar dados em conhecimento.

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

Como uma grande empresa, a Target rastreia suas encomendas e interações, tanto online como na loj Ela usa os dados em um modelo preditivo (http://nyti.ms/1EQTznL) (http://nyti.ms/1EQTznL) para sab 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 vo onde provavelmente eles teriam sido úteis. Normalmente,

DataSciencester - Rede Social de Cientistas de Dados

Considere um cenário hipotético no qual você é um cientista de dados recém-contratado para trabalha uma rede social destinada a cientistas de dados.

Seu primeiro trabalho é identificar quem são os “conectores-chave” entre os cientistas de dados. Para ele lhe dá uma parte de toda a rede da DataSciencester. Na vida real, você geralmente não recebe os de que precisa.

In [1]:

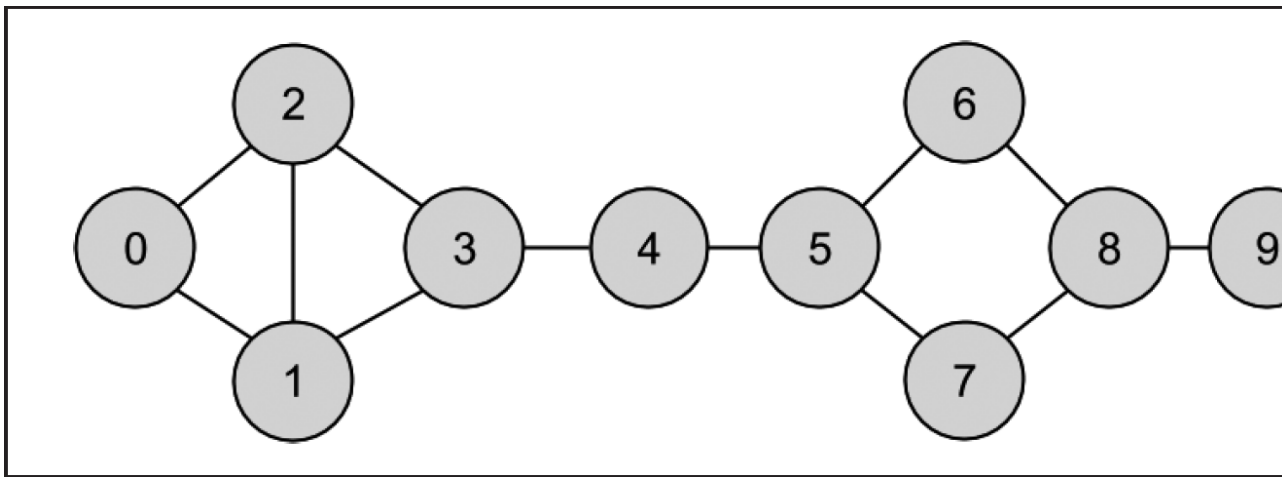
```
users = [
{ "id": 0, "name": "Hero" },
{ "id": 1, "name": "Dunn" },
{ "id": 2, "name": "Sue" },
{ "id": 3, "name": "Chi" },
{ "id": 4, "name": "Thor" },
{ "id": 5, "name": "Clive" },
{ "id": 6, "name": "Hicks" },
{ "id": 7, "name": "Devin" },
{ "id": 8, "name": "Kate" },
{ "id": 9, "name": "Klein"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

Você também recebe dados sobre os vínculos de amizades, representado por uma lista de pares (tupl
ID's:

In [2]:

```
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 con
(Dunn) são amigos. A rede é ilustrada na figura abaixo:



Já que representamos nossos usuários como dicts, é fácil de aumentá-los com dados extras.
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:

In [3]:

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

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

In [4]:

```
for i, j in friendships:
    users[i]["friends"].append(users[j])
    users[j]["friends"].append(users[i])
```

In [5]:

```
sum(len(user["friends"]) for user in users)
```

Out[5]:

```
24
```

Uma vez que o dict de cada usuário contenha uma lista de amigos, podemos facilmente perguntar sob
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 fri

In [6]:

```
def number_of_friends(user):
    return len(user["friends"])
```

In [7]:

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

Então, apenas dividimos pelo número de usuários:

In [8]:

```
num_users = len(users)
avg_connections = total_connections / num_users
avg_connections
```

Out[8]:

```
2.4
```

Também é fácil de encontrar as pessoas mais conectadas — são as que possuem o maior número de
Como não há muitos usuários, podemos ordená-los de “muito amigos” para “menos amigos”:

In [34]:

```
num_friends_by_id = [(user["id"], number_of_friends(user)) for user in users
```

In [35]:

```
num_friends_by_id
```

Out[35]:

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

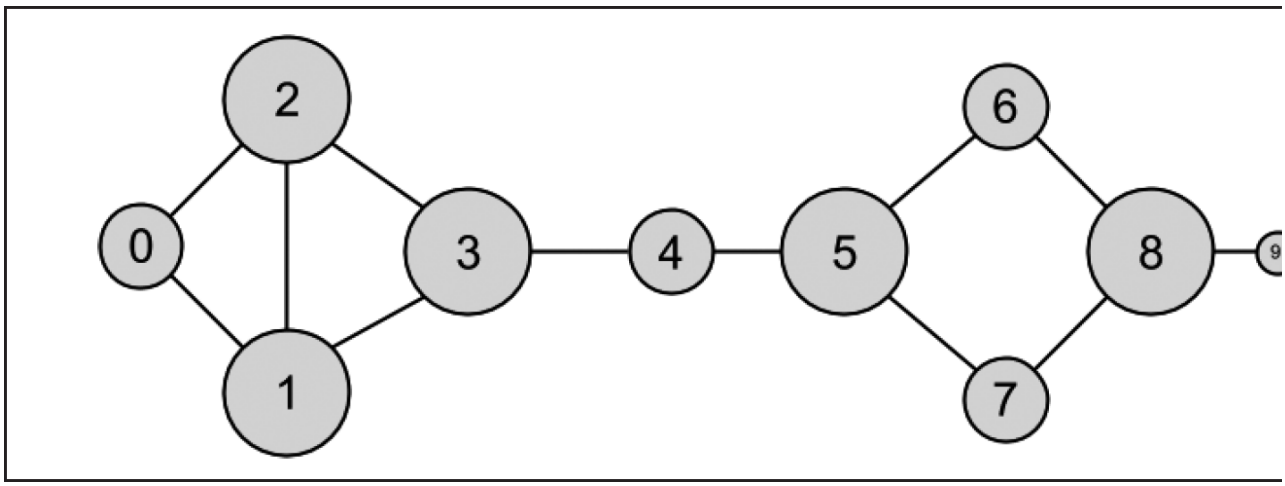
In [10]:

```
sorted(num_friends_by_id, key=lambda x: x[1], reverse=True)
# 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)]
```

Out[10]:

```
[(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 forma, centrais para a rede. Na verdade, o que acabamos de computar é uma rede métrica de grau de centralidade (figura abaixo).



Job 2: Cientistas de Dados que talvez você conheça!

Sua próxima missão é estimular mais conexões entre os membros da rede social, assim, você precisa desenvolver 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 comp para cada amigo de um usuário, itera sobre os amigos daquela pessoa, e coleta todos os resultados:

In [11]:

```
def friends_of_friend_ids_bad(user):
    # "foaf" is short for "friend of a friend"
    return [foaf["id"]
        for friend in user["friends"] # for each of user's friends
        for foaf in friend["friends"]] # get each of _their_ friends
```

In [37]:

```
users[5]["friends"]
```

Out[37]:

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

```
    {...},
```

In [13]:

```
friends_of_friend_ids_bad(users[0])
```

Out[13]:

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

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. Inc
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]

Definitivamente, devemos usar uma função de ajuda para excluir as pessoas que já são conhecidas d
usuário:

In [14]:

```
from collections import Counter
```

In [15]:

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

In [16]:

```
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"
```

In [17]:

```
from collections import Counter
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"] # para cada amigo de um dos meus amigo
        if not_the_same(user, foaf) # que não sejam eu
        and not_friends(user, foaf)) # e que não são meus amigos
```

In [18]:

```
friends_of_friend_ids(users[0])
```

Out[18]:

```
Counter({3: 2})
```

In [19]:

```
print(friends_of_friend_ids(users[3]))
```

```
Counter({0: 2, 5: 1})
```

Job 3: Interesses em comum

Como um cientista de dados, você sabe que você pode gostar de encontrar usuários com interesses s
(esse é um bom exemplo do aspecto “competência significativa” de data science). Depois de pergunta
você consegue pôr as mãos nesse dado, como uma lista de pares (user_id, interest):

In [20]:

```
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 interes
aprendizado de máquina.
É fácil construir uma função que encontre usuários com o mesmo interesse:

In [21]:

```
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 u
e interesses (ou se quisermos fazer muitas buscas), seria melhor construir um índice de interesses par
usuários:

In [22]:

