Entendendo o Big Data

(baseado em Guller (2015) - Big Data Analytics with Spark A Practitioner's Guide to Using Spark for Large-Scale Data Processing, Machine Learning, and Graph Analytics, and High-Velocity Data Stream Processing)

Falando sobre o Cenário de Big Data - O que é Serialização de Dados

Os dados têm seu próprio ciclo de vida, independente do programa que os cria ou consome. Na maioria das vezes, os dados sobrevivem ao aplicativo que os criou.

Geralmente, são salvos no disco. Às vezes, ele são enviado de um aplicativo para outro aplicativo em uma rede.

O formato no qual os dados são armazenados no disco ou enviados pela rede é diferente do formato em que vivem na memória.

O processo de conversão de dados na memória em um formato no qual possam ser armazenados em disco ou enviados pela rede é chamado de **serialização**. O processo reverso de leitura de dados do disco ou rede para a memória é chamado de desserialização.

Falando sobre o Cenário de Big Data - Como é a Serialização de Dados

Os dados podem ser serializados usando muitos formatos diferentes. Os exemplos incluem CSV, XML, JSON e vários formatos binários. Cada formato tem prós e contras.

Por exemplo, formatos de texto como CSV, XML e JSON são legíveis por humanos, mas não são eficientes em termos de espaço de armazenamento ou tempo de análise.

Por outro lado, os formatos binários são mais compactos e podem ser analisados muito mais rapidamente do que os formatos de texto. No entanto, os formatos binários não são legíveis por humanos.

Falando sobre o Cenário de Big Data - O Tempo de Serialização de Dados

O tempo de serialização/desserialização ou a diferença de espaço de armazenamento entre os formatos de texto e binários não é um grande problema quando um conjunto de dados é pequeno.

Portanto, as pessoas geralmente preferem formatos de texto para pequenos conjuntos de dados, pois são mais fáceis de gerenciar.

No entanto, para grandes conjuntos de dados, o tempo de serialização/desserialização ou a diferença de espaço de armazenamento entre os formatos de texto e binários são significativos.

Portanto, os formatos binários são geralmente preferidos para armazenar grandes conjuntos de dados.

Vamos descrever alguns dos formatos binários comumente usados para serializar big data.

Avro

Avro fornece um formato binário compacto independente de linguagem para serialização de dados. Ele pode ser usado para armazenar dados em um arquivo ou enviá-los pela rede. Ele oferece suporte a estruturas de dados ricas, incluindo dados aninhados.

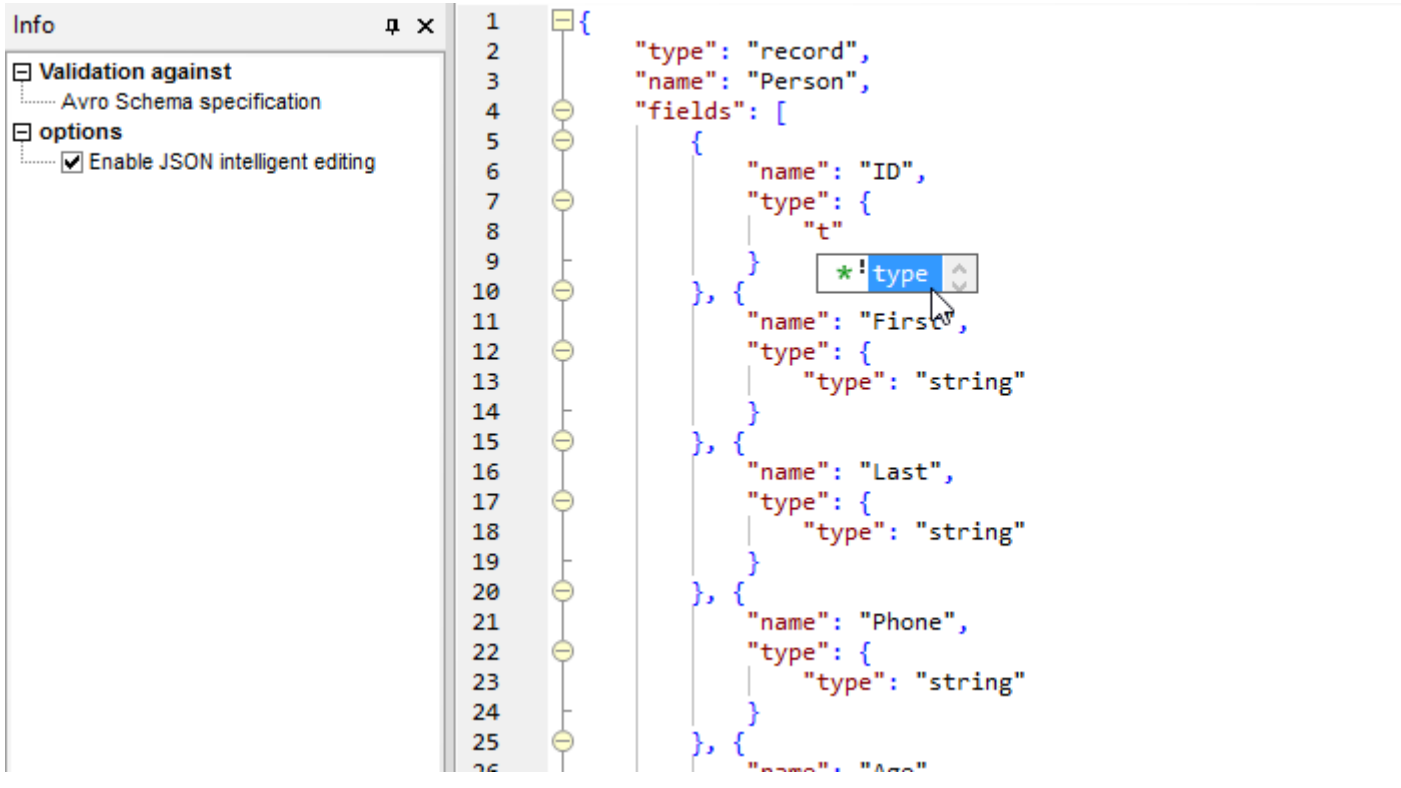
Avro usa um formato binário autoexplicativo. Quando os dados são serializados usando Avro, o esquema é armazenado junto com os dados.

Portanto, um arquivo Avro pode ser lido posteriormente por qualquer aplicativo. Além disso, como o esquema é armazenado junto com os dados, cada datum é gravado sem sobrecargas por valor, tornando a serialização rápida e compacta.

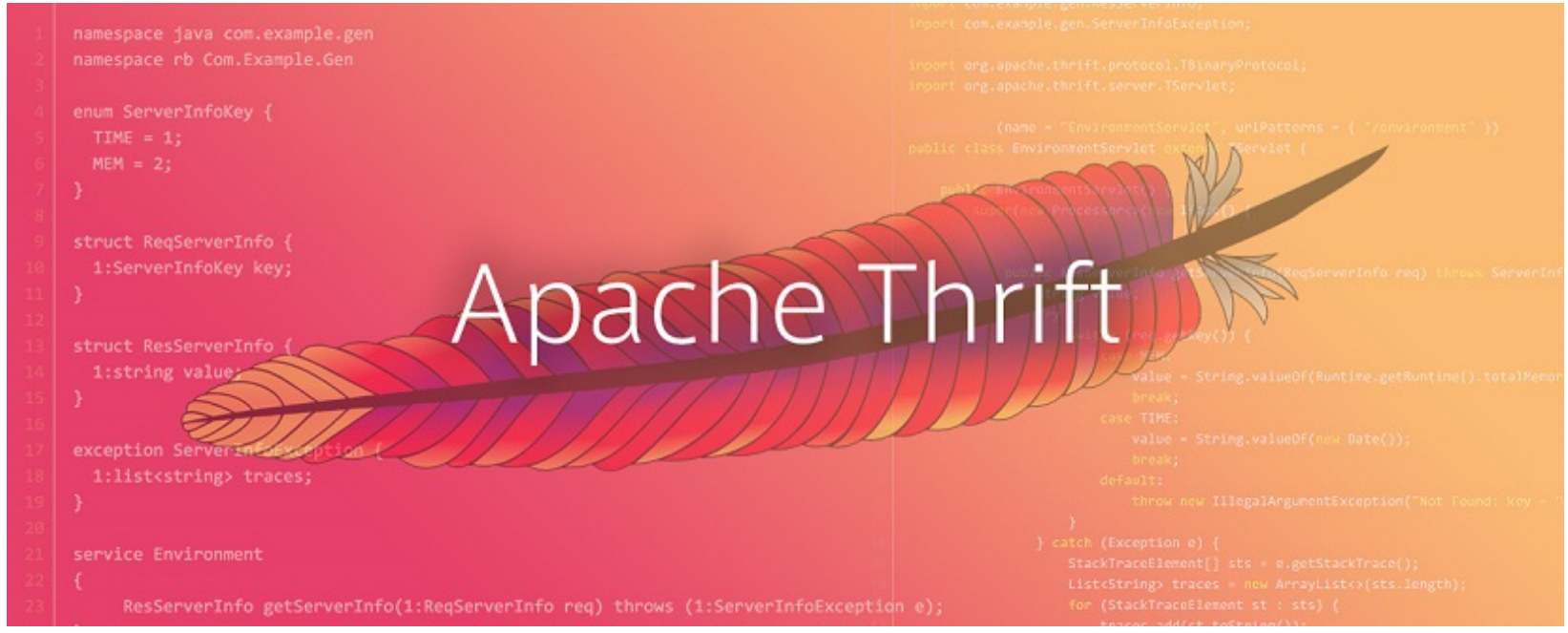
Quando os dados são trocados em uma rede usando Avro, o emissor e o receptor trocam esquemas durante um handshake de conexão inicial.

Avro

Um esquema Avro é descrito usando JSON [Leia mais sobre JSON aqui (https://pt.wikipedia.org/wiki/JSON)](https://pt.wikipedia.org/wiki/JSON).



Avro lida automaticamente com adição e remoção de campo e compatibilidade com versões anteriores e posteriores - tudo sem qualquer reconhecimento por parte de um aplicativo.



Thrift

Thrift é uma estrutura de serialização de dados independente de linguagem.

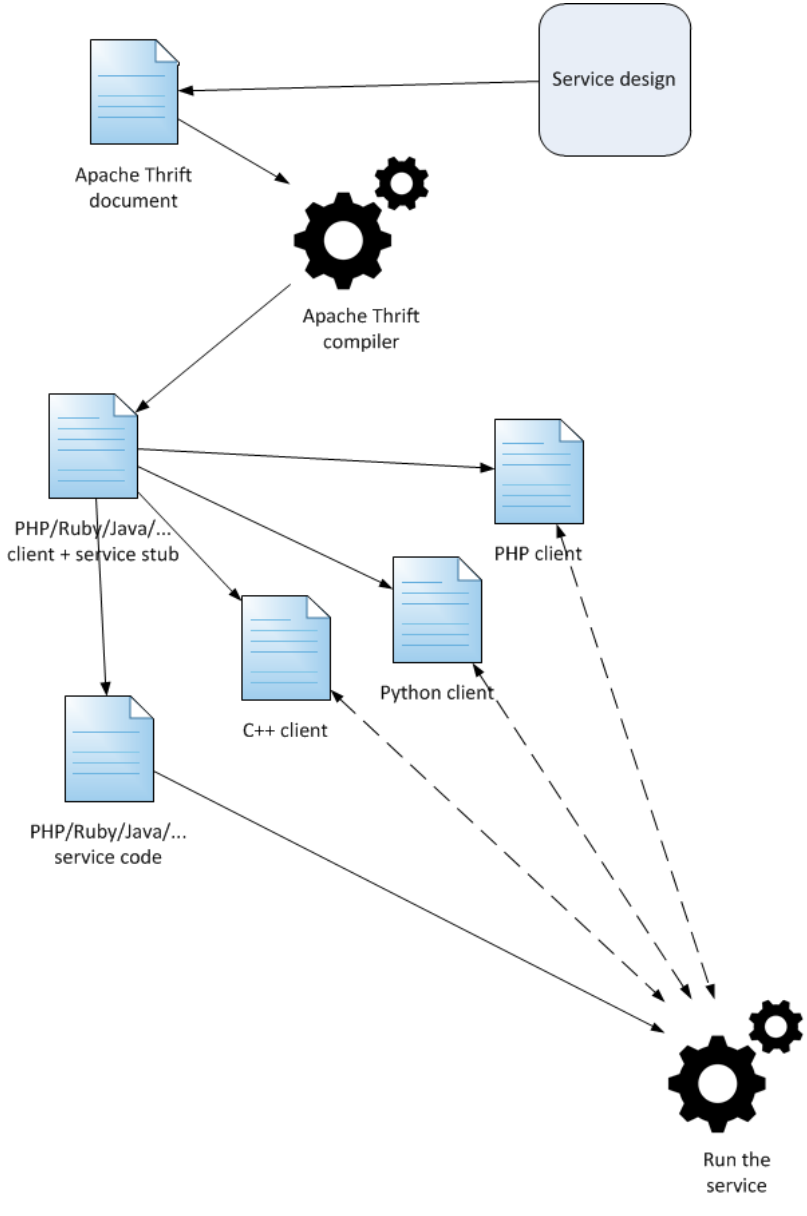
Ele fornece principalmente ferramentas para serializar a troca de dados em uma rede entre aplicativos escritos em diferentes linguagens de programação.

Suporta uma variedade de linguagens, incluindo C ++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C #, Cocoa, JavaScript, Node.js, Smalltalk, OCaml, Delphi e outras linguagens.

Thrift fornece uma ferramenta de geração de código e um conjunto de bibliotecas para serializar dados e transmiti-los através de uma rede.

Thrift abstrai o mecanismo para serializar dados e transportá-los através de uma rede. Assim, permite que um desenvolvedor de aplicativos se concentre na lógica do aplicativo central, em vez de se preocupar com a serialização de dados e a transmissão confiável e eficiente pela rede.

Um pouco mais sobre o Thrift



Com Thrift, um desenvolvedor de aplicativos define tipos de dados e interface de serviço em um arquivo de definição de interface com neutralidade de idioma.

Os serviços definidos em um arquivo de definição de interface são fornecidos por um aplicativo de servidor e usados por um aplicativo cliente.

O compilador Thrift compila esse arquivo e gera código que um desenvolvedor pode usar para construir rapidamente aplicativos cliente e servidor.

Um servidor e um cliente baseados em Thrift podem ser executados no mesmo computador ou em computadores diferentes em uma rede. Da mesma forma, o servidor e o aplicativo cliente podem ser desenvolvidos usando a mesma linguagem de programação ou diferentes linguagens de programação.

Veja um tutorila rápidp de Thrift neste link (http://www.thrift.pl/)

Protocol Buffers (Buffers de protocolo)

Protocol Buffers é uma estrutura de serialização de dados de código aberto desenvolvida pelo Google. Assim como Thrift e Avro, é neutro em termos de linguagem. O Google usa internamente os buffers de protocolo como formato de arquivo principal. Ele também o usa internamente para trocar dados entre aplicativos em uma rede.

Protocol Buffers é semelhante ao Thrift.

Ele fornece um compilador e um conjunto de bibliotecas que um desenvolvedor pode usar para serializar dados. Um desenvolvedor define a estrutura ou esquema de um conjunto de dados em um arquivo e o compila com o compilador de buffers de protocolo, que gera o código que pode ser usado para ler ou gravar facilmente esses dados.

Comparados ao Thrift, os Buffers de protocolo oferecem suporte a um conjunto menor de linguagens. Atualmente, ele oferece suporte a C ++, Java e Python.

Veja um tutorila do protocol Buffers neste link (https://www.freecodecamp.org/news/googles-protocol-buffers-in-python/)

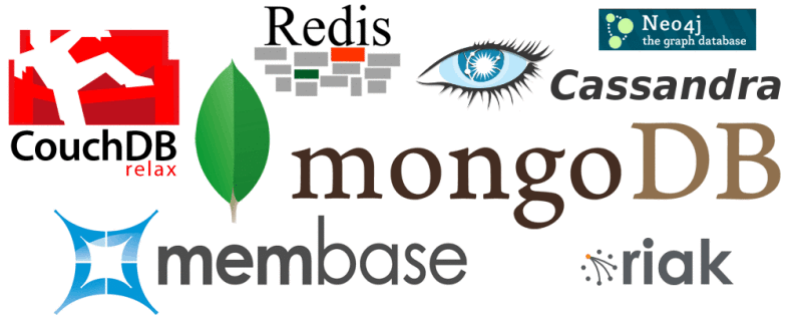
SequenceFile

SequenceFile é um formato de arquivo simples binário para armazenar pares de valores-chave. É comumente usado no Hadoop como formato de arquivo de entrada e saída.

MapReduce também usa SequenceFiles para armazenar a saída temporária de funções de mapa. Um SequenceFile pode ter três formatos diferentes: Uncompressed, Record Compressed e Block Compressed.

Em um Record Compressed SequenceFile, apenas o valor em um registro é compactado; enquanto em um Block Compressed SequenceFile, as chaves e os valores são compactados.

Veja um pouco mais neste link (https://cwiki.apache.org/confluence/display/HADOOP2/SequenceFile)



Falando sobre o Cenário de Big Data - NoSQL

O termo NoSQL é usado para uma ampla categoria de bancos de dados modernos não relacionais.

Inicialmente, NoSQL significava “Sem suporte a SQL”, uma vez que esses bancos de dados não suportavam SQL.

No entanto, agora significa “Não apenas SQL”, já que alguns desses bancos de dados suportam um subconjunto de comandos SQL.

Os bancos de dados NoSQL têm objetivos de design diferentes dos bancos de dados RDBMS.

Um banco de dados relacional garante ACID (Atomicidade, Consistência, Isolamento, Durabilidade).

Um banco de dados NoSQL troca a conformidade com ACID por escalabilidade linear, desempenho, alta disponibilidade, esquema flexível e outros recursos.

Vamos discutir alguns dos bancos de dados NoSQL comumente usados.

Cassandra

Cassandra é um banco de dados NoSQL distribuído, escalonável e tolerante a falhas, projetado para armazenar grandes conjuntos de dados.

É um armazenamento de linha particionado com consistência ajustável.

Um de seus principais recursos é o esquema dinâmico. Cada linha pode armazenar colunas diferentes, ao contrário de bancos de dados relacionais, onde cada linha tem exatamente as mesmas colunas. Além disso, o Cassandra é otimizado para gravações, portanto, as inserções têm alto desempenho.

Cassandra tem uma arquitetura distribuída masterless. Portanto, não possui um único ponto de falha. Além disso, ele fornece distribuição automática de linhas em um cluster. Um aplicativo cliente lendo ou gravando dados pode se conectar a qualquer nó em um cluster Cassandra.

Cassandra

O Cassandra fornece alta disponibilidade por meio de suporte integrado para replicação de dados. O número de réplicas a serem salvas é configurável. Cada réplica é armazenada em um nó diferente em um cluster. Portanto, se o fator de replicação for 3, mesmo se um ou dois nós falharem, o cluster ainda estará disponível.

Os dados são modelados no Cassandra usando uma hierarquia de keyspace, tabela, linha e coluna. Um keyspace é conceitualmente semelhante a um banco de dados ou esquema em um RDBMS. É uma coleção lógica de tabelas. Representa a família, é conceitualmente semelhante a uma tabela em um RDBMS. Uma família de colunas consiste em uma coleção de linhas particionadas.

Cada linha consiste em uma chave de partição e um conjunto de colunas. É importante observar que embora um keyspace, tabela, linha e coluna no Cassandra pareçam semelhantes a um esquema, tabela, linha e coluna, respectivamente, em um banco de dados relacional, sua implementação e armazenamento físico são diferentes.

Neste link (https://yadax.com.br/nosql/arquitetura-do-cassandra-replicacao/) o primeiro de uma série de artigos sobre Apache Cassandra (https://pt.wikipedia.org/wiki/Apache_Cassandra)

Na proxima aula continuamos com explicando conceitos e e mais componentes do cenário Big Data

Conceitos de estatística descritiva

A estatística descritiva pode ser entendida como uma ferramenta capaz de descrever ou resumir dados, mostrando aspectos importantes do conjunto de dados, como o tipo de distribuição associada e os valores mais representativos do conjunto, e permitindo criar visualizações referentes a tais aspectos.

População

População, ou universo, é o nome que se dá a um conjunto de unidades (elementos ou exemplares/dados se aproximarmos a definição ao contexto de mineração de dados) que compartilham características comuns e sobre o qual é pretendido o desenvolvimento de um estudo.

O conjunto de vendas realizadas durante o tempo de funcionamento de um restaurante é um exemplo de população (de vendas).

A população pode ser finita ou infinita.

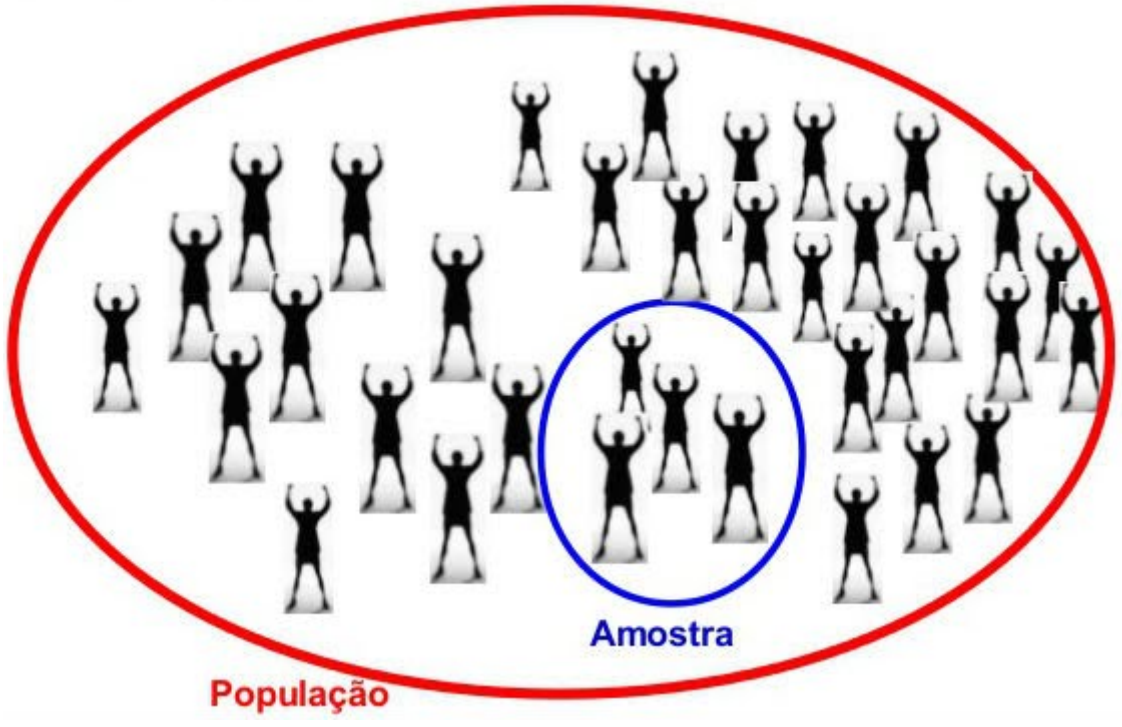
Sendo finita, se pequena, um estudo pode envolver sua totalidade. Caso seja muito grande ou infinita, faz-se necessário estabelecer uma amostra, para que um estudo sobre ela se viabilize.

Amostra

Uma amostra, então, é um subconjunto da população (também chamado de fração ou parte) que deve ser estabelecido com cuidado, seguindo técnicas apropriadas, pois o objetivo de obter uma amostra é reduzir o tamanho da população sem que características essenciais associadas a ela sejam perdidas.

Boas técnicas de amostragem geram amostras representativas e imparciais da população, ou seja, mantêm a proporcionalidade dos fenômenos que ocorrem na população e conferem chances iguais aos elementos da população de fazerem parte da amostra.

Há diferentes formas de estabelecer uma amostra, que resultam em diferentes tipos de amostragem: aleatória, estratificada, por conglomerados, acidental, intencional, por quotas etc...

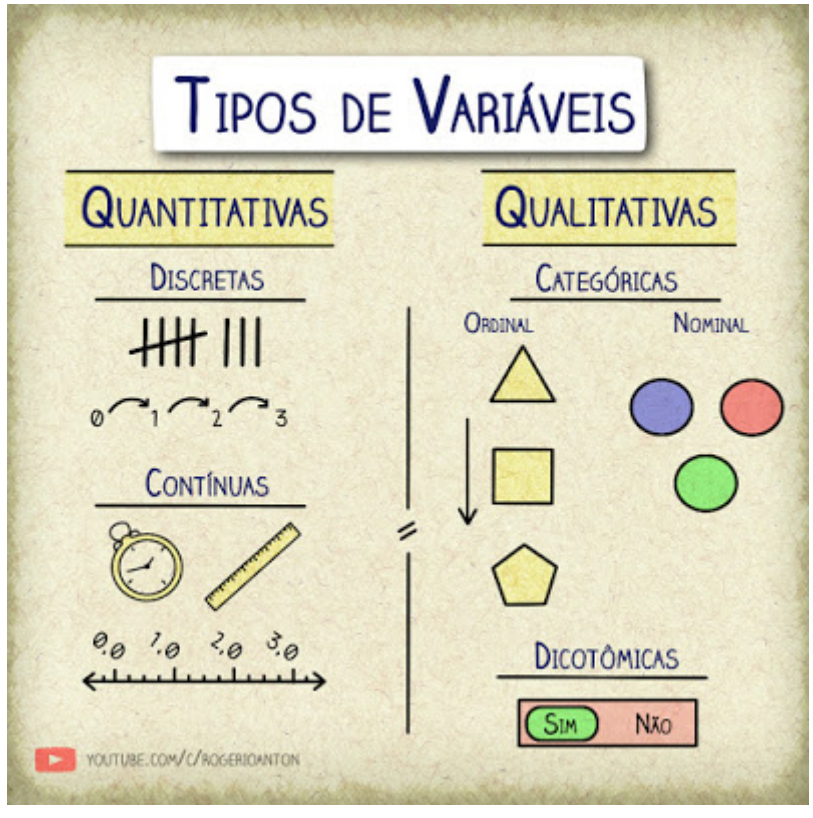

Amostra
População

Variável

Na estatística descritiva, uma variável diz respeito a uma característica associada a elementos de uma população.

Por exemplo, num restaurante, uma variável pode ser a temperatura em que um vinho deve ser servido (com valores como 5°C ou 12°C) ou o tipo de um prato presente no cardápio (com os seguintes valores, por exemplo: aperitivo, pasta, carne branca ou sobremesa quente).

Fazendo uma analogia à representação de um dado usada neste livro, uma variável é equivalente a um atributo descritivo. As variáveis, no contexto da estatística descritiva, podem ser **quantitativas** ou **qualitativas**.


TIPOS DE VARIÁVEIS
QUANTITATIVAS
QUALITATIVAS
DISCRETAS
0 1 2 3
CONTÍNUAS
0.0 1.0 2.0 3.0
CATEGÓRICAS
ORDINAL
NOMINAL
DICOTÔMICAS
SIM
NÃO
YOUTUBE.COM/C/ROGERIOANTON

Variável Quantitativa

As **variáveis quantitativas** (também nomeadas no contexto de mineração de dados como variáveis numéricas) são aquelas que podem ser medidas em uma escala de valores numéricos, e se dividem em **discretas** (assumem valores dentro de um conjunto finito ou infinito contável) ou **contínuas** (assumem valores em uma escala contínua).

Variável Qualitativa

As **variáveis qualitativas** (também nomeadas no contexto de mineração de dados como variáveis categóricas) são aquelas que representam uma classificação do elemento ao qual o valor da variável está associado, e se dividem em **nominais** (quando não há ordenação entre valores) ou **ordinais** (quando há uma ordenação entre os valores). Alguns exemplos de variáveis, considerando o contexto de um restaurante, e alguns valores que elas podem assumir estão organizados a seguir

Variáveis quantitativas

Discretas

Quantidade de bebidas vendidas: 15, 6, 3, 0
Idade do cliente: 5, 10, 25, 32, 57

Contínuas

Preço de um item do cardápio: 5,50, 35,00, 99,50
Peso de um corte de carne: 100g, 226g, 500g

Variáveis qualitativas

Nominais

Código da forma de pagamento: 11, 21, 31
Forma de pagamento: dinheiro, débito, crédito

Ordinais

Faixa etária dos clientes: 0 a 10 anos, 11 a 17 anos, 18 a 30 anos, 31 a 50 anos, mais de 50 anos
Mês de observação: janeiro, abril, novembro

Medidas de posição e separatrizes

Medidas de posição , ou medidas de tendência central , permitem encontrar os valores que orientam a análise dos dados no que diz respeito à sua localização, ou como a distribuição associada aos valores se comporta no universo amostrado.

As medidas de posição mais comuns no contexto de análise exploratória de dados são média aritmética, mediana e moda. Relacionadas com a mediana estão as medidas separatrizes percentis e quartis.

Para um conjunto de valores $v = v_1, v_2, \ldots, v_n$, em que n é a quantidade de valores do conjunto, a média aritmética simples (média(v) ou) é a soma dos valores do conjunto v, dividida pela quantidade de valores presentes nesse conjunto.

Formalmente, $m\acute{e}dia(v)$ é definida como:

$$media(v) = \frac{1}{n}\sum_{i=1}^{n} v_i$$

Medidas de posição e separatrizes

A **mediana** ($mediana(v)$) é o valor que divide a distribuição dos valores exatamente ao meio (podendo ser vista também como uma medida separatriz).

Importante lembrar que tal valor não precisa estar presente no conjunto v.

Para o cálculo da mediana, todos os valores presentes no conjunto devem ser ordenados de forma crescente (formando o conjunto v'), e a mediana será, então, dada por:

$$mediana(v) = \begin{cases} se\ n\ par\ v'^{i}\ onde\ i = \frac{n+1}{2} \\ se\ n\ impar\ \frac{v'_i + v'_{i+1}}{2}\ onde\ i = \frac{n}{2} \end{cases}$$

em que i é uma posição no conjunto de valores v'.

Medidas de posição e separatrizes

moda , por sua vez, é o valor mais frequente em um conjunto de valores.

Ela é a única medida de posição que pode assumir mais de um valor.

Essa situação ocorre quando dois ou mais valores aparecem no conjunto de valores v com a mesma frequência, a máxima no conjunto. Assim, um conjunto de valores pode ser:

- **amodal** (não possui moda),
- **unimodal** (possui uma moda),
- **bimodal** (possui duas modas) ou
- **multimodal** (possui diversas modas).

Percentis e Quartis

Os **percentis** e **quartis** são medidas separatrizes que dividem o conjunto de valores, ordenado de forma crescente, em partes tão iguais quanto possível.

O **percentil de ordem p** determina os **p% menores valores contidos em v'**, e a posição i, que delimita o percentil de ordem p em v', é dada por:

$$i = \frac{p(n+1)}{100}$$

Medidas de dispersão

As medidas de posição são úteis para apresentar uma sumarização dos dados. No entanto, não são capazes de descrever a variação ou dispersão do conjunto de valores.

Para esse fim, aplicam-se as medidas de dispersão , capazes de descrever o quanto os valores de um conjunto estão próximos ou distantes de uma medida central, como a média.

As medidas mais comuns de dispersão ou variância dos dados são a **amplitude**, **variância**, **desvio-padrão**.

Amplitude

Para um conjunto de valores v, a medida da amplitude é a diferença entre o maior e o menor valor do conjunto, portanto:

$$amplitude(v) = max(v) - min(v).$$

Embora a medida da amplitude seja simples, sua interpretação precisa ser feita com cuidado, pois ela pode ser influenciada por valores extremos, conhecidos como outliers que veremos mais a frente.

Variância

A **variância** $\sigma^2(v)$ é uma medida de dispersão definida como a média dos quadrados das diferenças entre cada valor do conjunto v e a média desse conjunto. Formalmente, tem-se:

$$\sigma^2(v) = \frac{1}{n}\sum_{i=1}^{n}(v_i - media(v))^2$$

Desvio-padrão

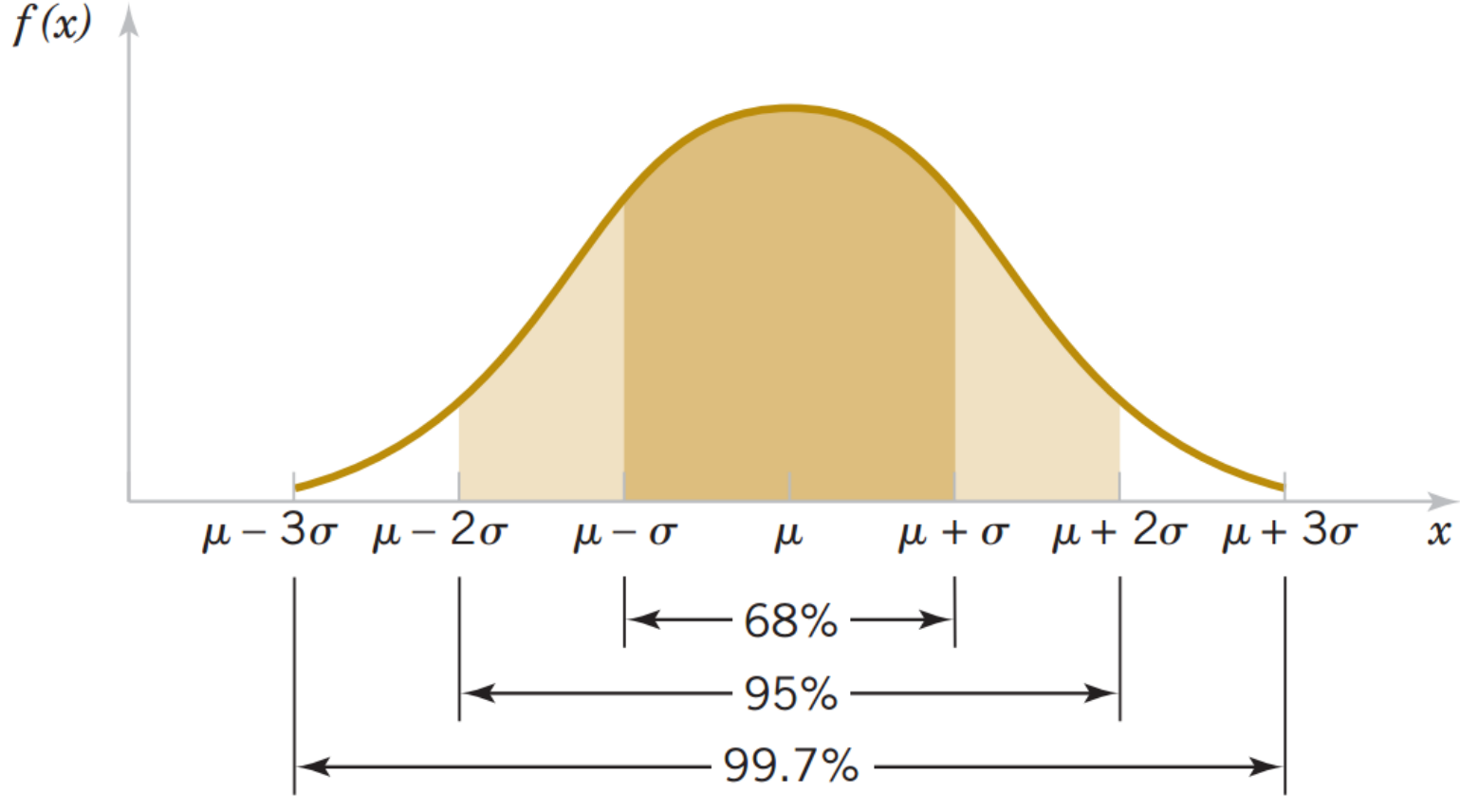
O **desvio-padrão** $\sigma(v)$ é a raiz quadrada da variância.

O uso do desviopadrão só faz sentido quando a média for usada como medida de posição (ou de centro).

Ele é semelhante à medida de amplitude, com a diferença de que o cálculo do desvio-padrão usa todos os valores de um conjunto.

O resultado dessa medida geralmente é usado para verificar a consistência de um fenômeno (um fenômeno é consistente quando o cálculo do desvio-padrão resulta em valores baixos).

Distribuição Normal μ e σ



Funções estatísticas com DataFrames no Apache Spark

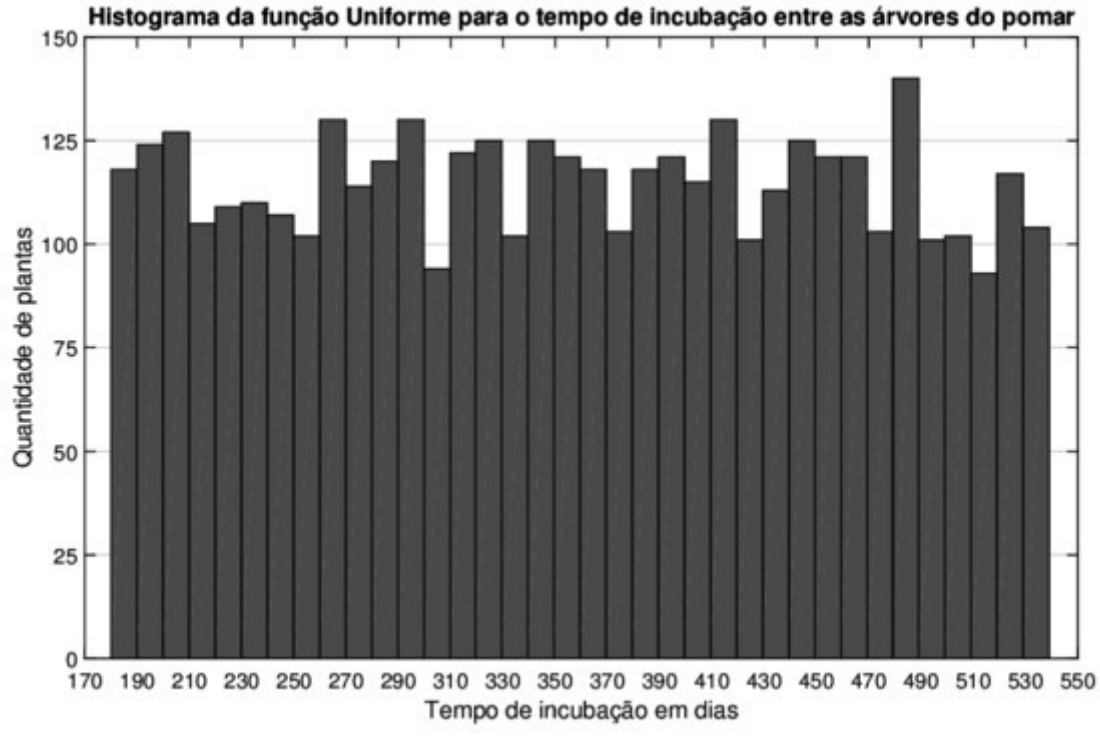
A estatística é uma parte importante da ciência de dados cotidiana. Nesta aula, examinamos algumas das funções importantes, incluindo:

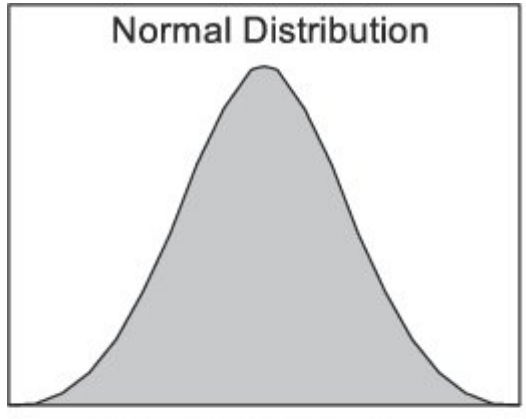
1. Geração de dados aleatórios
2. Estatísticas resumidas e descritivas
3. Covariância e correlação de amostra
4. Tabulação cruzada (a.k.a. tabela de contingência)
5. Itens frequentes
6. Funções matemáticas

Traduzido deste link (https://databricks.com/blog/2015/06/02/statistical-and-mathematical-functions-with-dataframes-in-spark.html)

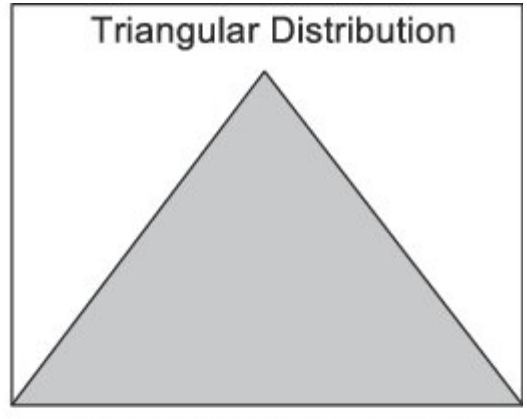
1. Geração de dados aleatórios A geração de dados aleatórios é útil para testar algoritmos existentes e implementar algoritmos aleatórios, como projeção aleatória. Fornecemos métodos em sql.functions para gerar colunas que contêm i.i.d. valores retirados de uma distribuição, por exemplo, uniforme (`rand`) e normal padrão (`randn`).

- Mais sobre distribuição uniforme neste link (https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_uniforme)
- Mais sobre distribuição normal neste link (https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_normal)
- Material interessante sobre distribuições simulações neste link (https://rpubs.com/EstatBasica/Cap9)

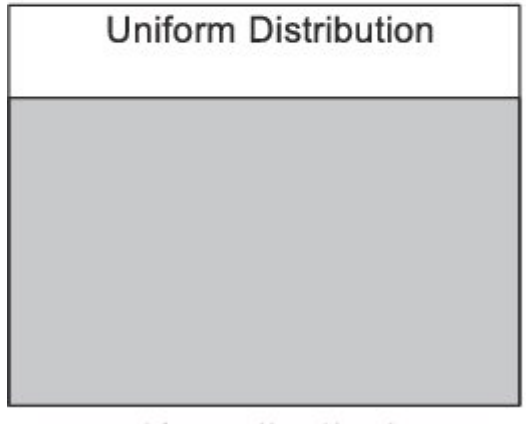

Histograma da função Uniforme para o tempo de incubação entre as árvores do pomar
Quantidade de plantas
0
25
50
75
100
125
150
170 190 210 230 250 270 290 310 330 350 370 390 410 430 450 470 490 510 530 550
Tempo de incubação em dias



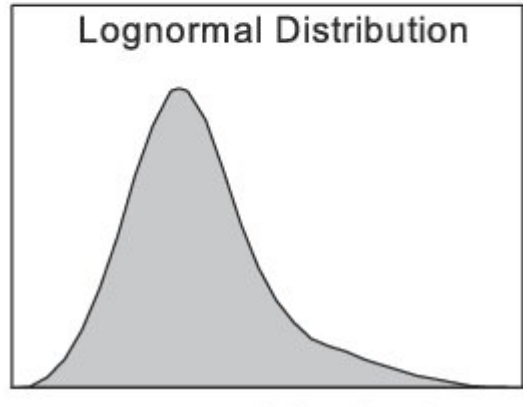
Normal distribution



Triangular distribution



Uniform distribution



Log-normal distribution

In [0]:
```
from pyspark.sql.functions import rand, randn
```

In [0]:

```
 # Crie um DataFrame com uma coluna interna e 10 linhas.
df = sqlContext.range(0, 10)
df.show()
```

```
+---+
 id|
+---+
  0|
  1|
  2|
  3|
  4|
  5|
  6|
  7|
  8|
  9|
+---+
```

In [0]:
```
# Gere duas outras colunas usando distribuição uniforme e distribuição normal.
df.select("id", rand(seed=10).alias("uniform"), randn(seed=27).alias("normal")).show()
```

```
+---+------------------+--------------------+
| id|           uniform|              normal|
+---+------------------+--------------------+
|  0| 0.1709497137955568| -0.8664700627108758|
|  1|0.03422639313807285| 0.45800664187768786|
|  2| 0.3654625958161396| 0.16420866768809156|
|  3| 0.4175019040792016| -1.0451987154313813|
|  4| 0.9899129399827472|  0.9543129239458202|
|  5|0.16452185994603707|  0.8306551181802446|
|  6|0.18141810315190554|  1.3792681955381285|
|  7|0.49595620559530806|-0.025114845140222...|
|  8| 0.9697474945375325|  0.5991404703866096|
|  9|0.07530606222259384|  1.7264843633887004|
+---+------------------+--------------------+
```

2. Estatísticas de sumarização e descritivas

A primeira operação a ser realizada após a importação de dados é para ter uma ideia de sua aparência. Para colunas numéricas, conhecer as estatísticas de resumo descritivo pode ajudar muito no entendimento da distribuição de seus dados. A função describe retorna um DataFrame contendo informações como número de entradas não nulas (contagem), média, desvio padrão e valores mínimo e máximo para cada coluna numérica.

In [0]:

```
from pyspark.sql.functions import rand, randn
# Uma maneira ligeiramente diferente de gerar as duas colunas aleatórias
df = sqlContext.range(0, 10).withColumn('uniform', rand(seed=10)).withColumn('normal', r
andn(seed=27))
```

In [0]:
```
df.show()
```

```
+---+------------------+--------------------+
| id|           uniform|              normal|
+---+------------------+--------------------+
|  0| 0.1709497137955568| -0.8664700627108758|
|  1|0.03422639313807285| 0.45800664187768786|
|  2| 0.3654625958161396| 0.1642086676880915 6|
|  3| 0.4175019040792016| -1.0451987154313813|
|  4| 0.9899129399827472|  0.9543129239458202|
|  5|0.16452185994603707|  0.8306551181802446|
|  6|0.18141810315190554|  1.3792681955381285|
|  7|0.4959562055953080 6|-0.025114845140222 99|
|  8| 0.9697474945375325|  0.5991404703866096|
|  9|0.07530606222259384|  1.7264843633887004|
+---+------------------+--------------------+
```

Estatística descritiva

A estatística descritiva pode ser entendida como uma ferramenta capaz de descrever ou resumir dados, mostrando aspectos importantes do conjunto de dados, como o tipo de distribuição associada e os valores mais representativos do conjunto, e permitindo criar visualizações referentes a tais aspectos.

Para saber mais sobre estatística descritiva (https://github.com/DeepFluxion/Analise_Dados_Linguagem_R/blob/master/Notebooks/Au
Se você tiver um DataFrame com um grande número de colunas, também pode executar `describe()` em um subconjunto das colunas:

In [0]:
```
df.describe('uniform', 'normal').show()
```

```
+-------+------------------+------------------+
summary|         uniform|          normal|
+-------+------------------+------------------+
 count|              10|              10|
  mean| 0.3865003272265095| 0.4175292757722803|
stddev|0.34583270415314343| 0.8931883269570855|
   min|0.03422639313807285|-1.0451987154313813|
   max| 0.9899129399827472| 1.7264843633887004|
+-------+------------------+------------------+
```

Obviamente, embora a descrição funcione bem para uma análise exploratória rápida de dados, você também pode controlar a lista de estatísticas descritivas e as colunas às quais se aplicam usando a seleção normal em um DataFrame:

In [0]:
```
from pyspark.sql.functions import mean, min, max
df.select([mean('uniform'), min('uniform'), max('uniform')]).show()
```

```
+-----------------+-----------------+-----------------+
    avg(uniform)|     min(uniform)|     max(uniform)|
+-----------------+-----------------+-----------------+
0.3865003272265095|0.03422639313807285|0.9899129399827472|
+-----------------+-----------------+-----------------+
```

In [0]:
```
df.select([mean('normal'), min('normal'), max('normal')]).show()
```

```
+-----------------+-----------------+-----------------+
    avg(normal)|      min(normal)|      max(normal)|
+-----------------+-----------------+-----------------+
0.4175292757722803|-1.0451987154313813|1.7264843633887004|
+-----------------+-----------------+-----------------+
```

3. Covariância e Correlação de uma Amostra

Covariância é uma medida de como duas variáveis mudam uma em relação à outra. Um número positivo significaria que há uma tendência de que, à medida que uma variável aumenta, a outra também aumenta. Um número negativo significaria que à medida que uma variável aumenta, a outra variável tem uma tendência de diminuir. A Formula da Covariância é :

$$S_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Link:

Covariância (https://pt.wikipedia.org/wiki/Covari%C3%A2ncia#:~:text=Em%20teoria%20da%20probab

Correlação (https://pt.wikipedia.org/wiki/Correla%C3%A7%C3%A3o)

A covariância de amostra de duas colunas de um DataFrame pode ser calculada da seguinte forma:

In [0]:

```
from pyspark.sql.functions import rand
df = sqlContext.range(0, 100).withColumn('rand1', rand(seed=10)).withColumn('rand2', ran
d(seed=27))
df.show()
```

```
+---+------------------+------------------+
| id|             rand1|             rand2|
+---+------------------+------------------+
|  0| 0.1709497137955568| 0.8894415403143504|
|  1| 0.8051143958005459| 0.9658033539477212|
|  2| 0.5775925576589018|0.18783354174799438|
|  3| 0.9476047869880925|0.28489885393300474|
|  4|   0.2093704977577| 0.520716534523733 7|
|  5|0.36664222617947817|  0.29022461831445|
|  6| 0.8078688178371882| 0.7962482611496515|
|  7| 0.7135143433452461| 0.2948555807675217|
|  8| 0.7195325566306053| 0.840640428653815 2|
|  9|0.3133529231117545 6|0.05654934737553252|
| 10| 0.8062503712025726| 0.5591744861079746|
| 11|0.10814914646176654|0.31446013845788146|
| 12|0.03422639313807285|  0.808152402708103|
| 13|0.11682250456449328| 0.1753332481572636|
| 14| 0.3252227809451943| 0.9894530945118741|
| 15| 0.4358851046481089| 0.7275724505361818|
| 16|0.27831507107951303|  0.72995211246392|
| 17| 0.8601856145216583|0.30284571233687096|
| 18| 0.8069471711557737| 0.9569877068769661|
```

```
 19| 0.2718207931603598| 0.4974632546664315|
+---+------------------+------------------+
only showing top 20 rows
```

In [0]:
```
df.stat.cov('rand1', 'rand2')
```

Out[47]: -0.0024860826156065 23

In [0]:
```
df.stat.cov('id', 'id')
```

Out[48]: 841.6666666666666

In [0]:
```
df.display()
```

id	rand1	rand2
0	0.1709497137955568	0.8894415403143504
1	0.8051143958005459	0.9658033539477212
2	0.5775925576589018	0.18783354174799438
3	0.9476047869880925	0.28489885393300474
4	0.2093704977577	0.5207165345237337
5	0.36664222617947817	0.29022461831445
6	0.8078688178371882	0.7962482611496515
7	0.7135143433452461	0.2948555807675217
8	0.7195325566306053	0.8406404286538152
9	0.31335292311175456	0.05654934737553252
10	0.8062503712025726	0.5591744861079746
11	0.10814914646176654	0.31446013845788146
12	0.03422639313807285	0.808152402708103
13	0.11682250456449328	0.1753332481572636

Como você pode ver acima, a covariância das duas colunas geradas aleatoriamente é próxima de zero, enquanto a covariância da coluna id com ela mesma é muito alta.

O valor de covariância de 842 pode ser difícil de interpretar.

Correlação é uma medida normalizada de covariância que é mais fácil de entender, pois fornece medidas quantitativas da dependência estatística entre duas variáveis aleatórias.

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}}$$

veja mais sobre correlação neste link (https://github.com/DeepFluxion/Analise_Dados_Linguagem_R/blob/master/Notebooks/Au

In [0]:
```
df.stat.corr('rand1', 'rand2')
```

Out[50]: -0.0312321111106837743

In [0]:
```
df.stat.corr('id', 'id')
```

Out[51]: 1.0

No exemplo acima, id se correlaciona perfeitamente com ele mesmo, enquanto as duas colunas geradas aleatoriamente têm baixo valor de correlação.

In [0]:
```
df.display()
```

id	rand1	rand2
0	0.1709497137955568	0.8894415403143504
1	0.8051143958005459	0.9658033539477212
2	0.5775925576589018	0.18783354174799438
3	0.9476047869880925	0.28489885393300474
4	0.2093704977577	0.5207165345237337
5	0.36664222617947817	0.29022461831445
6	0.8078688178371882	0.7962482611496515
7	0.7135143433452461	0.2948555807675217
8	0.7195325566306053	0.8406404286538152
9	0.31335292311175456	0.05654934737553252
10	0.8062503712025726	0.5591744861079746
11	0.10814914646176654	0.31446013845788146
12	0.03422639313807285	0.808152402708103
13	0.11682250456449328	0.1753332481572636

4. Tabulação cruzada (tabela de contingência)

A tabulação cruzada fornece uma tabela da distribuição de frequência para um conjunto de variáveis. A tabulação cruzada é uma ferramenta poderosa em estatística usada para observar a significância estatística (ou independência) das variáveis. No Spark os usuários são capazes de fazer tabulação cruzada de duas colunas de um DataFrame para obter as contagens dos diferentes pares observados nessas colunas. Aqui está um exemplo de como usar a crosstab para obter a tabela de contingência.

In [0]:

```
# Crie um DataFrame com duas colunas (nome, item)
nomes = ["Alice", "Roberto", "Marcos"]
items = ["leite", "pão", "manteiga", "maças", "laranjas"]
df = sqlContext.createDataFrame([(nomes[i % 3], items[i % 5]) for i in range(100)], ["no
me", "item"])
```

In [0]:
```
df.display()
```

nome	item
Alice	leite
Roberto	pão
Marcos	manteiga
Alice	maças
Roberto	laranjas
Marcos	leite
Alice	pão
Roberto	manteiga
Marcos	maças
Alice	laranjas
Roberto	leite
Marcos	pão
Alice	manteiga
Roberto	maças

In [0]:
```
df.stat.crosstab("nome", "item").show()
```

```
+---------+--------+-----+--------+-----+---+
nome_item|laranjas|leite|manteiga|maças|pão|
+---------+--------+-----+--------+-----+---+
  Marcos|     6|   7|     7|   7| 6|
 Roberto|     7|   6|     7|   6| 7|
   Alice|     7|   7|     6|   7| 7|
+---------+--------+-----+--------+-----+---+
```

Uma coisa importante a se ter em mente é que a cardinalidade das colunas nas quais executamos a crosstab não pode ser muito grande. Ou seja, o número de “nome” e “item” distintos não pode ser muito grande. Imagine se “item” contém 1 bilhão de entradas distintas: como você colocaria essa tabela na sua tela ?!

5. Itens Freqüentes

Descobrir quais itens são frequentes em cada coluna pode ser muito útil para entender um conjunto de dados. No Spark, os usuários podem encontrar os itens frequentes para um conjunto de colunas usando DataFrames. Implementamos um algoritmo de uma passagem proposto por Karp et al. Este é um algoritmo rápido e aproximado que sempre retorna todos os itens frequentes que aparecem em uma proporção mínima de linhas especificada pelo usuário. Observe que o resultado pode conter falsos positivos, ou seja, itens que não são frequentes.

In [0]:
```
df = sqlContext.createDataFrame([(1, 2, 3) if i % 2 == 0 else (i, 2 * i, i % 4) for i in range(100)], ["a", "b", "c"])
df.show(10)
```

```
+---+---+---+
 a| b| c|
+---+---+---+
 1| 2| 3|
 1| 2| 1|
 1| 2| 3|
 3| 6| 3|
 1| 2| 3|
 5| 10| 1|
 1| 2| 3|
 7| 14| 3|
 1| 2| 3|
 9| 18| 1|
+---+---+---+
only showing top 10 rows
```

In [0]:

```
freq = df.stat.freqItems(["a", "b", "c"], 0.4)
freq.show()
```

```
+-----------+-----------+-----------+
a_freqItems|b_freqItems|c_freqItems|
+-----------+-----------+-----------+
  [11, 1]|   [2, 22]|    [1, 3]|
+-----------+-----------+-----------+
```

Dado o DataFrame acima, o código a seguir encontra os itens frequentes que aparecem 40% do tempo para cada coluna:

In [0]:
```
freq.collect()[0]
```

Out[58]: Row(a_freqItems=[11, 1], b_freqItems=[2, 22], c_freqItems=[1, 3])

Como você pode ver, “11” e “1” são os valores frequentes para a coluna “a”.