```
from collections import defaultdict
```

In [23]:

```
# 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:

In [24]:

```
# as chaves são user_ids, os valores são as listas de interests para aquele
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.

In [25]:

```
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"])
```

In [26]:

```
'''print(interests_by_user_id)'''
print(most_common_interests_with(users[0]))
```

```
Counter({9: 3, 1: 2, 8: 1, 5: 1})
```

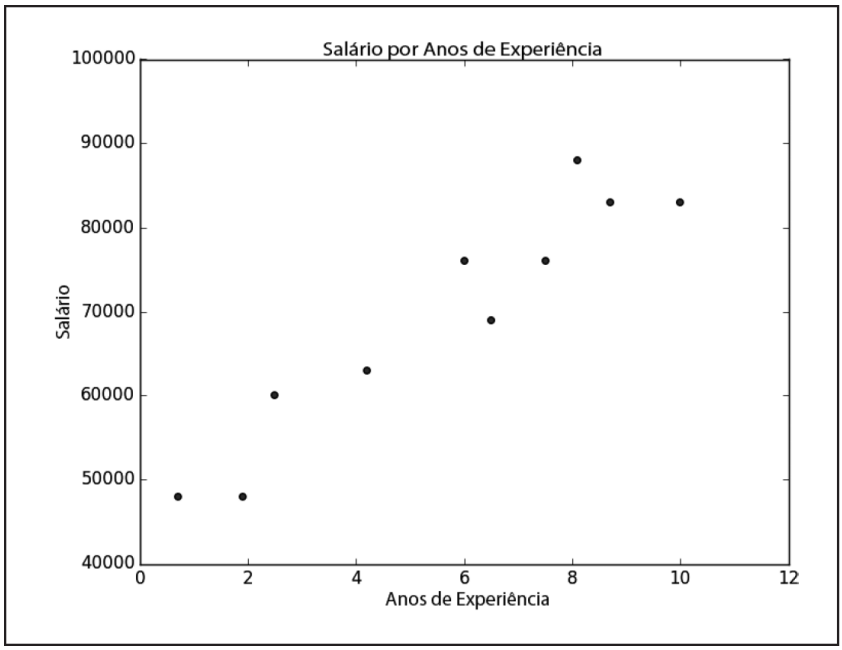
Job 4: Salários e Experiência

Apesar de dados de salário ser, de fato, um tópico sensível,você recebe um conjunto de dados anônin
contendo o salary (salário) de cada usuário (em dólares) e tenure (experiência) de cientista de dados (e

In [27]:

```
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, como cientista de dados, o primeiro passo é plotar os dados (veremos como fazê-lo a f
Os resultados se encontram na figura abaixo:



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

In [28]:

```
# 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 repo
apenas os salários individuais dos usuários:

In [29]:

```
{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}
```

Out[29]:

```
{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:

In [30]:

```
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:

In [31]:

```
# 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:

In [32]:

```
# 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.items()
}
```

que é mais interessante:

In [33]:

```
average_salary_by_bucket
```

Out[33]:

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

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

Job 5: Contas Premium (pagas)

O próximo job consiste em entender melhor quais são os usuários que pagam por contas e quais que
pagam (sabemos seus nomes, mas essa informação não é essencial).
Você percebe que parece haver uma correspondência entre os anos de experiência e as contas pagas

In [ ]:

```
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 quantid
mediana de experiência não. Logo, se você quisesse criar um modelo — apesar de não haver dados c
suficiente para servir de base para um — você talvez tentasse prever “paid” para os usuários com pou
muitos anos de experiência, e “unpaid” para os usuários com quantidade mediana de experiência:

In [64]:

```
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 um usuário pagaria, baseado em seus anos de experiência. Investigaremos esse tipo de problema ma frente.

Job 6: Tópicos de Interesse

Quando seu dia está terminando, a vice-presidente da Estratégia de Conteúdo pede dados sobre em tópicos os usuários estão mais interessados, para que ela possa planejar o calendário do seu blog de Você já possui os dados brutos para o projeto sugerido, estou falando da lista "interests" que definimo

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

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

In [65]:

```
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:

In [66]:

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

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

In [67]:

```
c = Counter(['a', 'a', 'a', 'a', 'b', 'b'])
c
```

Out[67]:

```
Counter({'a': 4, 'b': 2})
```

In [68]:

```
teste = friends_of_friend_ids(users[3])
teste
```

Out[68]:

```
Counter({0: 2, 5: 1})
```

In [ ]: