DSA – Formação Cientista de Dados

5. Big Data Real-Time Analytics com Python e Spark

5.1. Introdução

Intrudução ao Curso:

Este é o Segundo Curso da Formação Cientista de Dados.

Estrutura do Curso:

- Parte 1 Análise de Dados com Python
- Parte 2 Estatística
- Parte 3 Machine Learning
- Parte 4 Spark

O que não veremos neste curso?

Conteúdo Básico de Python Configuração do Cluster Spark

Quais ferramentas usaremos?

Anaconda Python

Jupyter Notebook

Apache Spark

Ambiente em Cloud da Databricks

Avaliação Final

50 questões – 3 tentativas – 70%

Objetivos ao final deste curso:

- Desenvolver habilidades de processamento e análise de dados em tempo real.
- Aprender técnicas de Machine Learning e Processamento de Dados.
- Compreender os conceitos do ciclo de vida de projetos de Big Data Analytics.
- Aplicar o conhecimento deste curso em casos reais do dia a dia.
- Compreender a função da Estatística no processo de Data Science.

O que é o Apache Spark?

Spark é atualmente o projeto open-source mais ativo ligado a Big Data. Embora ele seja considerado o sucessor do Hadoop/MapReduce, veremos que não é bem assim e as duas tecnologias podem ser usadas em conjunto. Mas o Spark realmente oferece vantagens.

Principais Benefícios do Apache Spark

O Spark oferece 3 benefícios principais:

- Fácil de usar
- Veloz
- Engine de uso geral

Apache Spark é uma plataforma de computação em cluster (conjunto de computadores), criado para ser veloz e de uso geral, sendo ideal para processamento iterativo (que se repete) e processamento de streaming de dados (fluxo contínuo de dados).

Spark realiza a computação em memória (o que ajuda a explicar sua velocidade), mas também é eficiente quando executa aplicações em disco.

Por Que Aprender Apache Spark?

- Cada vez mais utilizado por empresas interessadas em analisar dados em tempo real.
- Uma das tecnologias mais quentes em Big Data.
- Suporte cada vez maior.
- Crescente demanda por profissionais que saibam processar e analisar dados em tempo real.

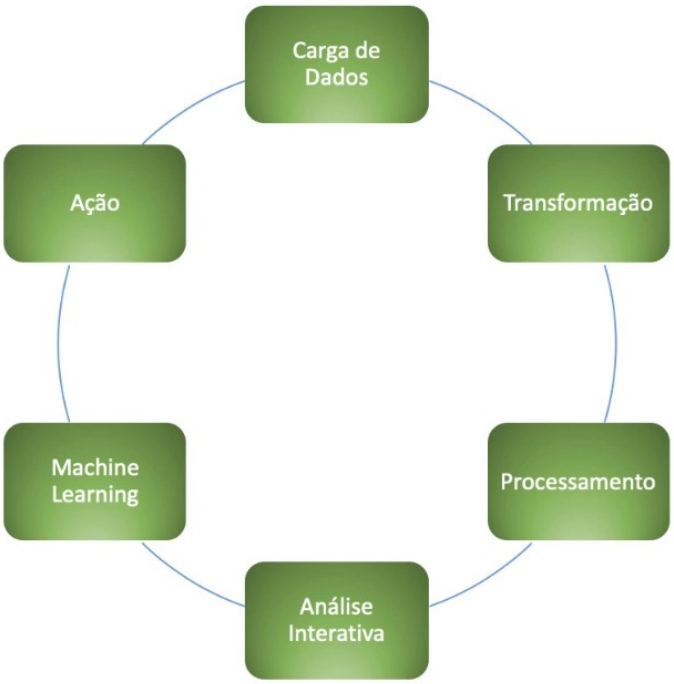
O Spark vem sendo utilizado principalmente em:

- Detecção de Fraudes em tempo real.
- Detecção de Invasão de Redes.
- Campanhas de Marketing e Propaganda em tempo real.
- Análise de Sentimento em Redes Sociais.
- Pode ser usado como ferramenta ETL (Extraction Transformation Load)

Por Que Python e Spark?

Embora o Spark tenha sido desenvolvido em Java e Scala, as habilidades da linguagem Python para manipulação e análise de dados, além de Machine Learning, a tornam a ferramenta ideal para trabalhar com Spark!

Apache Spark Workflow



Descrição do (ciclo do) Workflow:

1. Carga de Dados
    - Fontes que geram dados em tempo real, HDFS, NoSQL
2. Transformação
    - Filtro, Limpeza, Join
3. Processamento
    - Em memória, HDFS, NoSQL
4. Análise Interativa
    - Shell, SparkSQL
5. Machine Learning
    - Modelos Preditivos Aplicados a Stream de Dados
6. Ação
    - Tomada de Decisão

Real-Time Analytics e Computação Distribuída

Com a explosão do Big Data, tem-se 2 grandes desafios:

1. Coletar os dados
2. Analisar em Tempo Real

O Big Data é definido pelos seus V's: Volume, Variedade, Veracidade e Velocidade.

Neste curso vamos focar no V de velocidade do Big Data, ou seja, velocidade com que os dados são gerados.

Até alguns anos atrás, a única forma de analisar todo este conjunto de dados seria através de soluções analíticas que eram executadas em apenas um computador com grande capacidade computacional, um servidor!

Atualmente, é possível resolver esse problema através de **Computer Clusteres**. Um cluster é um grupo de máquinas que trabalha em conjunto e se comportam como apenas uma grande máquina.

Ou seja, problema resolvido? Não!

A computação em cluster resolveu o problema de armazenar um grande conjunto de dados, mas gerou outro: a maioria das aplicações linguagens de programação não estão prontas para trabalhar em **Computação Distribuída**, ou seja, em vários servidores que requerem uma série de controles e transações.

Hadoop e Spark foram pensados para Computação Distribuída!

O Que é Streaming de Dados?

Dados em streaming são dados gerados continuamente por milhares de fontes de dados, que geralmente enviam os registros de dados simultaneamente, em tamanhos pequenos (na ordem dos kilobytes).

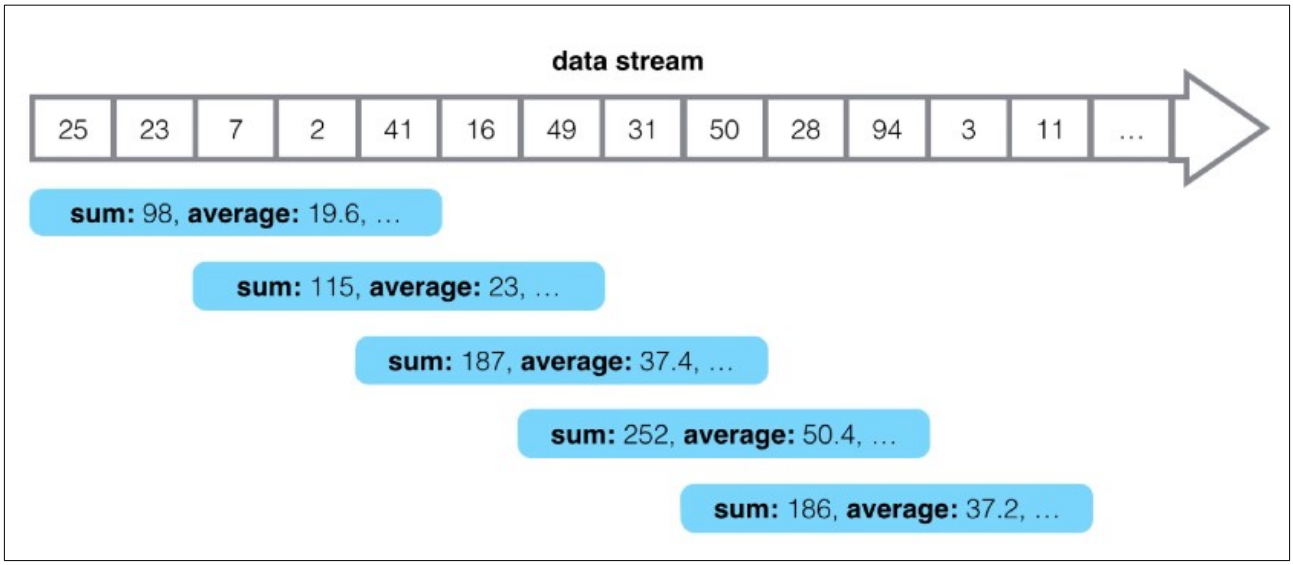
Os dados em streaming devem ser processados de maneira sequencial e incremental por registro e usados para uma ampla variedade de análises de dados, como correlações, agregações, filtragem e amostragem.

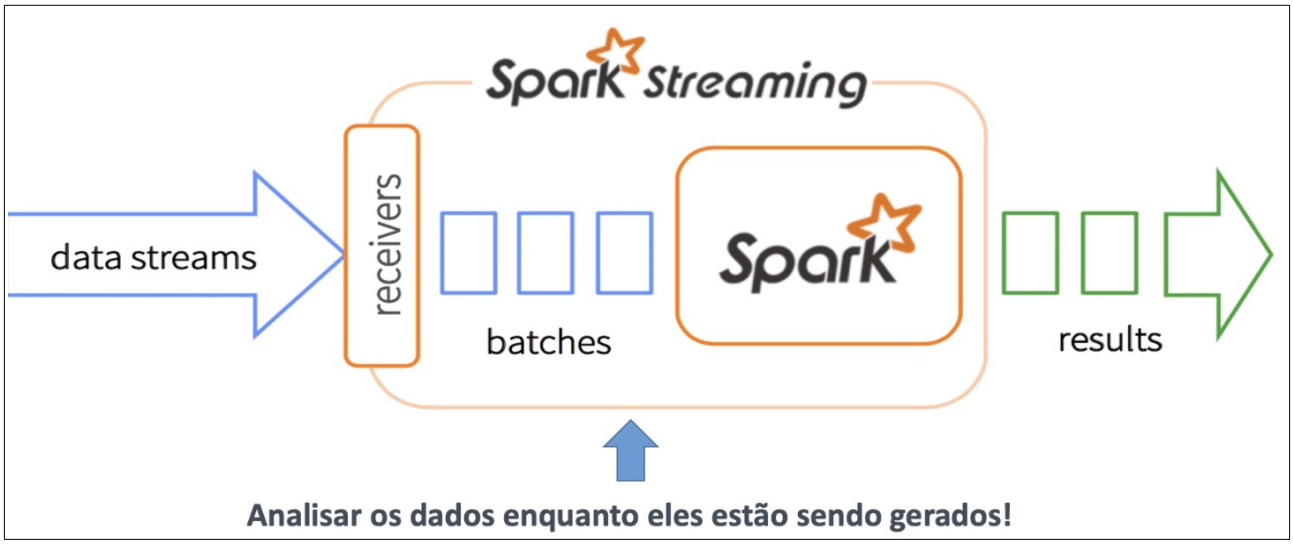
Possíveis Fontes para o Spark Streaming:

- Sensores em veículos
- Monitoramento de cotação de ações na bolsa de valores
- Arquivos texto (no momento em que eles são gerados)
- Redes Sociais (Facebook, Twitter, Instagram)

- Dados de dispositivos móveis
- Cliques em web sites
- Apache Kafka
- Apache Flume

Janela de tempo ou Window – análise de dados em tempo real:





Processamento de Lotes (Batch) x Processamento de Streaming de Dados

	Batch	**Streams**
Escopo de Dados	Consultas ou processamento de todos ou da maioria dos dados no conjunto de dados.	Consultas ou processamento de dados dentro de um período rotacional, ou apenas do registro de dados mais

		recente.
Tamanho dos Dados	Grandes lotes de dados.	Registros individuais ou microlotes compostos de alguns registros.
Desempenho	Latências em minutos ou horas.	Exige latência na ordem dos segundos ou milissegundos.
Análise	Análises de dados mais complexas.	Análises de dados menos complexas.

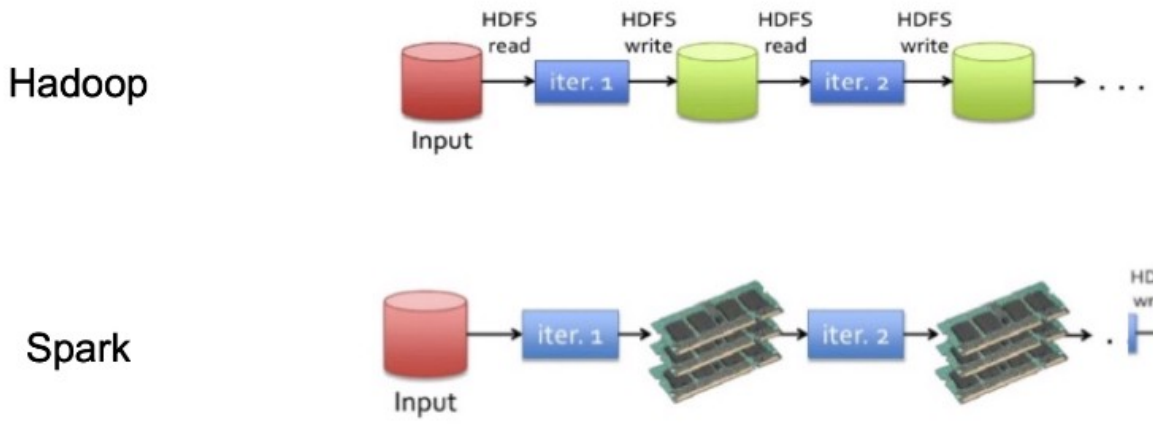
Apache Spark X Apache Hadoop

O Apache Spark não substituirá o Apache Hadoop. Isso porque o Hadoop é um conjunto de componentes. Por exemplo, o Spark não possui um mecanismo de armazenamento. Sendo assim, é possível utilizar o Spark para processamento (em memória), enquanto utiliza-se o Hadoop para armazenamento distribuído.

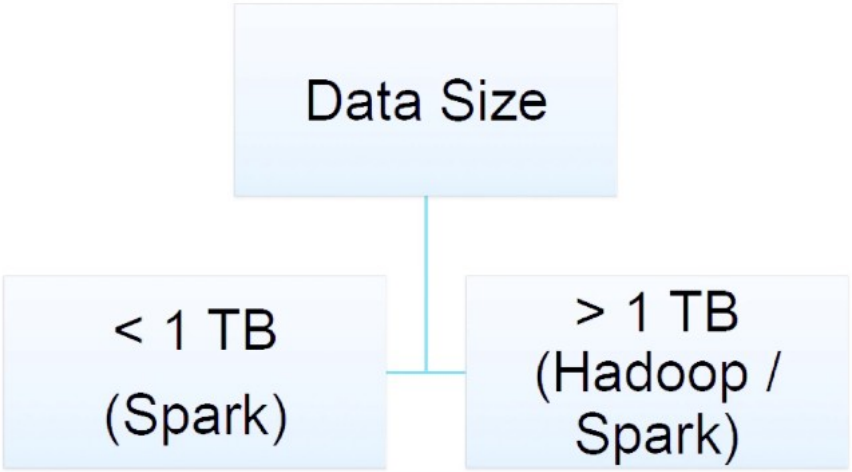
O Hadoop foi criado para processar grande volume de dados (big data), em batch.

O Apache Spark foi a primeira plataforma de Big Data a integrar processamento de dados em batch, streaming e computação distribuída em um único framework.

Hadoop	Spark
Armazenamento distribuído + Computação distribuída	Somente computação distribuída
Framework MapReduce	Computação genérica
Normalmente processa dados em disco (HDFS)	Em disco / Em memória
Não é ideal para trabalho iterativo (de repetição)	Excelente para trabalhos iterativos (Machine Learning)
Processo batch	Até 10x mais rápido para dados em disco Até 100x mais rápido para dados em memória
Basicamente Java	Suporta Java, Python, Scala e R
Não possui um shell (terminal) unificado	Shell para exploração ad-hoc (tempo real)



O Spark é muito bom quando os dados podem ser processados em memória. Mas e quando não podem?



	Hadoop	**Spark**
Processamento batch	Hadoop MapReduce (Java, Pig, Hive)	Spark RDD (Java, Python, Scala, R)
Query SQL	Hadoop: Hive	Spark SQL
Processamento Stream / Processamento em Tempo Real	Storm, Kafka	Spark Streaming
Machine Learning	Mahout	Spark ML Lib
Algoritmos iterativos	Lento	Muito rápido (em memória)
Workflow ETL (Extraction, Transformation, Load)	Pig, Flume	Pig com Spark ou Mix de Spark SQL e programação RDD
Volume de Dados	Volume gigante (Petabytes)	Volume médio (Gigabytes / Terabytes)

Preparando o Ambiente Python e Spark

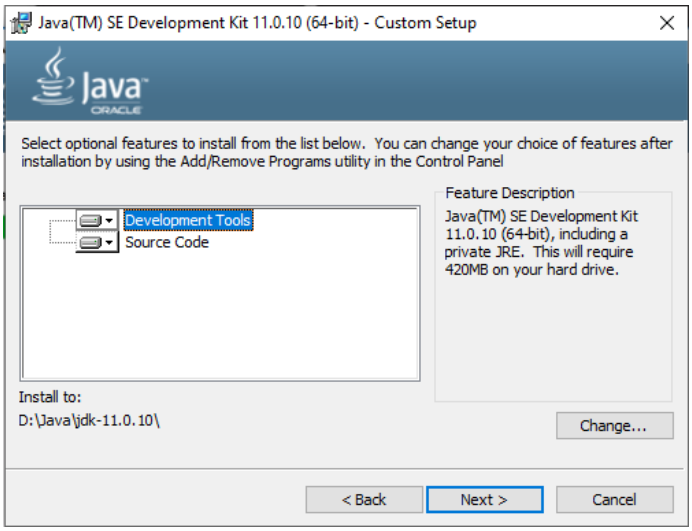
1. Faça o download o Anaconda3
    1. Acesse: www.anaconda.org
    2. Em downloads, utilize a versão do anaconda com Python 3 (3.7)
    3. Caso queira instalar uma versão diferente, utilize o google e digite “download anaconda archive”
    4. Para informações sobre o python, acesse: www.python.org
2. Faça o download do Spark
    1. Acesse: www.spark.apache.org
    2. Utilize a versão 2.4.2

3. Caso queira instalar uma versão diferente, utilize o google e digite “download spark archive”

3. Faça o download do Java JDK 11
    1. Caso queira instalar uma versão diferente, utilize o google e digite “download jdk 11”

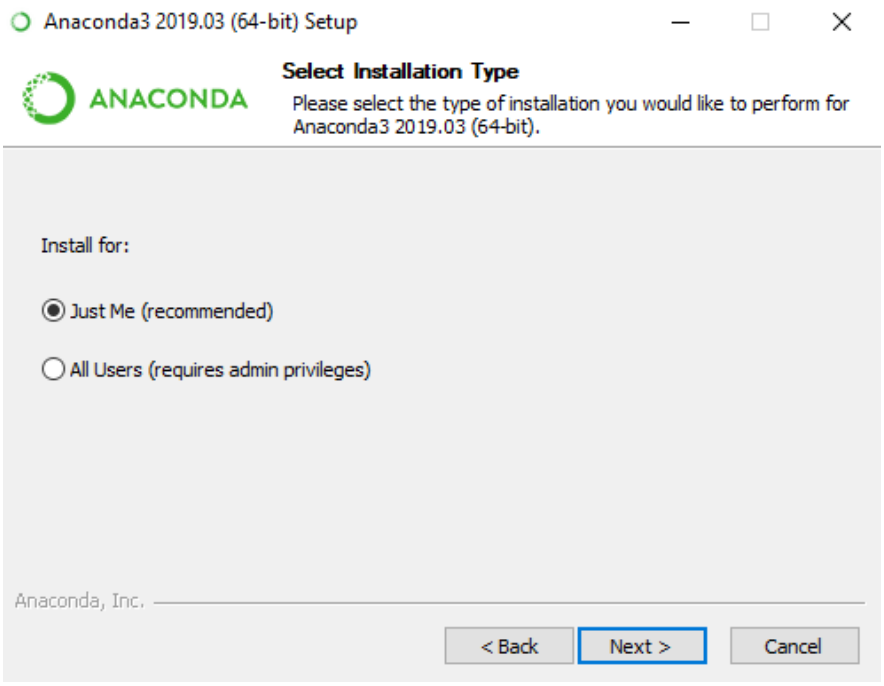
Instalação:

1. Comece pelo JDK:
    1. Instale num diretório Java (altere para um local que não haja espaços nos nomes dos diretórios).
        - D:\Java\jdk-11.0.10\
    2. Clique em Next

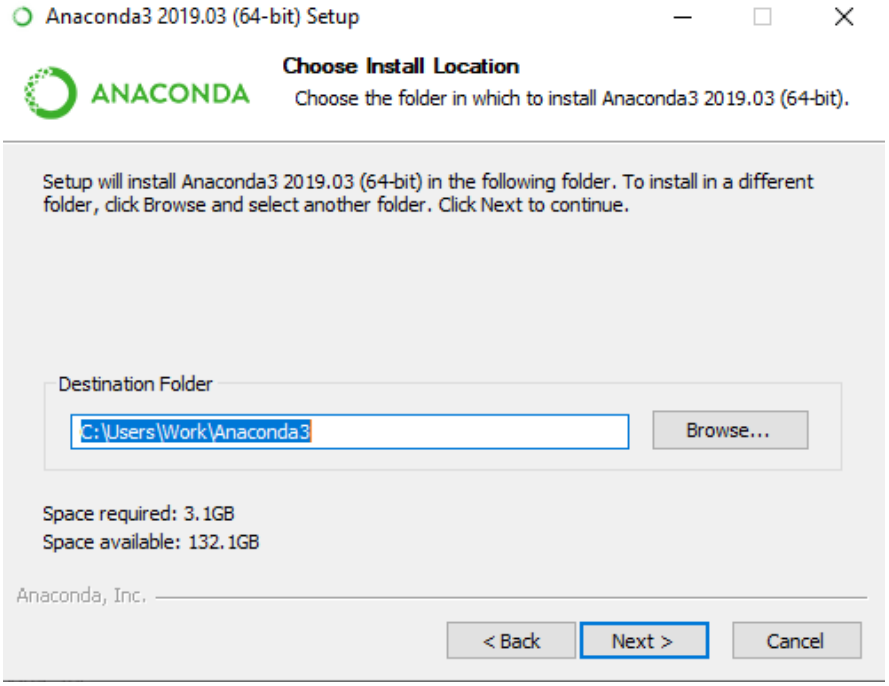


    3. Clique em Close

2. Agora, instale o Anaconda:
    1. Instale apenas no usuário de trabalho:

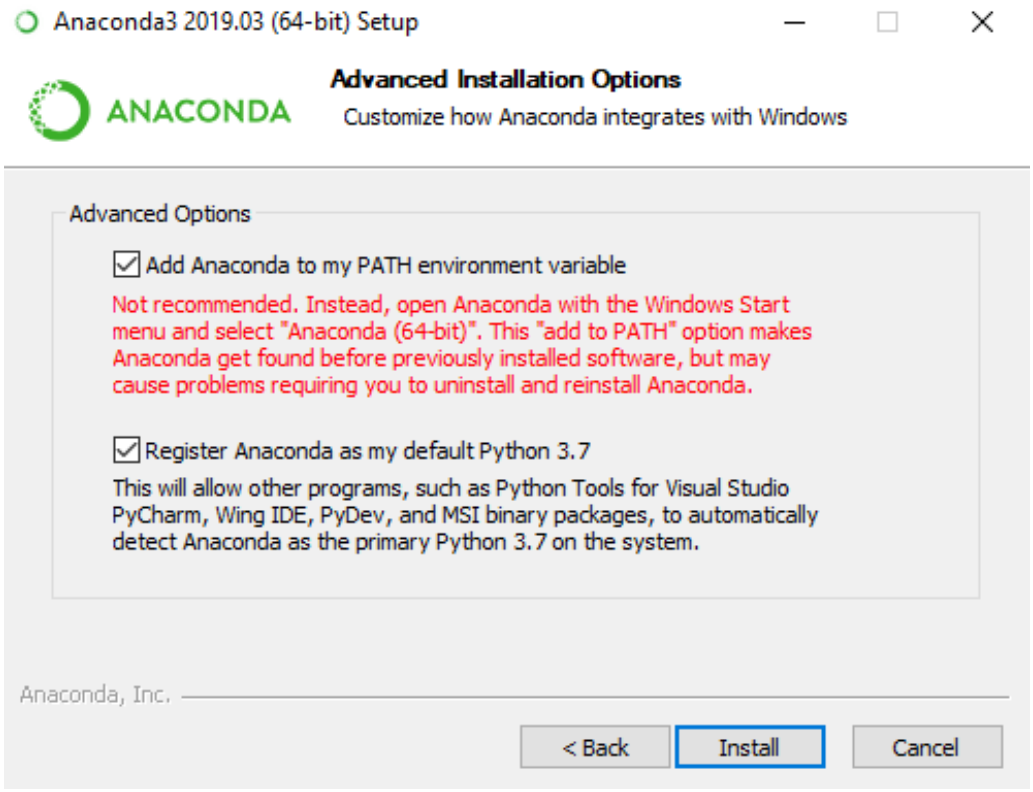


    2. Selecione um local sem espaços ou acento:



    Ex: C:\Users\Work\Anaconda3

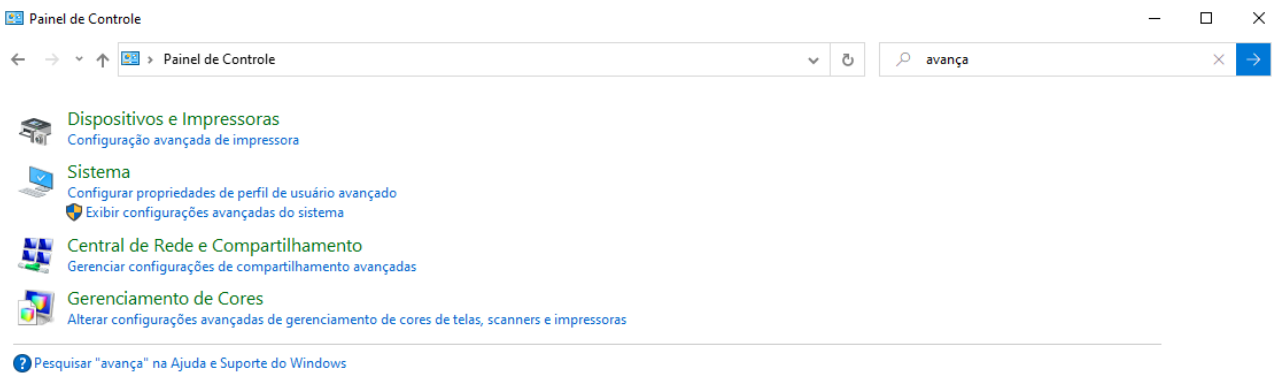
    3. Selecione a opção de adicionar o Anaconda ao PATH:



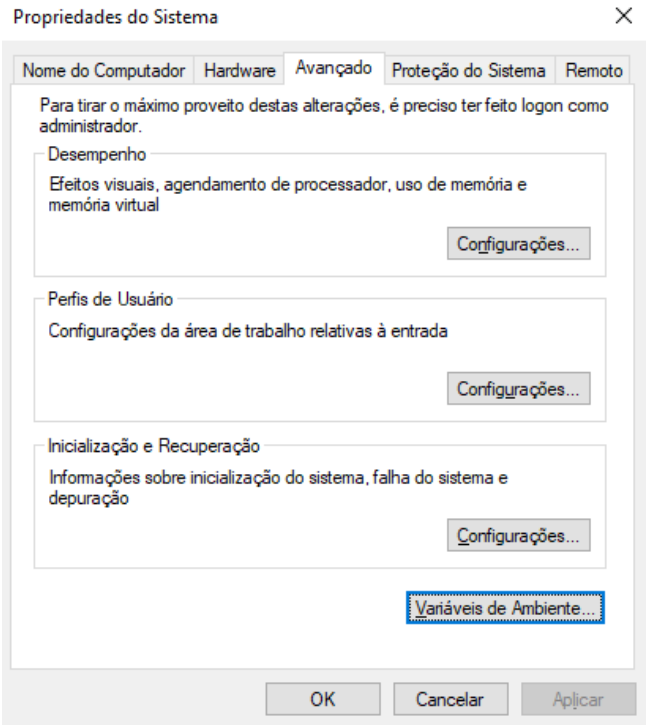
3. Instale o Spark
    1. O arquivo .tgz é uma extensão de arquivo compactado muito usado em linux e mac. Para descompactá-lo no windows, instale o 7-zip. Acesse www.7-zip.org/download
    2. Descompacte o arquivo .tgz
    3. Descompacte o arquivo .tar
    4. Mova o diretório para um local sem espaçamento ou acento
    5. Renomeie o diretório para “Spark”

Configurando Variáveis de Ambiente:

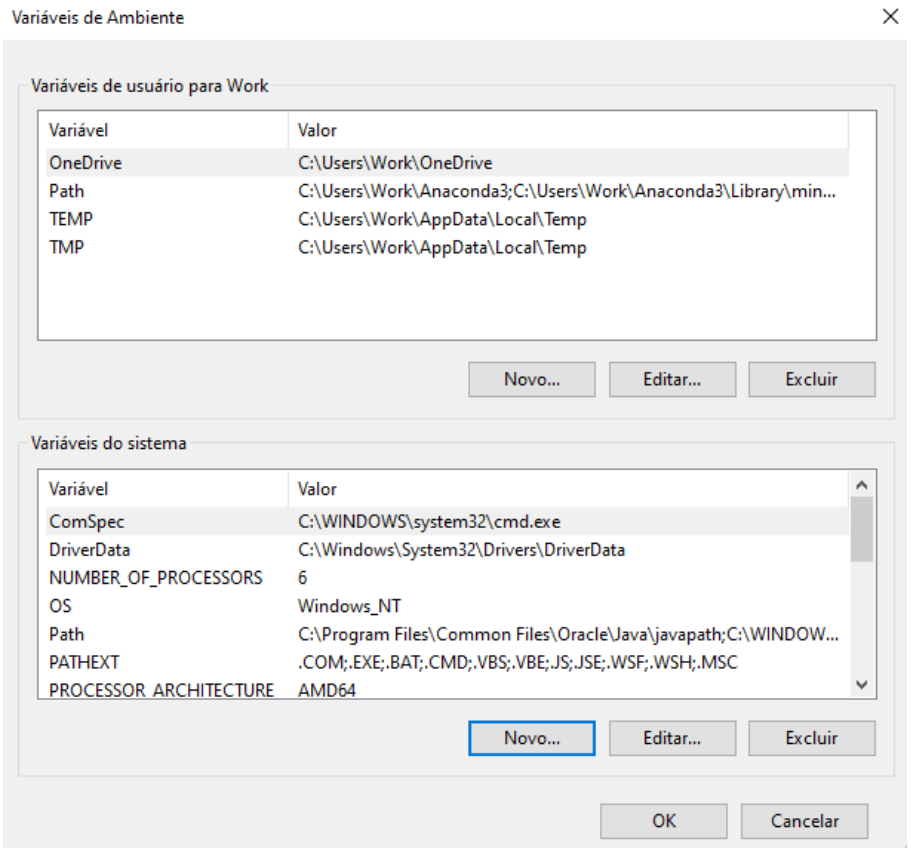
1. Clique em Iniciar > Executar > digite Control Panel > Procure por “avança” ou “adv” (se inglês)
2. Clique em Exibir configurações avançadas do sistema



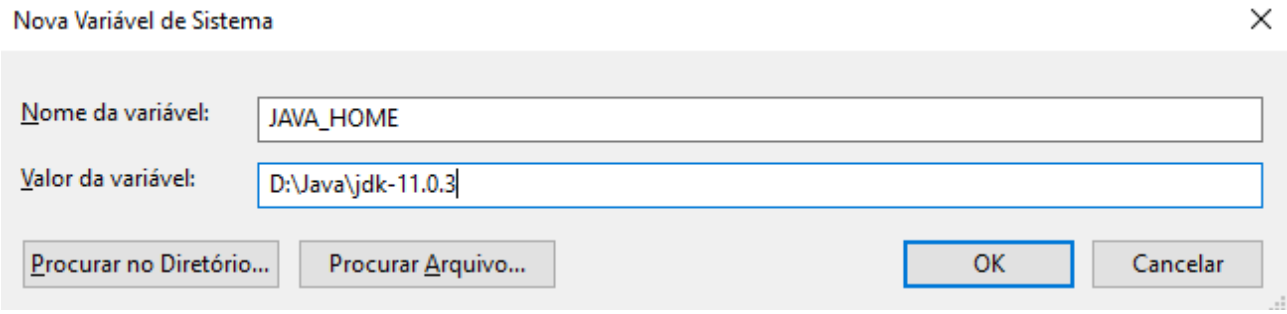
3. Clique em Variáveis de Ambiente



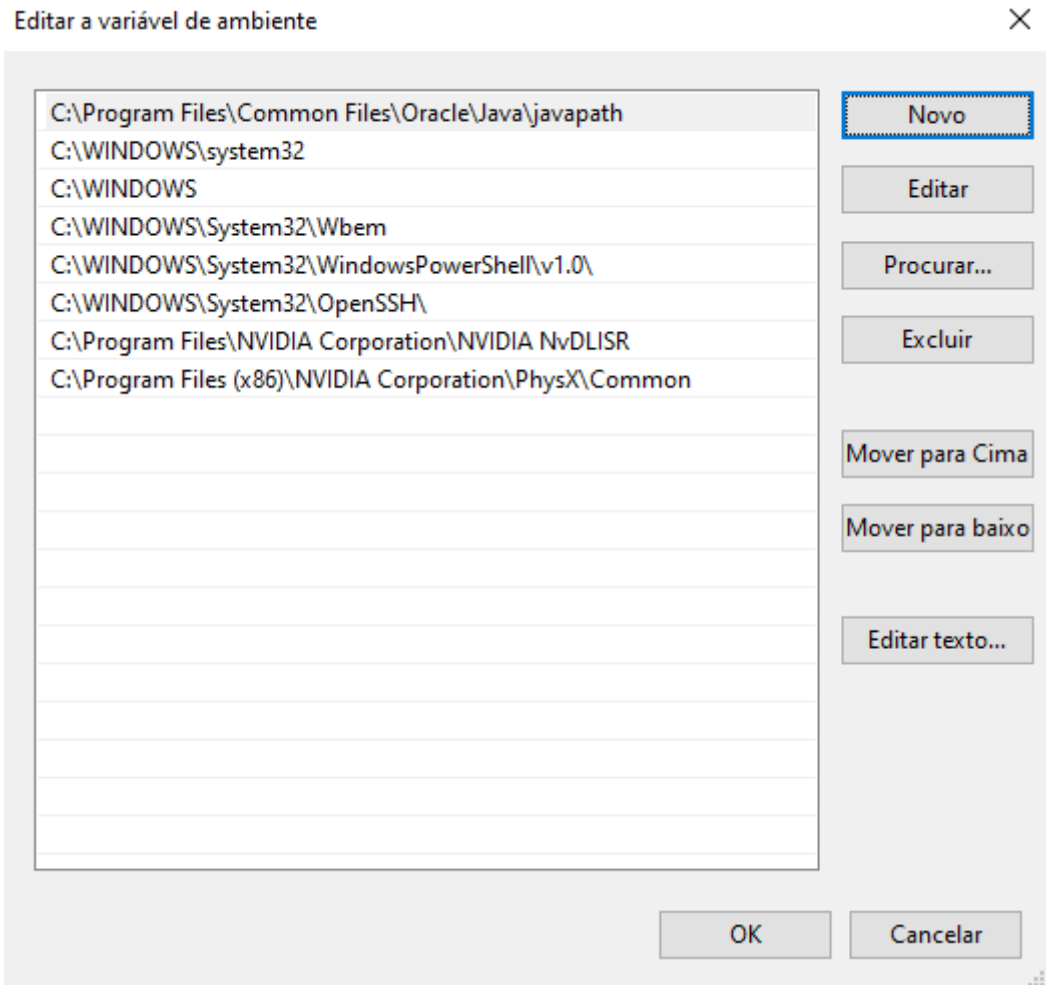
4. Em Variáveis do sistema, clique em Novo



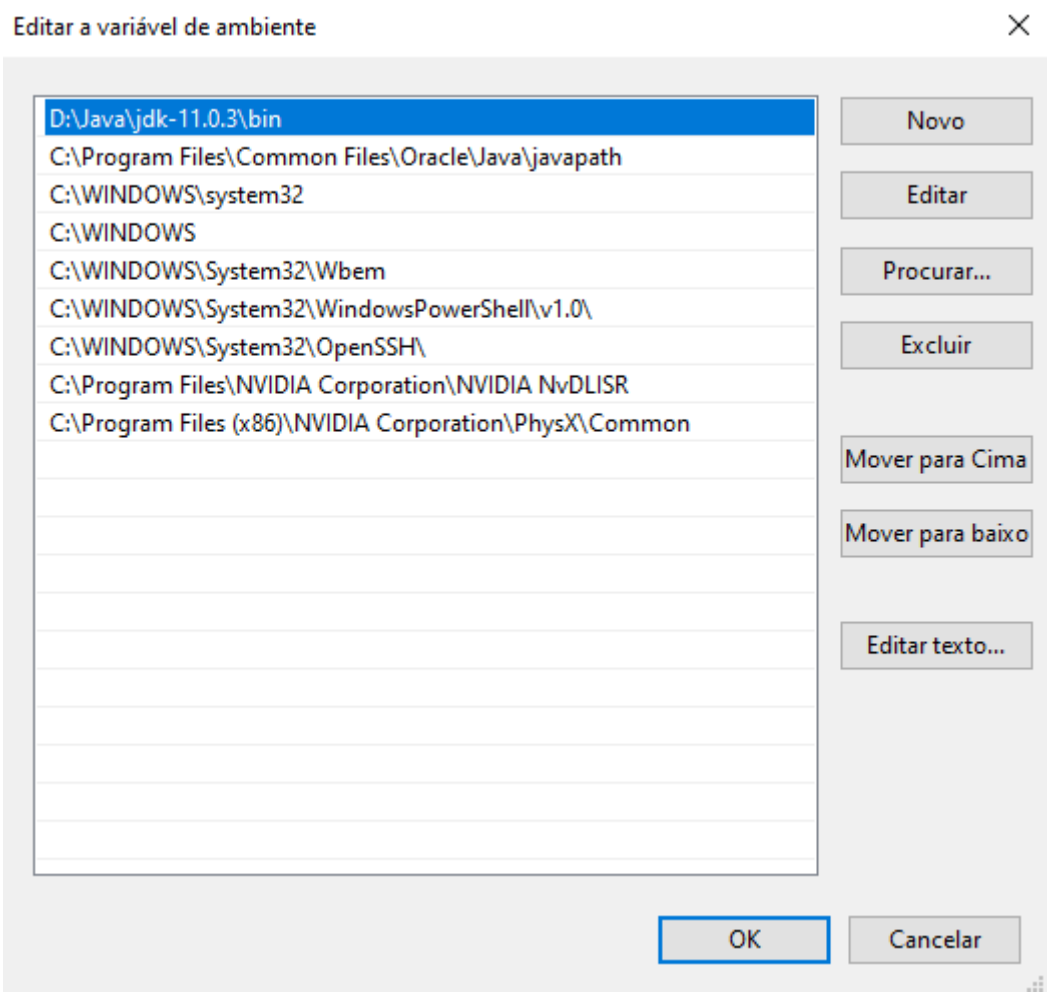
5. Informe a variável e o valor com o local onde o java foi instalado (utilize o arquivo de variáveis baixado no capítulo)



6. Edite a variável de sistema Path > clique em Novo

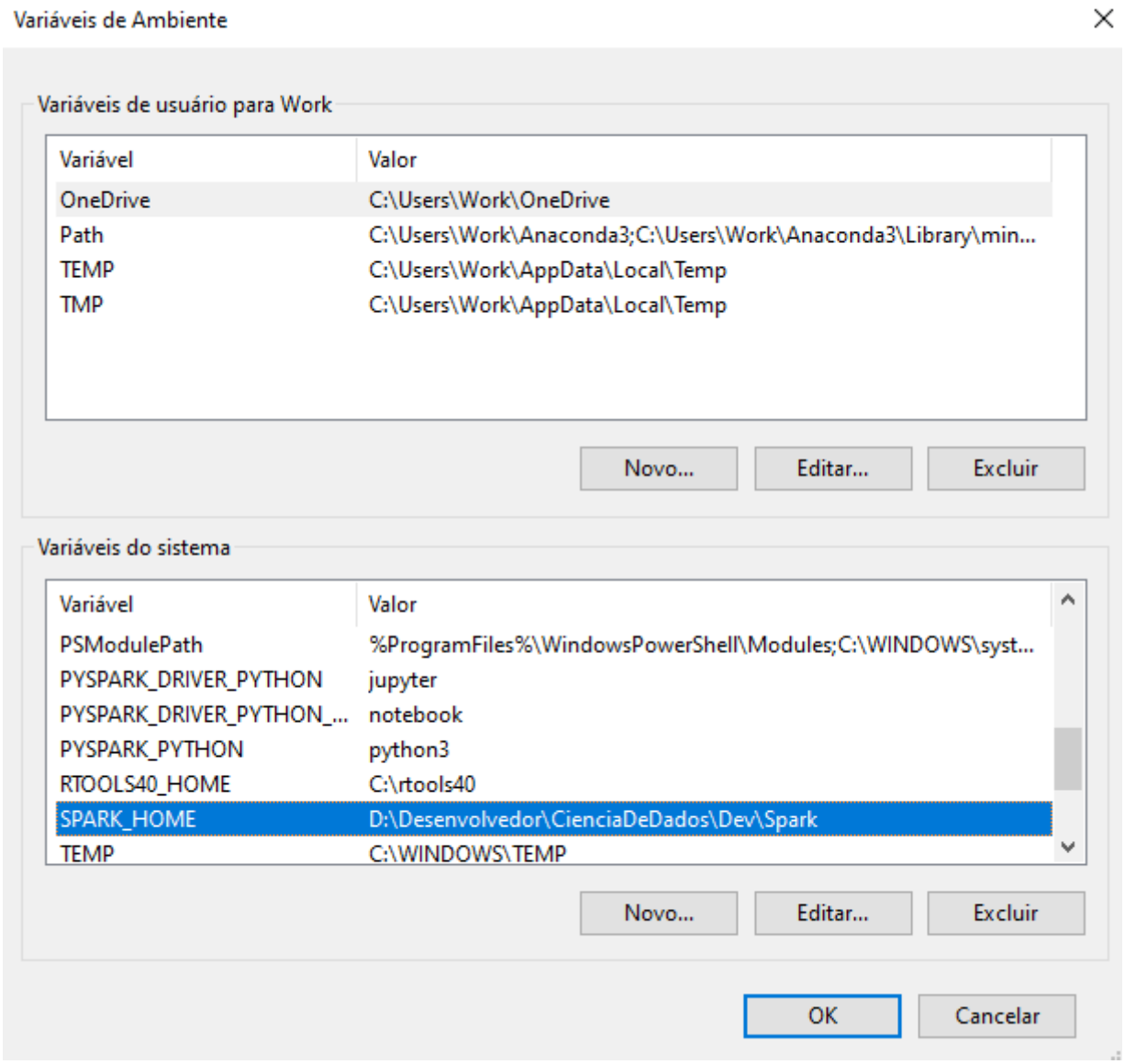


7. Adicione a pasta bin do Java (com os executáveis) e depois Mova para o início



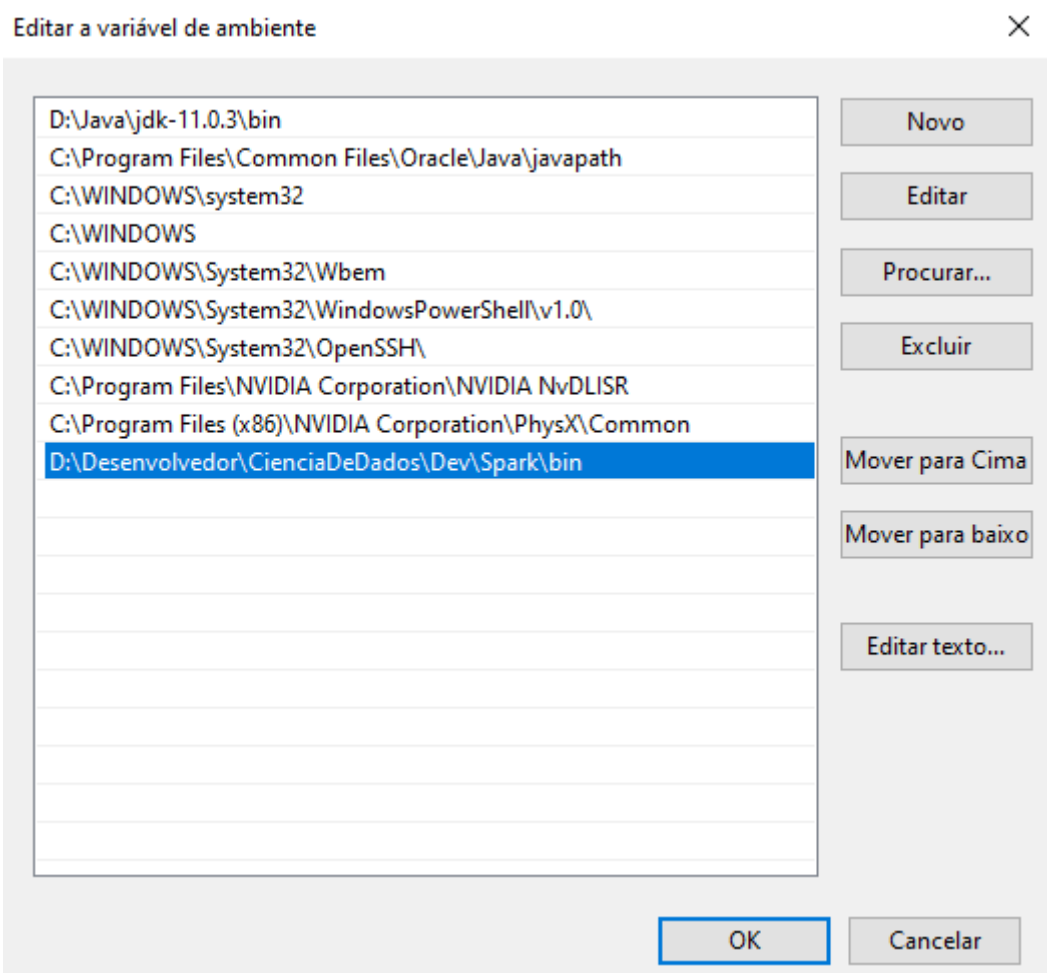
8. Faça os mesmos procedimentos para o Spark
    1. Crie as novas:

        SPARK_HOME = D:\Desenvolvedor\CienciaDeDados\Dev\Spark
        PYSPARK_DRIVER_PYTHON = jupyter
        PYSPARK_DRIVER_PYTHON_OPTS = notebook
        PYSPARK_PYTHON = python3



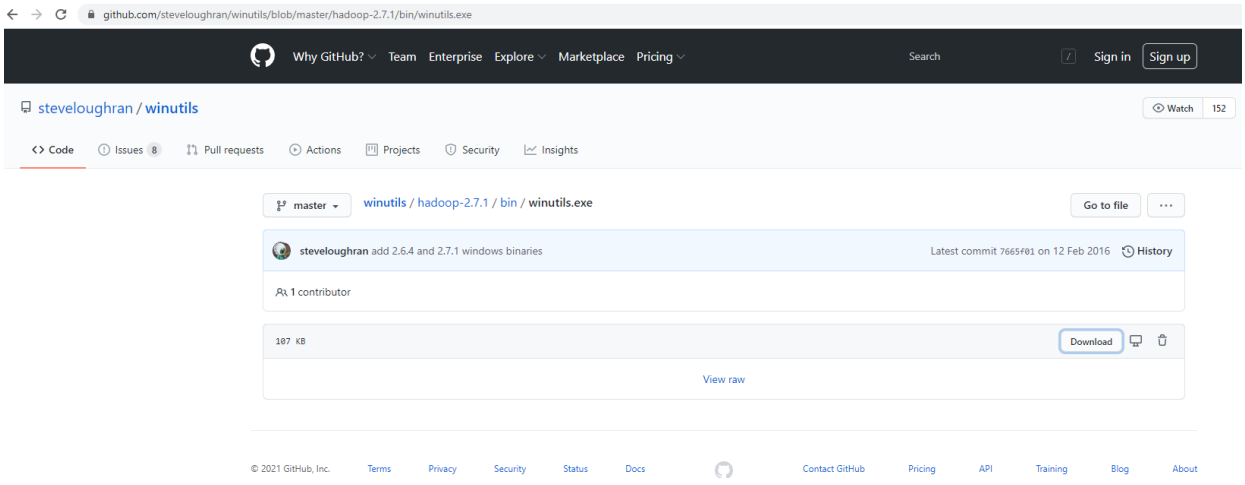
2. Edite o Path (não precisa mover para cima):

   PATH = D:\Desenvolvedor\CienciaDeDados\Dev\Spark\bin



Concluindo Configuração do Spark:

1. Instale o wintools para configurações de privilégio:
   1. Acesse # Wintools 64 bits: https://github.com/steveloughran/winutils/tree/master/hadoop-2.7.1/bin
   2. Ou acesse # Wintools 32 bits: https://drive.google.com/file/d/0B4PlPwU6yGTNT2FBdl9nTGNGOFk/view
   3. Clique em winutils.exe > Clique em Download



4. Crie o novo diretório, conforme abaixo:
    1. C:\Hadoop\bin (opção dada no curso)
    2. D:\Desenvolvedor\CienciaDeDados\Dev\Hadoop\bin (minha config)
5. Cole o arquivo do download na nova pasta
6. Configure as variáveis de ambiente
    1. Crie as Novas:

        HADOOP_HOME = D:\Desenvolvedor\CienciaDeDados\Dev\Hadoop
    2. Edite o Path (não precisa mover para cima):

        PATH = D:\Desenvolvedor\CienciaDeDados\Dev\Hadoop\bin
7. Crie o diretório, conforme abaixo:
    1. C:\tmp\hive (sugerido no curso)
    2. C:\tmp\hive
8. Abra o prompt e digite:
    1. C:\Hadoop\bin\winutils.exe chmod -R 777 C:\tmp\hive (sugestão do curso)
    2. D:\Desenvolvedor\CienciaDeDados\Dev\Hadoop\bin\winutils.exe chmod -R 777 D:\Desenvolvedor\CienciaDeDados\Dev\tmp\hive
    - onde:
        - chmod – change mode (muda o privilégio)
        - -R – modo recursivo
        - 777 – todo o privilégio
        - C:\tmp\hive – local ou diretório

Testando a Preparação do Ambiente

1. Abra o prompt de comando com “cmd”
2. Validando o Java:
    1. java -version (deve aparecer algo como: *java version "11.0.10" 2021-01-19 LTS*)
3. Validando o Python:
    1. python (deve aparecer algo como: *Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32*)
    2. exit() (para sair do shell do python)
4. Validando o Spark:
    1. spark-shell
        1. “:q” – para sair do shell
    2. pyspark
        1. ctrl+c – para sair do pyspark

5.2. Manipulação de Dados com Python

Introdução:

O que estudaremos neste capítulo?

- Módulos Python para Análise de Dados
- Estruturas de Dados
- NumPy
- Pandas
- Pré-Processamento

Preciso realmente aprender uma linguagem de programação para trabalhar como Cientista de Dados?

Você precisa aprender a usar uma ferramenta analítica!

Data Science é a junção de diferentes áreas de conhecimento:

- Matemática e Estatística
- Ciência da Computação
- Conhecimento da área de negócio

Mas então por que estou aprendendo linguagens de programação como ferramenta analítica?

- R e Python são linguagens gratuitas e podem ser usadas livremente, reduzindo custos com licenciamento de software, por exemplo.
- Embora R e Python requeiram mais trabalho manual para criar uma solução de análise, elas oferecem muito mais flexibilidade.
- Todo o conhecimento adquirido com R e Python, pode ser facilmente aplicado em outras soluções analíticas.
- Soluções Microsoft, Oracle e IBM, dão suporte a R e Python, como forma de estender as funcionalidades de suas soluções.
- As duas linguagens possuem uma grande e ativa comunidade e muita documentação disponível.

Por Que Linguagem Python Para Análise de Dados?

Cada vez mais dados para analisar (Big Data)

Cada vez menos tempo (Precisamos de Soluções de Análise em Tempo Real)

A boa notícia é que a linguagem Python pode nos ajudar em todas as etapas do processo de análise de dados!

- Fácil de aprender
- Muito popular
- Uso geral
- Linguagem interpretada
- Comunidade

Principais concorrentes da linguagem Python:

- R (linguagem gratuita)
- SAS (ferramenta proprietária)
- Matlab (ferramenta proprietária)
- Stata (ferramenta proprietária)
- C e C++
- Fortran
- Cython (http://cython.org) (visa utilizar python com C e C++)

Quando não usar Python?

Por ser uma linguagem interpretada, python é um pouco mais lenta do que linguagens como Java e C++, por exemplo.

Python não é ideal para aplicações com muita concorrência ou aplicações multi-thread, e isso pode ser um problema ao se trabalhar com Big Data.

Para trabalhar com Python, pode-se utilizar:

- IDE (Ambiente Integrado de Desenvolvimento)
  - PyCharm
  - Spyder
  - Canopy
  - WinPython
- Programação via browser
  - Jupyter Notebook
  - Jupyter Lab

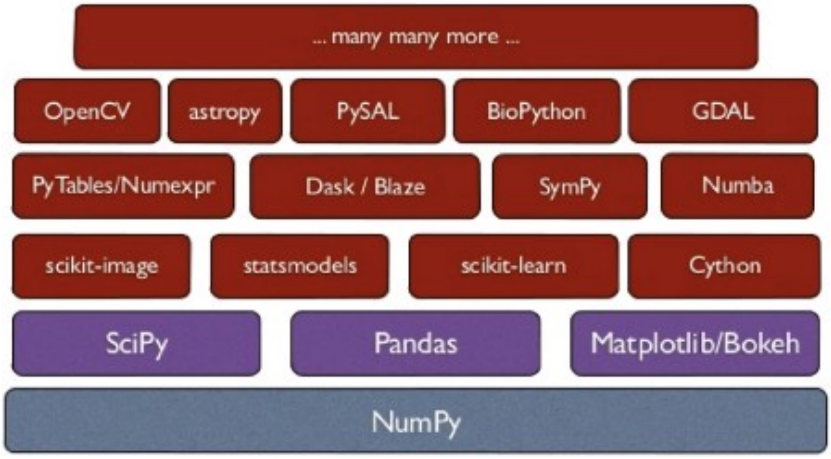
Principais Módulos Python Para Análise de Dados

Python, quanto linguagem de programação, possui seus comandos e funções bulti-in. É o que chamamos de linguagem pura.

Principais pacotes:

- NumPy (computação numérica / matemática)
- Pandas (manipulação e análise de dados)
- SciPy (computação científica)
- Matplotlib (visualização)
- Statsmodels (estatística)
- Scikit-learn (machine learning)

PyData Stack é um conjunto de pacotes python, específico para trabalhar com data science.



Principais vantagens de se utilizar o Pandas:

- Pode processar dados em diferentes formatos, como dados de séries temporais, matrizes, dados estruturados ou não estruturados.
- Facilita muito o trabalho de carga e importação de dados em arquivos csv ou de bancos de dados.
- Fornece funções para as mais variadas etapas de pré-processamento, como: subsetting, slicing, filtros, merge, agrupamentos, ordenação, reshape.
- Permite facilmente tratar dados missing.
- Pode ser usado para converter dados, bem como para aplicar modelagem estatística.
- É totalmente integrado com outros pacotes Python, como SciPy, NumPy e Scikit-Learn.

Uma rápida comparação entre linguagem R e Python:

R	Python
Vetores	Arrays Unidimensionais (NumPy)
Matrizes	Arrays Multidimensionais (NumPy)
DataFrames	DataFrames (Pandas)

Preparando o Ambiente de Análise de Dados

1. Crie um diretório para cada capítulo do curso
2. Abra o prompt de comando
    1. Iniciar > “cmd”
3. Acesse o diretório do capítulo
    1. Digite cd “diretório/diretório”
    2. Utilize a tecla tab para completar o nome do arquivo/diretório enquanto escreve

    Exemplo:

    d/Desenvolvedor/CienciaDeDados/Estudos/DSA-Cursos/FCD/05_BDataRealTimeAnalyticsComPythonSpark/Cap02
4. Abra o jupyter notebook
    1. Digite jupyter notebook
5. O jupyter abrirá diretamente no seu navegador
6. Para desligar/shutdown
    1. Tecle control+c
    2. Digite y
7. Para digitar os comandos usando o Jupyter
    1. Clique em New > Python 3
8. Manual do Jupyter Notebook
    1. https://www.youtube.com/embed/JF5hQ9M2YEY

Quizz Cap03

Séries temporais são conjuntos de dados gerados ao longo do tempo.

5.4. Análise Estatística de Dados

Definindo Estatística

O que é Estatística?

É a ciência que nos permite aprender a partir dos dados.

Com a Estatística nós podemos:

- Coletar
- Organizar
- Apresentar
- Descrever
- Interpretar os Dados

Tipos de Dados

Qualitativos (Categóricos)		**Quantitativos (Numéricos)**	
Nominais (sem hierarquia)	Ordinais (com hierarquia)	Discretos (contagem e finitos)	Contínuos (pode assumir qualquer valor no range)
Profissão, Sexo, Religião	Escolaridade, Classe Social, Fila	Número de Filhos, Número de carros, Número de acessos	Altura, Peso, Salário

Dados Qualitativos Nominais – representam descrições para os dados e não permitem ranqueamento. Exemplo: CEP (70.098-080).

Dados Qualitativos Ordinais – existe uma ordenação entre as categorias (ranqueamento) e os dados podem ser medidos.

Dados Quantitativos Discretos – valores baseados em observações que podem ser contados, normalmente representados por valores inteiros.

Dados Quantitativos Contínuos – valores baseados em observações que podem ser medidas e normalmente representados por valores decimais.

Observação x Experimentação

Há dois tipos de estudos estatísticos:

- Observacional
    - Em um estudo de observação, os dados e as características específicas são recolhidos e observados, entretanto, não há iniciativa de modificar os estudos que estão sendo realizados.

- Experimental
    - Em um estudo experimental, cada indivíduo é aleatoriamente atribuído a um grupo de tratamento, em seguida, os dados e as características específicas são observados e coletados.

A Análise de Dados é o meio através do qual utilizamos a estatística para apresentar e demonstrar os resultados dos dados que foram avaliados.

Estatística não tem sido usada apenas por técnicos, mas também por gestores de todos os níveis. Para onde se olha, se vê Estatística sendo aplicada, desde o planejamento corporativo, até decisões simples do dia a dia.

Principais Áreas da Estatística

- Estatística Descritiva
- Probabilidade
- Estatística Inferencial

Estatística Descritiva

É um conjunto de métodos estatísticos utilizados para descrever as principais características dos dados.

O principal propósito de métodos gráficos é organizar e apresentar os dados de forma gerencial e ágil.

A Estatística Descritiva tem por objetivo sumarizar e mostrar os dados, de forma que se possa rapidamente obter uma visão geral da informação que está sendo analisada.

Por meio da Estatística Descritiva entendemos melhor um conjunto de dados através de suas características.

As três principais características são:

- Um valor representativo do conjunto de dados. Ex.: a média.
- Uma medida de dispersão ou variação. Ex: variância, desvio padrão.
- A natureza ou forma da distribuição dos dados: sino, uniforme ou assimétrica.

Tabela de Frequência

Um dos meios mais simples de descrever dados é através de tabelas de frequência, que refletem as observações feitas nos dados.

Número de tablets vendidos por dia	Frequência	
0	5	Classes
1	8	
2	14	
3	13	
4	6	

Cada linha em uma tabela de frequência corresponde a uma classe.

Cada classe corresponde a uma categoria em uma tabela de frequência.

Distribuição de Frequência

Uma Distribuição de Frequência mostra o número de observações de dados que estão em um intervalo específico.

Como construir uma Distribuição de Frequência?

1. Criar o Rol
2. Definir a Amplitude
3. Determinar o Número de Classes
4. Determinar o Tamanho do Intervalo de Classes
5. Construir a Distribuição de Frequência

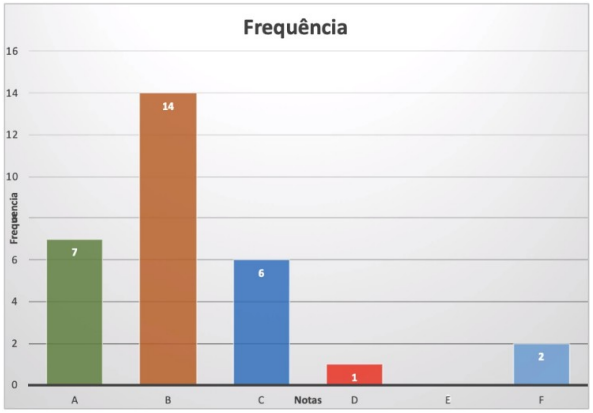
Ferramentas Oferecidas Pela Estatística Descritiva

Quais as principais ferramentas e/ou elementos usados na Estatística Descritiva?

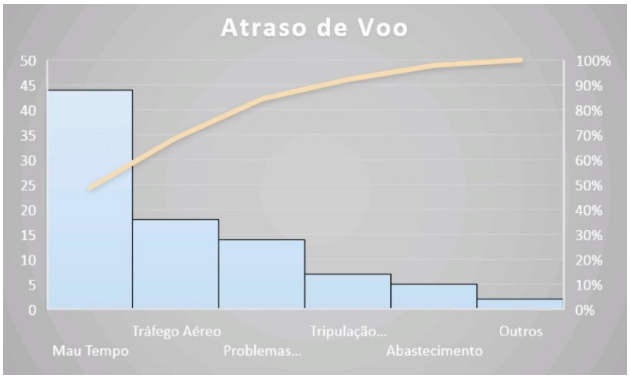
- Análise Univariada (apenas uma variável)
  - Tabela de Frequência

Notas	Frequência
A	7
B	14
C	6
D	1
E	0
F	2

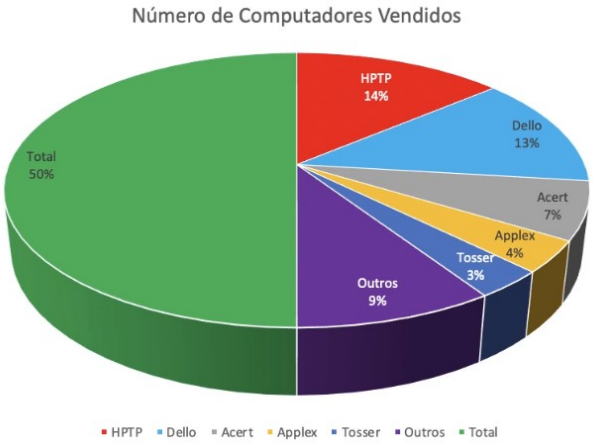
  - Gráfico de Barras – importante garantir que a escala esteja apropriada.



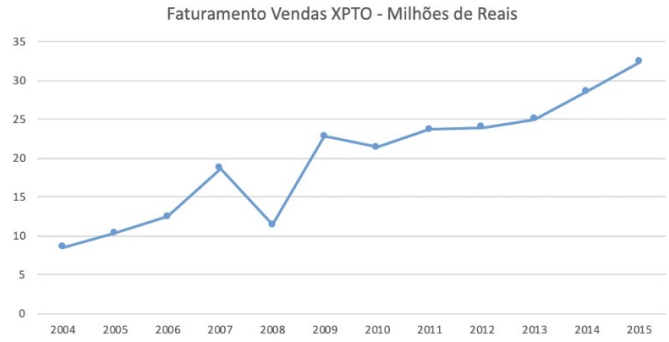
  - Gráfico de Pareto



  - Gráfico de Pizza



- Gráfico de Linha

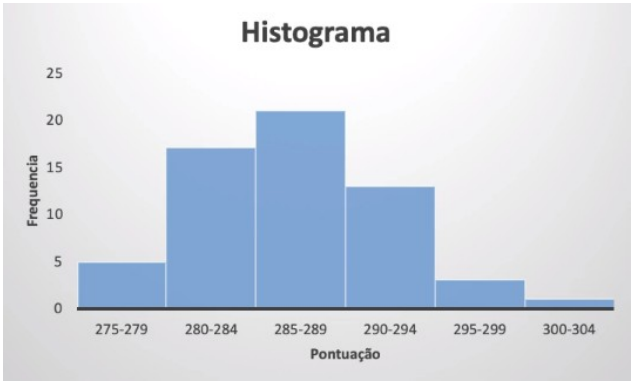


- Caule e Folha – divide os dados em duas partes:
  - O caule (ramo) representa os valores maiores e ficam à esquerda do traço vertical.
  - As folhas são os menores valores, ficam à direita do traço vertical. Listando todas folhas à direita de cada caule, podemos graficamente descrever como os dados estão distribuídos.

Diâmetros abdominais de 40 indivíduos

Ramo (dezena)	Folhas (unidades)
5	7 9
6	0 0 2 3 3 3 4 6 6 8 9 9
7	0 0 1 2 2 3 4 5 5 7 8
8	1 3 5 6 6 7 8 8 9
9	1 4 5
10	1 7
11	9

- Histograma – mostra distribuição de frequência



- Análise Bivariada (duas variáveis)
  - Tabela de Contingência

Cliente	Sexo	Condição de Pagamento
1	Feminino	Dinheiro
2	Masculino	Cartão
3	Masculino	Dinheiro
4	Masculino	Dinheiro
5	Feminino	Cartão
6	Feminino	Cartão
7	Masculino	Dinheiro
8	Feminino	Cartão
9	Masculino	Cartão
10	Feminino	Dinheiro
11	Masculino	Cartão
12	Feminino	Cartão
13	Masculino	Dinheiro
14	Feminino	Cartão
15	Feminino	Dinheiro

Soma de Cliente	Rótulos de Coluna		
Rótulos de Linha	Cartão	Dinheiro	Total Geral
Feminino	45	26	71
Masculino	22	27	49
Total Geral	**67**	**53**	**120**

  - Gráfico de Dispersão

Tamanho da TV LED	Preço da TV R$
46	2600
46	3980
32	1200
40	1480
26	970
32	1115
46	3400
46	5560
32	2400
40	1120
26	1130
32	1320

Medidas de Tendência Central

Estas são as principais medidas de tendência central utilizadas em Estatística Descritiva:

Média (Mean ou Average em inglês) é uma medida de tendência central dos dados, ou seja, um número em torno do qual um dataset inteiro está distribuído. É um número único que pode estimar o valor do conjunto de dados completo.

Médias são as formas mais simples de identificar tendências em um conjunto de dados. Entretanto, médias podem trazer armadilhas que levam a conclusões distorcidas.

Mediana é o valor que divide os dados em 2 partes iguais, ou seja, o número de termos no lado direito é igual ao número de termos no lado esquerdo quando os dados são organizados em ordem crescente ou decrescente.

A **Mediana** será um elemento do meio da distribuição, se o número de termos for ímpar.

A **Mediana** será a média de 2 elementos do meio da distribuição, se o número de termos for par.

A **Moda** é o termo que aparece mais vezes no conjunto de dados, ou seja, o termo que tem a frequência mais alta.

Mas pode haver um conjunto de dados em que não há nenhuma **Moda**, pois todos os valores aparecem o mesmo número de vezes.

Se dois valores aparecerem ao mesmo tempo e mais do que o resto dos valores, o conjunto de dados será **bimodal**. Se três valores aparecerem no mesmo tempo e mais do que o resto dos valores, o conjunto de dados é **trimodal** e, para n modas, esse conjunto de dados é **multimodal**.

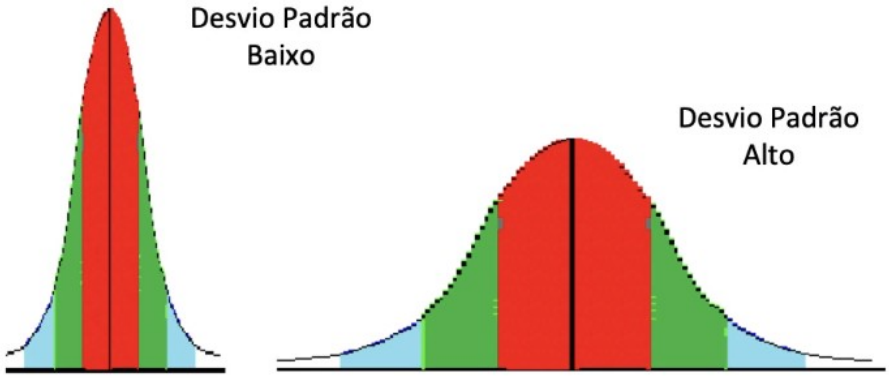
O que usar?	**Vantagens**	**Desvantagens**
Média	Relativamente fácil de calcular Fácil de compreender seu significado	Pode ser muito afetada por valores extremos
Mediana	Não é afetada por valores extremos	Requer mais esforço para ser determinada que a Média
Moda	Pode ser usada com dados descritivos	Pode não existir em um conjunto de dados Pode não ser única (pode existir mais de uma moda)

Medidas de Dispersão

Medidas de Dispersão referem-se à variabilidade dentro do conjunto de dados.

Desvio Padrão (Standard Deviation)

O desvio padrão é a medida da distância média entre cada elemento e a média. Isto é, como os dados são distribuídos a partir da média. Um desvio padrão baixo indica que os pontos de dados tendem a estar próximos da média do conjunto de dados, enquanto um desvio padrão alto indica que os pontos de dados estão espalhados em uma faixa mais ampla de valores.



Variância (Variance)

A variância é o quadrado do desvio padrão.

Intervalo (Range)

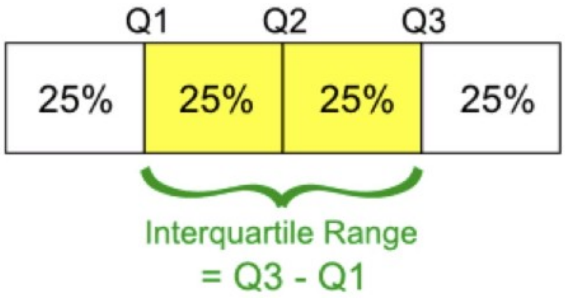
Intervalo é uma das técnicas mais simples de estatística descritiva. É a diferença entre o menor e o maior valor do conjunto de dados.

Percentil

O percentil é uma maneira de representar a posição de um valor no conjunto de dados. Para calcular o percentil, os valores no conjunto de dados devem estar sempre em ordem crescente.

Quartil

Os quartis são valores que dividem os dados em quarters, desde que os dados sejam classificados em ordem crescente.



Medidas de Forma

As medidas de assimetria (skewness) e curtose (kurtosis) caracterizam a forma da distribuição de elementos em torno da média.

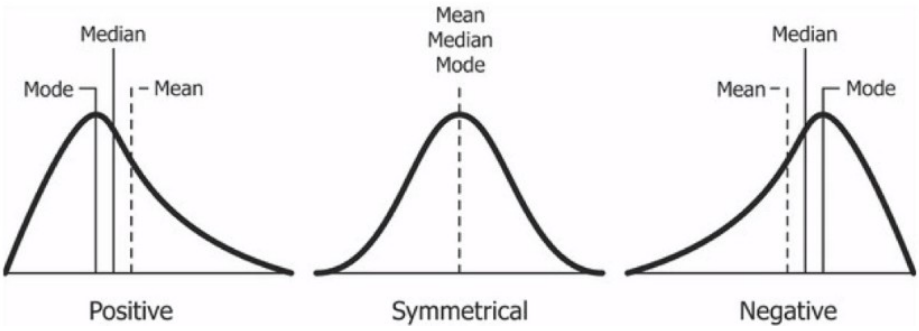
Assimetria (Skewness)

Skewness é uma medida da assimetria da distribuição de probabilidade de uma variável aleatória de valor real sobre sua média. O valor da assimetria pode ser positivo, negativo ou indefinido

Em uma distribuição normal perfeita, as caudas de cada lado da curva são imagens espelhadas exatas uma da outra.

Quando uma distribuição é inclinada para a direita, a cauda no lado direito da curva é maior que a cauda no lado esquerdo, e a média é maior que a moda. Essa situação também é chamada de assimetria positiva.

Quando uma distribuição é inclinada para a esquerda, a cauda do lado esquerdo da curva é maior que a cauda do lado direito e a média é menor que a moda. Essa situação também é chamada de assimetria negativa.



Para calcular o coeficiente de assimetria, usamos:

$$\frac{\text{Mean - Mode}}{\text{Standard Deviation}}$$

First Coefficient of Skewness
(Mode skewness)

$$\frac{\text{3 (Mean - Median)}}{\text{Standard Deviation}}$$

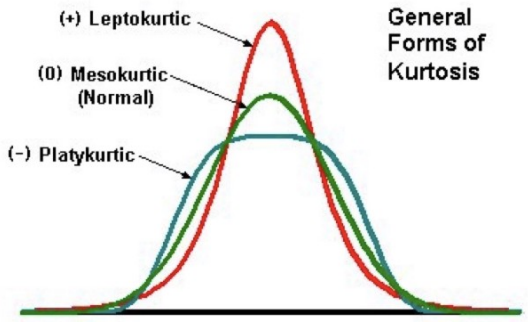
Second Coefficient of Skewness
(Median skewness)

- A direção da assimetria é dada pelo sinal. Um zero significa nenhuma assimetria.
- Um valor negativo significa que a distribuição é negativamente assimétrica. Um valor positivo significa que a distribuição está positivamente assimétrica.

- O coeficiente compara a distribuição da amostra com uma distribuição normal. Quanto maior o valor, mais a distribuição difere de uma distribuição normal.

Curtose (Kurtosis)

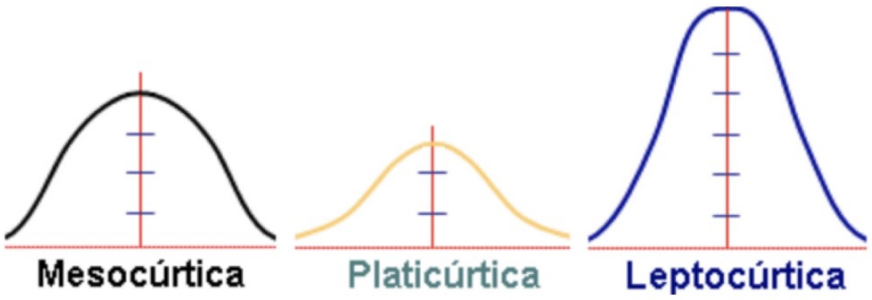
Um dos coeficientes mais utilizados para medir o grau de achatamento ou curtose de uma distribuição é o coeficiente percentílico de curtose, ou simplesmente coeficiente de curtose (k), calculado a partir do intervalo interquartil dos percentis de ordem 10 e 90.



Quando a forma da distribuição não é nem muito achatada e nem muito alongada, com uma aparência semelhante à da curva normal, é denominada mesocúrtica.

Por outro lado, quando a distribuição apresenta uma curva de frequências mais achatada que a curva normal é denominada platicúrtica. Apresenta uma medida de curtose menor que a da distribuição normal.

Ou ainda, quando a distribuição apresenta uma curva de frequências mais alongada que a curva normal é denominada leptocúrtica. Apresenta uma medida de curtose maior que a da distribuição normal.



Se k = 0,263 → dizemos que a distribuição é mesocúrtica

Se k > 0,263 → dizemos que a distribuição é platicúrtica

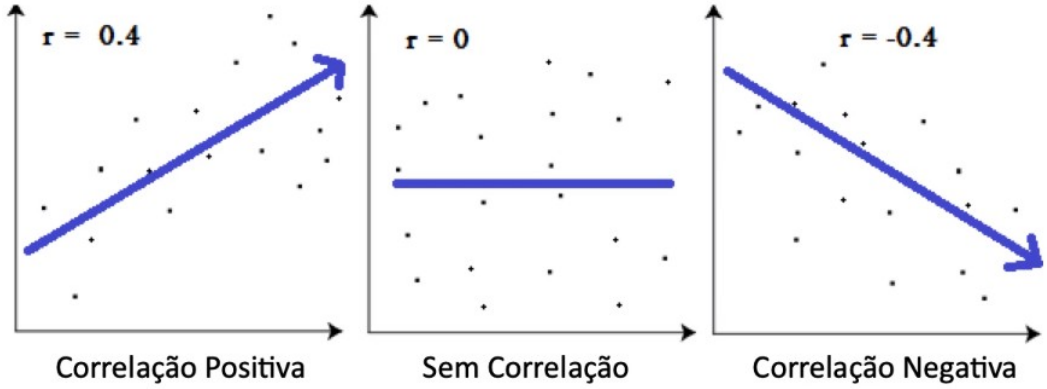
Se k < 0,263 → dizemos que a distribuição é leptocúrtica

$$K = \frac{\frac{1}{2}(Q_3 - Q_1)}{P_{90} - P_{10}} \text{ — } 0{,}263$$

Coeficiente de Correlação

A Correlação permite determinar quão fortemente os pares de variáveis estão relacionados.

O principal resultado de uma correlação é chamado de coeficiente de **correlação** (ou “r”). Varia de -1.0 a +1.0. Quanto mais próximo r for +1 ou - 1, mais próximas as duas variáveis estarão relacionadas.



Quizz

Um dos meios mais simples de descrever dados é através de tabelas de frequência, que refletem as observações feitas nos dados.

5.5. Análise Estatística de Dados – 2

Introdução à Probabilidade

A probabilidade é um número que varia de 0 (zero) a 1 (um) e que mede a chance de ocorrência de um determinado resultado. Quanto mais próxima de zero for a probabilidade, menores são as chances de ocorrer o resultado e quanto mais próxima de um for a probabilidade, maiores são as chances.

As probabilidades podem ser expressas de diversas maneiras, inclusive decimais, frações e percentagens. Por exemplo, a chance de ocorrência de um determinado evento pode ser expressa como 10%; 5 em 10; 0,20 ou 1/7.

A teoria da probabilidade consiste em utilizar a intuição humana para estudar os fenômenos do nosso cotidiano. Para isso, vamos utilizar o princípio básico do aprendizado humano que é a ideia de experimento.

Podemos classificar os experimentos em dois tipos:

- Aleatórios (casuais)
- Não aleatórios (determinísticos)

Os experimentos determinísticos são totalmente caracterizados a priori, ou seja, são fenômenos em que o resultado é sabido antes mesmo em que ele ocorra e desta forma, nada temos a fazer.

Os experimentos que iremos estudar são os aleatórios, dos quais não sabemos o resultado a priori.

Um experimento é dito aleatório quando não conseguimos afirmar o resultado que será obtido antes de realizar o experimento.

Experimento Aleatório

Experimento aleatório é o fenômeno que, quando repetido inúmeras vezes em processos semelhantes, possui resultados imprevisíveis.

O lançamento de um dado e de uma moeda são considerados exemplos de experimentos aleatórios. No caso dos dados podemos ter seis resultados diferentes {1, 2, 3, 4, 5, 6} e no lançamento da moeda, dois {cara, coroa}.

Experimento é qualquer atividade realizada que pode apresentar diferentes resultados. Um experimento é dito aleatório quando não conseguimos afirmar o resultado que será obtido antes de realizar o experimento. Um experimento é dito equiprovável se todos os possíveis resultados possuem a mesma chance de ocorrer.

Tipos de Probabilidade

Evento – um ou mais resultados de um experimento.

O resultado e/ou resultados são um subconjunto do espaço da amostra.

Probabilidade Clássica: é usada quando nós sabemos o número de possíveis resultados do evento de interesse e podemos calcular a probabilidade do evento com a seguinte fórmula:

P(A) = Número de possíveis resultados do evento A / Número total de possíveis resultados dentro do espaço da amostra

Onde: P(A) é a probabilidade de um evento ocorrer.

A **Probabilidade Empírica**, envolve conduzirmos um experimento, para observarmos a frequência com que um evento ocorre.

Para calcularmos a probabilidade empírica, usamos a fórmula:

P(A) = Frequência em que o evento A ocorre / Número total de observações

Usamos **Probabilidade Subjetiva** quando dados ou experimentos não estão disponíveis para calcular a probabilidade.

Regras da Probabilidade

1. Regra:

   Se P(A) = 1, então podemos garantir que o evento A ocorrerá.

2. Regra:

   Se P(A) = 0, então podemos garantir que o evento A NÃO ocorrerá.

3. Regra:

   A probabilidade de qualquer evento sempre será entre 0 e 1.

   Probabilidades nunca podem ser negativas ou maior que 1.

4. Regra:

   A soma de todas as probabilidades para um evento simples, em um espaço de amostra, será igual a 1.

5. Regra:

   O complemento do evento A é definido como todos os resultados em um espaço de amostra, que não fazem parte do evento A. Ou seja:

   P(A) = 1 – P(A'), onde P(A') é o complemento do evento A.

Eventos e Espaço Amostral

O primeiro elemento na modelagem de um experimento é o espaço amostral, que consiste no conjunto de todos os possíveis resultados do experimento.

Espaço Amostral é o conjunto de possíveis resultados de um experimento ou fenômeno aleatório, representado por S.

Evento é qualquer subconjunto do espaço amostral S de um evento aleatório.

S = {defeituoso, não defeituoso}

S = {Sim, Não}

S = {1, 2, 3, 4, 5, 6}

S = {cara, coroa}

Eventos Complementares

Sabemos que um evento pode ocorrer ou não. Sendo p a probabilidade de que ele ocorra (sucesso) e q a probabilidade de que ele não ocorra (insucesso), para um mesmo evento existe sempre a relação:

$$p + q = 1 \rightarrow q = 1 - p$$

Eventos Independentes (Regra do “e”)

Dizemos que dois eventos são independentes quando a realização ou não realização de um dos eventos não afeta a probabilidade da realização do outro e vice-versa.

Assim, sendo p1 a probabilidade de realização do primeiro evento e p2 a probabilidade do segundo evento, a probabilidade de que tais eventos se realizem simultaneamente é dada por:

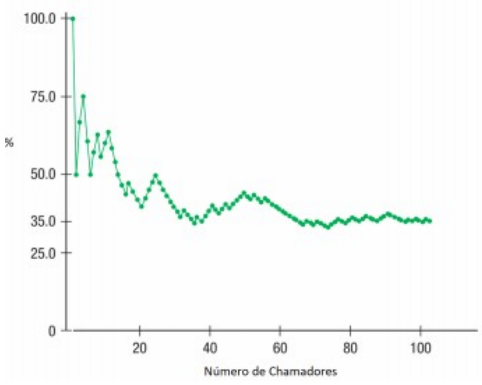
$$p = p1 \times p2$$

O Fenômeno Aleatório

Quando um cliente liga para o número 0-800 de uma empresa de cartão de crédito, ele é solicitado a fornecer um número de cartão antes de ser conectado a um operador. À medida que a conexão é feita, os registros de compra desse cartão e as informações demográficas do cliente são recuperados e exibidos na tela do operador.

Se a pontuação (score) do cliente for alta o suficiente, o operador pode ser solicitado a “vender” outro serviço - talvez um novo cartão “platinum” para clientes com uma pontuação de crédito de pelo menos 750. Claro, a empresa não sabe quais clientes vão ligar. As chegadas de chamadas são um exemplo de um fenômeno aleatório.

Com fenômenos aleatórios, não podemos prever os resultados individuais, mas podemos esperar compreender as características dos seus comportamentos de longo prazo. Não sabemos se o próximo cliente se qualificará para o cartão platinum, mas quando as chamadas entrarem no call center, a empresa descobrirá que a porcentagem de chamadores qualificados para o cartão platinum se estabelecerá em um padrão, como mostrado no gráfico da figura abaixo.



À medida que as chamadas chegam ao call center, a empresa pode registrar se cada chamador se qualifica. O primeiro chamador hoje é qualificado. Em seguida, as cinco qualificações dos próximos chamadores foram não, sim, sim, não e não. Se traçarmos a porcentagem que se qualifica em relação ao número da chamada, o gráfico começaria em 100% porque o primeiro chamador se qualificou (um de um, para 100%). O próximo chamador não se

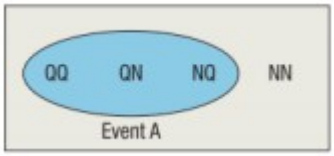
qualificou, então o percentual acumulado caiu para 50% (um em cada dois). O terceiro chamador qualificou-se (dois de três ou 67%) e assim por diante. A cada nova chamada, o novo dado é uma fração menor da experiência acumulada, de modo que, no longo prazo, o gráfico se estabiliza. À medida que se instala, parece que, de fato, a fração de clientes que se qualifica é de cerca de 35%.

Quando se fala em comportamento de longo prazo, ajuda a definir nossos termos. Para qualquer fenômeno aleatório, cada tentativa ou teste gera um resultado. Para o call center, cada chamada é uma tentativa. Algo acontece em cada tentativa, e chamamos o que quer que aconteça de resultado. Aqui, o resultado é se o chamador se qualifica ou não. Usa-se o termo mais geral para se referir a resultados ou combinações de resultados.

Por exemplo, suponha que categorizemos os chamadores em seis categorias de risco e enumeremos esses resultados de 1 a 6 (aumentando a capacidade de obtenção de crédito). Os três resultados 4, 5 ou 6 podem compor o evento "o chamador é pelo menos uma categoria 4".

Às vezes falamos sobre a coleta de todos os resultados possíveis, um evento especial ao qual nos referimos como o espaço de amostra. Denotamos o espaço amostral com a letra S. Mas qualquer que seja o símbolo que usamos, o espaço amostral é o conjunto que contém todos os resultados possíveis. Para as chamadas, se deixarmos Q = qualificado e N = não qualificado, o espaço de amostragem será simples: S = {Q, N}.

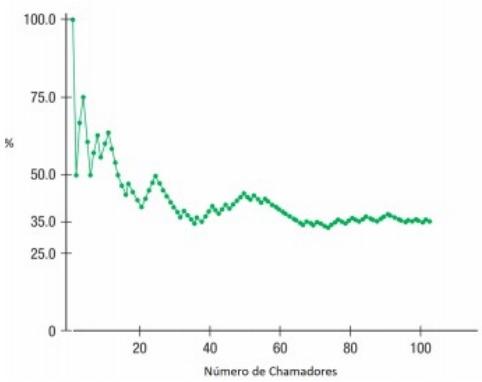
Se olharmos para duas chamadas juntas, o espaço amostral tem quatro resultados: S = {QQ, QN, NQ, NN}. Se estivéssemos interessados em pelo menos um chamador qualificado das duas chamadas, estaríamos interessados no evento (chame de A) consistindo nos três resultados QQ, QN e NQ, e escreveríamos A = {QQ, QN, NQ} conforme figura abaixo:



Probabilidade Empírica e a Lei dos Grandes Números

Embora possamos não ser capazes de prever um determinado resultado individual, como qual chamada recebida representa uma potencial venda, podemos dizer muito sobre o comportamento de longo prazo.

Olhe mais uma vez para a imagem abaixo. Se você for perguntado sobre a probabilidade de que um chamador aleatório se qualifique, você pode dizer que é de 35%, porque, no longo prazo, a porcentagem dos chamadores que se qualificam é de cerca de 35%.



E é exatamente isso que queremos dizer com probabilidade. Como se baseia na observação repetida do resultado do evento, essa definição de probabilidade é geralmente chamada de probabilidade empírica.

Isso realmente simplifica as coisas se as tentativas individuais forem independentes. Grosso modo, independência significa que o resultado de um teste não influencia ou altera o resultado de outro. Lembre-se dos capítulos anteriores em que chamamos duas variáveis independentes se o valor de uma variável categórica não influenciou o valor de outra variável categórica (verificamos a independência comparando as distribuições de frequência relativa entre as variáveis.) Não há razão para pensar que a qualificação de um chamador influencia a qualificação de outro chamador, portanto, esses são testes independentes.

Felizmente, para eventos independentes, podemos depender de um princípio chamado Lei dos Grandes Números, que afirma que, se os eventos forem independentes, quando o número de chamadas aumentar, ao longo de dias ou meses ou anos, a longo prazo a frequência relativa de chamadas qualificadas fica cada vez mais próxima de um único valor. Isso nos dá a garantia de que precisamos e torna a probabilidade um conceito útil.

Como a Lei dos Grandes Números garante que frequências relativas se estabelecem a longo prazo, podemos dar um nome ao valor que elas abordam. Nós chamamos isso de a probabilidade desse evento. Para o call center, podemos escrever P (qualificado) = 0,35. Como se baseia na observação repetida do resultado do evento, essa definição de probabilidade é frequentemente chamada de probabilidade empírica.

Um Pouco Mais Sobre os Tipos de Probabilidade

Nós discutimos a probabilidade empírica - a frequência relativa da ocorrência de um evento como a probabilidade de um evento. Existem outras maneiras de definir a probabilidade também. A probabilidade foi estudada extensivamente por um grupo de matemáticos franceses interessados em jogos de azar. Em vez de experimentar os jogos e correr o risco de perder seu dinheiro, eles desenvolveram modelos matemáticos de probabilidade. Para simplificar as coisas (como geralmente fazemos quando construímos modelos), eles começaram examinando jogos nos quais os diferentes resultados eram igualmente prováveis.

Felizmente, muitos jogos de azar são assim. Qualquer uma das 52 cartas tem a mesma probabilidade de ser a próxima distribuída de um baralho bem embaralhado. Cada face de um dado tem a mesma probabilidade de pousar (ou pelo menos deveria ser). Quando os resultados são igualmente prováveis, sua probabilidade é fácil de calcular - é apenas uma divisão pelo número de

resultados possíveis. Então a probabilidade de rolar um três com um dado justo é um em seis, que nós escrevemos como 1/6. A probabilidade de escolher o ás de espadas no topo de um baralho bem embaralhado é 1/52.

É quase tão simples encontrar probabilidades para eventos que são compostos de vários resultados igualmente prováveis. Apenas contamos todos os resultados que o evento contém.

A probabilidade do evento é o número de resultados no evento dividido pelo número total de resultados possíveis. A probabilidade de rolar um número par com um dado justo é 3/6 = 0,5, uma vez que existem três números pares de um total de seis. Isso é o que chamamos de probabilidade clássica ou probabilidade baseada em modelo.

Mas qual é a probabilidade do ouro ser vendido por mais de US $ 2000 o grama no final do próximo ano? Você pode conseguir um número que pareça razoável. Como você chegou a essa probabilidade? Em nossa discussão sobre probabilidade, definimos a probabilidade de duas maneiras: (1) em termos da frequência relativa - ou a fração de vezes - que um evento ocorre a longo prazo; ou (2) como o número de resultados no evento dividido pelo número total de resultados. Nenhuma das situações se aplica à sua avaliação das chances de venda de ouro por mais de US $ 2.000 o grama.

Usamos a linguagem da probabilidade na fala cotidiana para expressar um grau de incerteza sem base em frequências relativas de longo prazo. Sua avaliação pessoal de um evento expressa sua incerteza sobre o resultado. Chamamos esse tipo de probabilidade de probabilidade subjetiva ou probabilidade pessoal.

Probabilidade Conjunta

Escolha de Prêmios				
Sexo	Taco de Golfe	Câmera	Bicicleta	**Total**
Masculino	117	50	60	**227**
Feminino	130	91	30	**251**
Total	**247**	**141**	**90**	**478**

Se o vencedor for escolhido aleatoriamente por esses clientes, a probabilidade de selecionarmos uma mulher é apenas a frequência relativa correspondente (já que temos a mesma probabilidade de selecionar qualquer um dos 478 clientes). Há 251 mulheres nos dados de um total de 478, dando uma probabilidade de:

$$P(\text{mulher}) = 251/478 = 0,525.$$

Isso é chamado de **probabilidade marginal**, porque depende apenas dos totais encontrados nas margens da tabela. O mesmo método funciona para eventos mais complicados.

Por exemplo, qual é a probabilidade de escolher uma mulher cujo prêmio preferido é a câmera? Como 91 mulheres nomearam a câmera como sua preferência, então a probabilidade é:

$$P(\text{mulher e câmera}) = 91/478 = 0,190.$$

Probabilidades como essas são chamadas de **probabilidades conjuntas** porque elas dão a probabilidade de dois eventos ocorrerem juntos.

Probabilidade Condicional e Independência

Escolha de Prêmios				
Sexo	Taco de Golfe	Câmera	Bicicleta	**Total**
Masculino	117	50	60	**227**
Feminino	130	91	30	**251**
Total	**247**	**141**	**90**	**478**

Escrevemos a probabilidade de um cliente selecionado querer uma bicicleta, uma vez que selecionamos uma mulher como:

$$P(\text{bicicleta} \mid \text{mulher}) = 30/251 = 0{,}120.$$

Para os homens, olhamos para a distribuição condicional de prêmios preferidos dado "Homem" mostrado na linha superior da tabela. Lá, dos 227 homens, 60 disseram que seu prêmio preferido era uma bicicleta. Então,

$$P(\text{bicicleta} \mid \text{homem}) = 60/227 = 0{,}264$$

mais que o dobro da probabilidade das mulheres.

Em geral, quando queremos a probabilidade de um evento de uma distribuição condicional, escrevemos P(B | A) e o pronunciamos “a probabilidade de B dado A.”

Uma probabilidade que leva em conta uma dada condição como essa é chamada de uma **probabilidade condicional**.

Vamos ver o que fizemos. Nós trabalhamos com as contagens, mas também poderíamos trabalhar com as probabilidades. Havia 30 mulheres que selecionaram uma bicicleta como prêmio, e havia 251 mulheres clientes. Então nós achamos a probabilidade de ser 30/251.

Para encontrar a probabilidade do evento B dado o evento A, restringimos nossa atenção aos resultados em A. Então, encontramos em que fração desses resultados B também ocorreu.

Formalmente, escrevemos:

$$P(\mathbf{B} \mid \mathbf{A}) = \frac{P(\mathbf{A} \text{ and } \mathbf{B})}{P(\mathbf{A})}$$

A fórmula para probabilidade condicional requer uma restrição. A fórmula funciona somente quando o evento que é dado tem uma probabilidade maior que 0.

A fórmula não funciona se P(A) for 0 porque isso significaria que nós tínhamos “dado” o fato de que A é verdade mesmo que a probabilidade de A fosse 0, o que seria uma contradição. Lembra-se da Regra de Multiplicação para a probabilidade de A e B?

P (A e B) = P (A) * P (B) quando A e B são independentes.

Agora podemos escrever uma regra mais geral que não requer independência. Na verdade, já escrevemos. Nós só precisamos reorganizar a equação um pouco.

A equação na definição da probabilidade condicional contém a probabilidade de A e B.

A reorganização da equação fornece a Regra Geral de Multiplicação para eventos compostos que não exigem que os eventos sejam independentes:

P(A e B) = P(A) * P(B | A)

A probabilidade de que dois eventos, A e B, ocorram é a probabilidade de que o evento A ocorra multiplicado pela probabilidade de que o evento B também ocorra – isto é, pela probabilidade de que o evento B ocorra dado que o evento A ocorra.

Ou (Or)	Em Geral	P(A or B) = P(A) + P(B) - P(A and B)
	Se eventos são mutuamente exclusivos	P(A or B) = P(A) + P(B)
E (And)	Em Geral	P(A and B) = P(A) * P(B | A) = P(A | B) * P(B)
	Se eventos são independentes	P(A and B) = P(A) * P(B)

Independente ou Mutuamente Exclusivo

Os eventos mutuamente exclusivos são independentes? Ambos os conceitos parecem ter ideias semelhantes de separação e distinção sobre eles, mas, na verdade, eventos mutuamente exclusivos não podem ser independentes. Vejamos porquê.

Considere os dois eventos mutuamente exclusivos {você recebe um A em Geografia} e {recebe um B em Geografia}. Eles são mutuamente exclusivos porque não têm resultados em comum. Suponha que você saiba que obteve um A em Geografia.

Agora, qual é a probabilidade de você ter um B? Você não pode obter as duas notas, então deve ser 0. Pense no que isso significa. Sabendo que o primeiro evento (obtendo um A) ocorreu mudou sua probabilidade para o segundo evento (zero). Então, esses eventos não são independentes.

Eventos mutuamente exclusivos não podem ser independentes.

Eles não têm resultados em comum, então saber que um ocorreu significa que o outro não aconteceu. Um erro comum é tratar eventos mutuamente exclusivos como se fossem independentes e aplicar a Regra de Multiplicação para eventos independentes. Não cometa esse erro.

Tabela de Contingência

As **Tabelas de Contingência** são os meios de organizar as informações correspondentes aos dados classificados segundo dois critérios.

As Tabelas de Contingência permitem representar os dados, quer sejam eles qualitativos ou quantitativos.

Nas Tabelas de Contingência podemos ter os dados das linhas representados por um critério e os dados das colunas representados por outro critério totalmente diferente.

Nós usamos Tabela de Contingência para comparar 2 variáveis. As Tabelas de Contingência são muito utilizadas com probabilidades.

	Variável 2				
Variável 1	A	B	...	r	total
A	$n_{1\,1}$	$n_{1\,2}$		$n_{1\,r}$	$n_{1\,\cdot}$
B	$n_{2\,1}$	$n_{2\,2}$		$n_{2\,r}$	$n_{2\,\cdot}$
....	⋮	⋮		⋮	⋮
k	$n_{k\,1}$	n_{k2}		$n_{k\,r}$	$n_{k\,\cdot}$
total	$n_{\cdot\,1}$	$n_{\cdot\,2}$		$n_{\cdot\,r}$	n

Distribuições de Probabilidade – Variável Aleatória

Em Estatística, uma Distribuição de Probabilidade descreve a chance que uma variável (discreta ou contínua) pode assumir ao longo de um espaço de valores.

Uma Distribuição de Probabilidade é um modelo matemático que relaciona um certo valor da variável de estudo com a sua probabilidade de ocorrência.

O conjunto de todos os possíveis resultados de um experimento aleatório é denominado espaço amostral.

O resultado de um experimento de probabilidade geralmente é uma contagem ou uma medida. Quando isso ocorre, o resultado é chamado de **variável aleatória**.

As variáveis aleatórias podem ser de dois tipos: discretas ou contínuas.

Distribuição Discreta: Binomial, Poisson, Bernoulli, Geométrica e Hipergeométrica.

Distribuição Contínua: Normal, Uniforme, Exponencial, Gama e t de Student

Distribuições de Probabilidade Discreta

Distribuição Binomial:

A **Distribuição Binomial** é utilizada para descrever cenários em que os resultados de uma variável aleatória podem ser agrupados em duas categorias.

No geral, as duas categorias de uma distribuição binomial são classificadas como:

Sucesso ou **Falha.**

Portanto, a probabilidade de sucesso podemos chamar de p. E a probabilidade de falha vamos chamar de q. Ou seja:

$$p = 1 - q$$

Onde: p = probabilidade de sucesso q = probabilidade de fracasso

Condições de aplicação de uma Distribuição Binomial:

- São realizadas n repetições no experimento, onde n é uma constante.
- Só existem dois resultados possíveis em cada repetição, Sucesso e Falha.

- A probabilidade de sucesso e a de falha permanecem constantes em todas as repetições.
- Todas as repetições são independentes. Os resultados não são influenciados por resultados externos.

Os parâmetros da Distribuição Binominal são n e p.

A **Média de uma Distribuição Binomial**, representa a média de longo prazo de sucessos esperados, com base no **número de observações**. Fórmula:

$$\text{Média} = n.P$$

Onde: n = número de tentativas e p = probabilidade de sucesso

A **Variância de uma Distribuição Binomial**, representa a variação que existe no número de sucessos (p) sobre um número (n) de observações. Fórmula:

$$\text{Variância} = (n.p).(1-p)$$

Onde: n = número de tentativas e p = probabilidade de sucesso

Distribuição Poisson:

A **Distribuição Poisson** é utilizada para descrever cenários onde existe a probabilidade de ocorrência do evento em um intervalo contínuo.

Condições de aplicação de uma Distribuição Poisson:

- O número de ocorrências depende do tamanho do intervalo.
- As ocorrências não interferem sobre as ocorrências de intervalos externos.
- A probabilidade de duas ou mais ocorrências acontecerem num mesmo intervalo de tempo é muito pequena.

A Distribuição Poisson é caracterizada pelo parâmetro único chamado **λ (lambda), que representa a taxa média de ocorrência por unidade de medida**.

$$P(x) = \frac{e^{-\lambda}.\lambda^{x}}{x!}$$

Distribuição Hipergeométrica:

Um dos pontos chave das Distribuições **Binomial** e **Poisson** é que os **eventos são independentes** uns dos outros. Cada amostra de cada experimento é um conjunto novo de dados. Desta forma, a **probabilidade de sucesso** ou de número de ocorrências, se mantém **constante**.

A **Distribuição Hipergeométrica** é uma distribuição de probabilidade discreta que descreve o número de sucessos numa sequência de n extrações de uma população finita, ou seja, sem reposição.

Quando a amostragem é sem substituição, a **probabilidade de sucesso muda durante o processo de amostragem**, isso viola os requisitos para uma distribuição de probabilidade binomial. Nesse caso, use a Distribuição Hipergeométrica.

Fórmula da Distribuição Hipergeométrica:

$$P(x) = \frac{{}_{N-R}C_{n-x} \cdot {}_{R}C_{x}}{{}_{N}C_{n}}$$

onde: N = Tamanho da população, R = O número de sucessos da população, n = Tamanho da Amostra e x = Número de sucessos da amostra

Assim como as outras distribuições, a Distribuição Hipergeométrica também possui **média e desvio padrão**.

$$\mu = \frac{nR}{N}$$

onde:

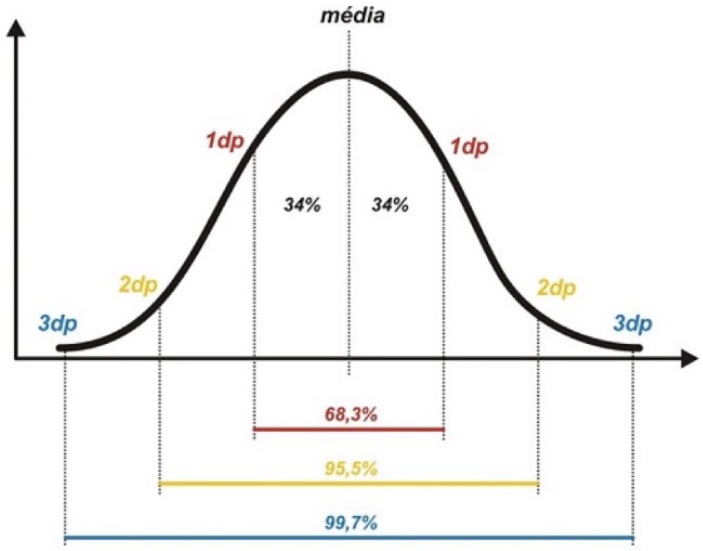
N = Tamanho da população
R = O número de sucessos da população
n = Tamanho da Amostra

$$\sigma = \sqrt{\frac{nR(N-R)}{N^2}} \sqrt{\frac{N-n}{N-1}}$$

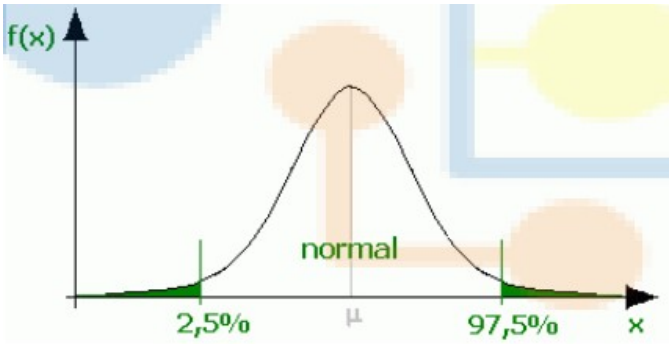
onde:

N = Tamanho da população
R = O número de sucessos da população
n = Tamanho da Amostra

Distribuições de Probabilidade Contínua



A **Distribuição Normal** é útil quando os dados tendem a estar próximos ao **centro da distribuição (próximos da média)** e quando **valores extremos (outliers)** são muito raros.



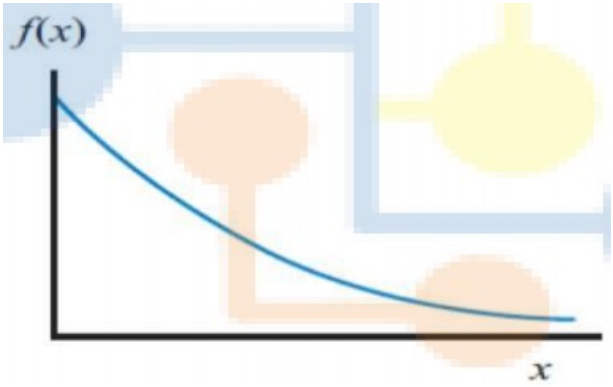
A **Distribuição Uniforme** é usada para descrever os dados quando todos os valores têm a mesma chance de ocorrer.

A Distribuição Uniforme é a distribuição de probabilidades contínua mais simples de conceituar: a probabilidade de se gerar qualquer ponto em um intervalo contido no espaço amostral é

proporcional ao tamanho do intervalo, visto que na distribuição uniforme a f(x) é igual para qualquer valor de x no intervalo considerado.

Outra maneira de se dizer "distribuição uniforme" seria "um número finito de resultados com chances iguais de acontecer". Ela é usada quando assumimos intervalos iguais da variável aleatória que tem a mesma probabilidade.

A **Distribuição Exponencial** é usada para descrever os dados quando valores mais baixos tendem a dominar a distribuição e quando valores muito altos não ocorrem com frequência.



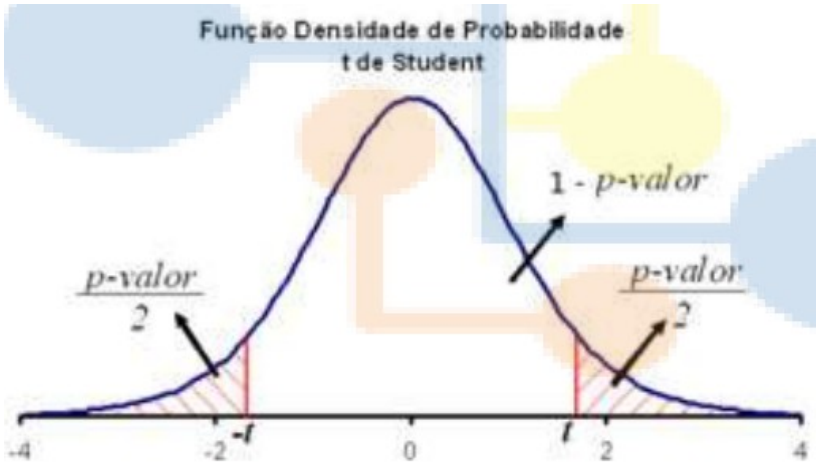
Na **Distribuição Poisson**, a variável aleatória é definida como o número de ocorrências em determinado período, sendo a média das ocorrências definida como λ (lambda).

Na **Distribuição Exponencial**, a variável aleatória é definida como o tempo entre duas ocorrências, sendo a média de tempo entre as ocorrências de 1 / λ.

A Distribuição Exponencial é amplamente usada no campo da confiabilidade, como um modelo para a distribuição dos tempos até a falha de componentes eletrônicos.

Nessas aplicações, o parâmetro λ representa a taxa de falha para o componente e 1 / λ é o tempo médio até a falha!

A **Distribuição t de Student** é uma das principais distribuições de probabilidade, com inúmeras aplicações em inferência estatística.



Resumindo:

Distribuições de Probabilidade Discreta e Contínua Na caracterização das distribuições de probabilidade é de grande importância a utilização de medidas que indiquem aspectos relevantes da distribuição, como medidas de posição (média, mediana e moda), medidas de dispersão (variância e desvio-padrão) e medidas de assimetria e curtose.

O entendimento dos conceitos relativos a probabilidade e distribuições de probabilidade auxiliarão o Cientista de Dados no estudo de tópicos sobre inferência estatística, incluindo testes de hipóteses paramétricos e não paramétricos, análise de regressão, etc.

A Distribuição Normal

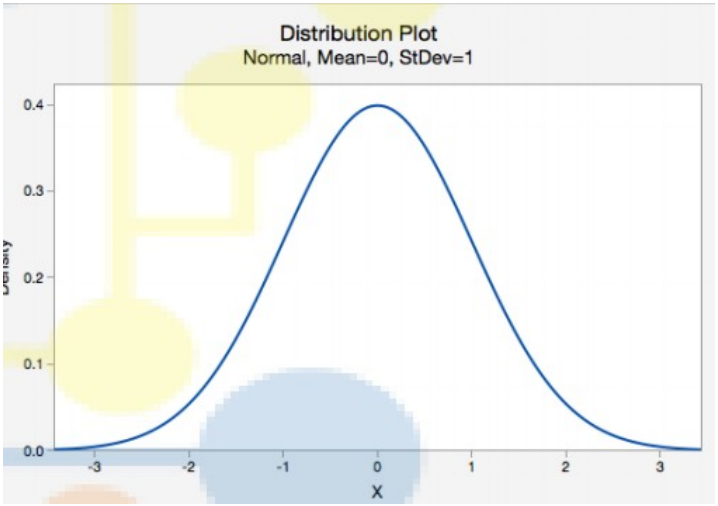
Uma distribuição estatística é uma função que define uma curva e a área sob essa curva determina a probabilidade de ocorrer o evento por ela correlacionado.

Você provavelmente já viu a Distribuição Normal antes, e se você já viu uma curva em forma de “bellshaped” (formato de sino), era provavelmente um modelo Normal.

Modelos normais são definidos por dois parâmetros: uma média e um desvio padrão. Por convenção, indicamos os parâmetros com letras gregas. Por exemplo, representamos a média de tal modelo com a letra grega m, que é o equivalente grego de "m", para média, e o desvio padrão com a letra grega s, o equivalente grego de "s", para padrão desvio.

A Distribuição Normal é a mais importante dentre as distribuições estatísticas. Também conhecida como Distribuição Gaussiana, é uma curva simétrica em torno do seu ponto médio, apresentando assim seu famoso formato de sino.

A curva normal representa o comportamento de diversos processos nas empresas e muitos fenômenos comuns, como por exemplo, altura ou peso de uma população, a pressão sanguínea de um grupo de pessoas, o tempo que um grupo de estudantes gasta para realizar uma prova.



Há um **Modelo Normal** diferente para cada combinação de m e s, mas se padronizarmos nossos dados primeiro, criando z-scores e subtraindo da média para fazer a média igual a 0 e dividindo pelo desvio padrão para fazer o desvio padrão igual a 1, então precisaremos apenas do modelo com

média 0 e desvio padrão 1. Chamamos isso de Modelo Normal Padrão ou Distribuição Normal Padrão (Standard Normal Distribution.).

Obviamente, não devemos usar um modelo Normal para todos os conjuntos de dados. Se o histograma não for em forma de sino, as pontuações (scores) z não serão bem modeladas pelo modelo Normal. E a padronização não ajuda, porque a padronização não altera a forma da distribuição. Portanto, sempre verifique o histograma dos dados antes de usar o modelo Normal.

Como Determinar Se a Distribuição é Normal?

Para determinar se uma determinada variável aleatória segue uma distribuição normal, basta verificar se essa segue a função densidade de probabilidade, dada por:

$$f(x) = \frac{e^{\frac{-(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma}}$$

Onde μ é a média e σ² é a variância de x.

A notação N(μ,σ²) é usada para representar tal distribuição. Para calcularmos então a probabilidade de um resultado, basta integrar a função f(x) em relação a x, com os limites de integração representando a faixa de valores que se quer obter a probabilidade. Vale notar que a integral da função densidade de probabilidade normal não possui solução analítica. Sendo assim, seu cálculo deve ser realizado através de um método numérico.

Para sanar tal dificuldade a função pode ser padronizada com a substituição dos parâmetros por μ = 0 e σ² = 1. Essa abordagem é dada pela definição de uma nova variável aleatória Z, chamada de variável aleatória normal padronizada. Se x for uma variável aleatória normal com média E(x) = μ e variância V(x) = σ², a variável aleatória Z = (x-μ)/σ será uma variável aleatória normal, com E(Z) = 0 e V(Z) = 1.

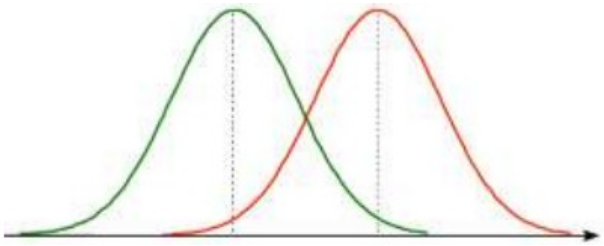
Ou seja, Z é uma variável aleatória normal padrão.

Dessa forma, é possível obter a área sob a curva da normal padrão de forma analítica, e então obter a área entre dois pontos sob a curva, diretamente com o uso de uma tabela de conversão, e essa área representa uma probabilidade.

Teorema do Limite Central

O Teorema do Limite Central é fundamental para a Estatística, uma vez que diversos procedimentos estatísticos comuns requerem que os dados sejam aproximadamente **normais** e o Teorema do Limite Central permite aplicar esses procedimentos úteis a populações que são fortemente **não-normais**.

Esse teorema possibilita medir o quanto sua média amostral irá variar, sem ter que pegar outra média amostral para fazer a comparação. Ou seja, permite-nos conduzir alguns procedimentos de inferência sem ter qualquer conhecimento de distribuição da população.



Esse teorema basicamente diz que sua **média amostral** tem uma **Distribuição Normal**, independente da aparência da distribuição dos dados originais.

Muitos procedimentos pressupõem que uma **Distribuição Normal** é uma **Distribuição Simétrica**.

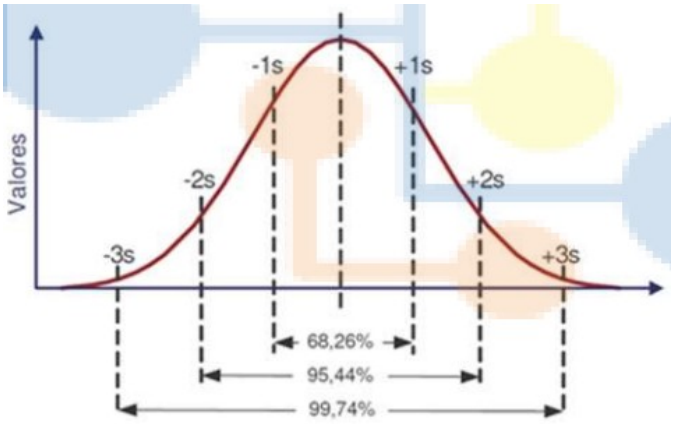
Assimetria indica variação no formato de distribuição.

Assimetria Positiva - Implica em uma concentração maior de valores menores, e o gráfico possuirá uma cauda mais longa à direita.

Assimetria Negativa - implica em uma concentração de valores maiores, e o gráfico possuirá uma cauda maior à esquerda.

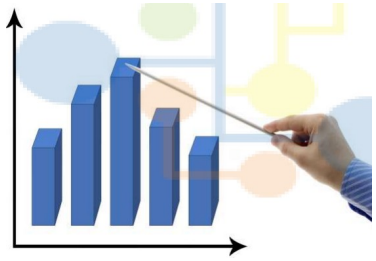
Os valores de grandes conjuntos de dados, normalmente se localizam ao redor da média ou da mediana. Desta forma, um histograma dos dados, mostraria uma curva simétrica bem definida (em forma de sino) em uma Distribuição Normal.

Em uma Distribuição Normal de dados, simétrica, nós podemos esperar que 68%, 95% e 99.7% dos valores estarão em, respectivamente, 1, 2 e 3 desvios-padrão acima e abaixo da média.



Ou seja, em uma curva simétrica dos dados, praticamente todos os dados estarão em até 3 desvios-padrão do centro dos dados (média).

Perceba que este conceito somente se aplica, quando os dados criam um histograma simétrico.



A área abaixo da curva Normal representa 100% de probabilidade associada a uma variável.

A probabilidade de uma variável aleatória tomar um valor entre dois pontos quaisquer é igual à área compreendida entre esses dois pontos.

Árvores de Probabilidade

Algumas decisões de negócios envolvem uma avaliação mais sutil das probabilidades. Dadas as probabilidades de várias circunstâncias que podem afetar os negócios, podemos usar uma imagem chamada “árvore de probabilidade” ou “diagrama de árvore” para ajudar a pensar no processo de tomada de decisão. Uma árvore mostra sequências de eventos como caminhos que parecem ramos de uma árvore. Isso pode nos permitir comparar vários cenários possíveis. Esse é o conceito por trás de uma família de algoritmos de Machine Learning, as Decision Trees e as Random Forests. Aqui está um exemplo.

Dispositivos eletrônicos pessoais, como smartphones e tablets, estão se tornando mais capazes o tempo todo. Fabricar componentes para esses dispositivos é um desafio e, ao mesmo tempo, os consumidores estão exigindo cada vez mais funcionalidades. As leis microscópicas e até submicroscópicas podem se desenvolver durante sua fabricação, o que pode eliminar pixels nas telas ou causar falhas de desempenho intermitentes. Os defeitos sempre ocorrerão, portanto, o engenheiro de qualidade responsável pelo processo de produção deve monitorar o número de defeitos e agir se o processo parecer fora de controle.

Suponhamos que o engenheiro seja chamado para a linha de produção porque o número de defeitos ultrapassou um limite. Ele deve decidir entre duas ações possíveis. Ele sabe que um pequeno ajuste nos robôs que montam os componentes pode resolver vários problemas, mas para problemas mais complexos, toda a linha de produção precisa ser desligada para identificar a fonte. O ajuste requer que a produção seja interrompida por cerca de uma hora, mas desligar a linha de produção interrompe o trabalho de pelo menos um turno inteiro (oito horas). Naturalmente, seu chefe preferiria que ele fizesse o ajuste simples. Mas sem saber a origem ou a gravidade do problema, ele não pode ter certeza se isso será bemsucedido.

Se o engenheiro quiser prever se o ajuste menor funcionará, ele poderá usar uma árvore de probabilidade para ajudar a tomar a decisão. Com base em sua experiência, o engenheiro acredita que há três problemas possíveis:

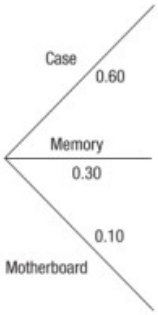
(1) As placas-mãe podem ter conexões defeituosas,

(2) A memória pode ser a fonte das conexões defeituosas ou

(3) Alguns cases das placas-mãe podem estar incorretamente acoplados na linha de montagem.

Ele sabe de dados empíricos anteriores com que frequência esses tipos de problemas surgem e como é provável que apenas fazer um ajuste conserte cada tipo de problema. Os problemas da placa-mãe são raros (10%), os problemas de memória têm aparecido em cerca de 30% do tempo e os problemas de alinhamento dos cases ocorrem com mais frequência (60%).

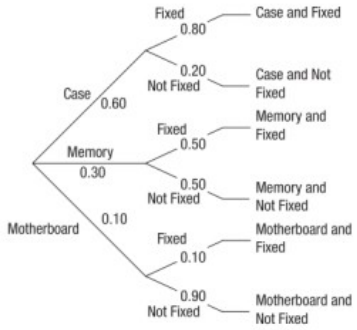
Podemos colocar essas probabilidades no primeiro conjunto de ramificações, como mostra a imagem abaixo.



Observe que cobrimos todas as possibilidades e, portanto, as probabilidades somam um. Para este diagrama, podemos agora adicionar as probabilidades condicionais de que um ajuste menor conserte cada tipo de problema. Dados anteriores indicam que:

- P (correção | placa-mãe) = 0.10,
- P (correção | memória) = 0.50 e
- P (correção | alinhamento do case) = 0.80.

No final de cada ramo que representa o tipo de problema, desenhamos duas possibilidades (Fixed ou Not Fixed – Corrigido ou Não Corrigido) e escrevemos as probabilidades condicionais nas ramificações, como mostra a imagem abaixo.

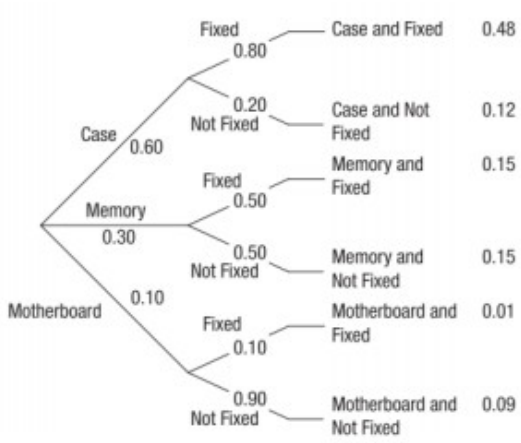


No final de cada segundo ramo, escrevemos o evento conjunto correspondente à combinação dos dois ramos. Por exemplo, o ramo superior é a combinação do problema sendo o alinhamento do case, e o resultado do pequeno ajuste é que o problema agora está corrigido.

Para cada um dos eventos conjuntos, podemos usar a Regra Geral de Multiplicação para calcular sua probabilidade conjunta. Por exemplo:

P (case e corrigido) = P (case) * P (corrigido | case) = 0.60 * 0.80 = 0.48

Escrevemos essa probabilidade ao lado do evento correspondente. Fazendo isso para todas as combinações de ramificações, é apresentada a imagem abaixo.



Todos os resultados na extrema direita são desarticulados, ou seja, eles não se sobrepõem, pois em cada nó, todas as opções são alternativas desarticuladas. E essas alternativas são todas as possibilidades, então as probabilidades na extrema direita devem se somar a um. Como os resultados finais são desarticulados, podemos adicionar qualquer combinação de probabilidades para encontrar probabilidades de eventos compostos.

Em particular, o engenheiro pode responder sua pergunta: qual é a probabilidade de o problema ser corrigido por um simples ajuste? Ele encontra todos os resultados na extrema direita em que o problema foi corrigido. Há três (um correspondente a cada tipo de problema) e ele soma suas probabilidades: 0,48 + 0,15 + 0,01 = 0,64. Portanto, 64% de todos os problemas são corrigidos pelo simples ajuste. Os outros 36% exigem uma investigação importante. Baseado em dados, agora o engenheiro pode tomar uma decisão.

Nesta aula, desenhamos nossas árvores de probabilidade da esquerda para a direita. Também podemos desenhar na vertical, de cima para baixo.

O Que é Estatística Inferencial?

Até aqui estudamos Estatística Descritiva, para descrever como os dados estão organizados e Probabilidade para medir a variabilidade de fenômenos casuais de acordo com a sua ocorrência.

A estatística inferencial tem como objetivo a extrapolação dos resultados (obtidos com a estatística descritiva) para a população.

População e Amostra

Imagine que você seja convidado a realizar uma pesquisa para medir a durabilidade das lâmpadas produzidas por uma determinada fábrica. Qual abordagem você usaria?

1- Testar TODAS as lâmpadas produzidas.

2- Obter uma amostra representativa da população de lâmpadas produzidas e então inferir a durabilidade de todas as lâmpadas.

População	Amostra
População é o conjunto de todos os elementos ou resultados sob investigação.	Amostra é qualquer subconjunto da população.

Princípios de Amostragem

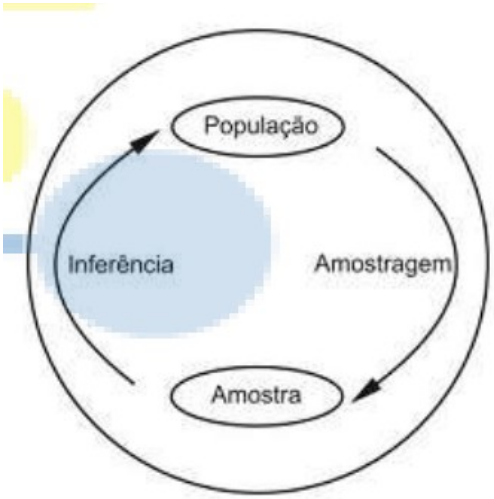
“Para saber se o bolo de chocolate está bom, basta comer uma fatia.”

Amostragem é o processo de determinação de uma amostra a ser pesquisada. A amostra é uma parte de elementos selecionada de uma população.

Enquanto que um censo envolve um exame a todos os elementos de um dado grupo, a amostragem envolve um estudo de apenas uma parte dos elementos.

A amostragem consiste em selecionar parte de uma população e observá-la com vista a estimar uma ou mais características para a totalidade da população.

A teoria da amostragem estuda as relações existentes entre uma população e as amostras extraídas dessa população. E útil para avaliação de grandezas desconhecidas da população, ou para determinar se as diferenças observadas entre duas amostras são devidas ao acaso ou se são verdadeiramente significativas.



Exemplos:

- Sondagens à opinião pública que servem para conhecer a opinião da população sobre variadas questões. As mais populares são as sondagens políticas.
- Inspeção de mercado utilizada com o intuito de descobrir as preferências das pessoas em relação a certos produtos. Um dos exemplos mais conhecidos da aplicação desta amostragem é a lista de audiências dos programas de televisão.
- Para estimar a prevalência de uma doença rara, a amostra pode ser constituída por algumas instituições médicas, cada uma das quais com registros dos pacientes.

Termos Básicos da Amostragem:

- População
- Unidade
- Amostra
- Variável

Tipos de Amostragem

Métodos Aleatórios	Métodos Não Aleatórios

• Amostragem Aleatória Simples
  ◦ Simples
  ◦ Sem Reposição
  ◦ Com Reposição
• Amostragem Sistemática
• Amostragem Estratificada
• Amostragem por Aglomerados
• Amostragem Multi-etapas
• Amostragem Multifásica
	• Amostra Intencional
	• Amostra “Snowball”
	• Amostra por quotas
	• Amostra por conveniência

Testes de Hipótese

Uma hipótese estatística é uma suposição sobre um determinado parâmetro da população, como média, desvio-padrão, coeficiente de correlação etc. Um teste de hipótese é um procedimento para decisão sobre a veracidade ou falsidade de uma determinada hipótese.

Um Teste de Hipótese Estatística é um procedimento de decisão que nos possibilita decidir entre Ho (hipótese nula) ou Ha (hipótese alternativa), com base nas informações contidas na amostra.

Ho: A hipótese nula afirma que um parâmetro da população (como a média, o desvio padrão, e assim por diante) é igual a um valor hipotético. A hipótese nula é, muitas vezes, uma alegação inicial baseado em análises anteriores ou conhecimentos especializados.

Ha: A hipótese alternativa afirma que um parâmetro da população é menor, maior ou diferente do valor hipotético na hipótese nula. A hipótese alternativa é aquela que você acredita que pode ser verdadeira ou espera provar ser verdadeira.

Como estamos analisando dados da amostra e não da população, erros podem ocorrer:

Erro Tipo I é a probabilidade de rejeitarmos a hipótese nula quando ela é efetivamente verdadeira.

Erro Tipo II é a probabilidade de rejeitarmos a hipótese alternativa quando ela é efetivamente verdadeira.

Exemplo:

Um pesquisador tem resultados de exames para uma amostra de alunos que fizeram um curso de formação para um exame nacional. O pesquisador quer saber se os alunos formados obtiveram pontuação acima da média nacional de 78.

Uma hipótese alternativa pode ser usada porque o pesquisador está especificamente levantando a hipótese de que as pontuações para alunos formados são maiores do que a média nacional.

Uma hipótese alternativa pode ser usada porque o pesquisador está especificamente levantando a hipótese de que as pontuações para alunos formados são maiores do que a média nacional. ($H0 : \mu = 78$ e $Ha : \mu > 78$)

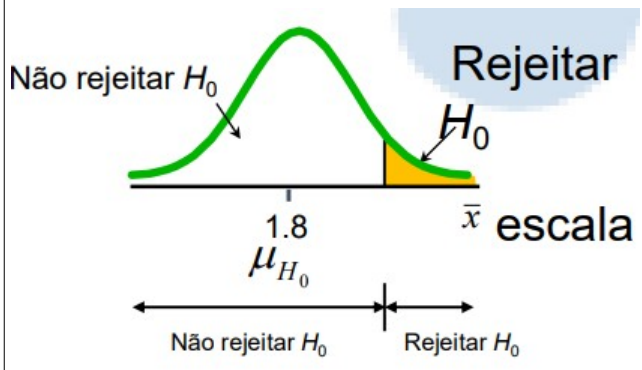
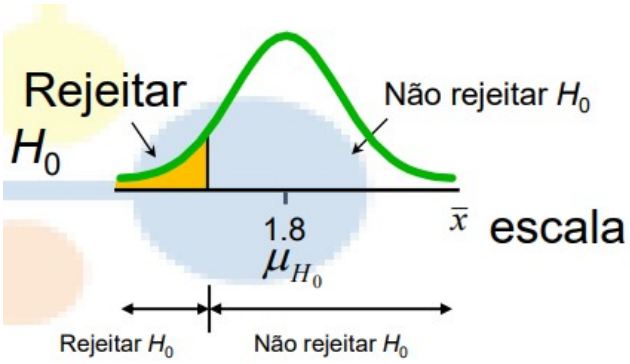
→ →

Formular as hipóteses nula e alternativa.	Coletar uma amostra de tamanho n e calcular a média da amostra.	Traçar a média da amostra no eixo x da distribuição da amostra.	Escolher um nível de significância ✔ com base na gravidade do erro tipo I.	Calcular a estatística, os valores críticos e a região crítica.	Se a média da amostra estiver na região branca do gráfico NÃO rejeitamos a hipótese nula.	Se a média da amostra estiver em uma das caudas nós rejeitamos a hipótese nula.

Testes de Hipótese Unilateral

O teste Unilateral ou Unicaudal é usado quando a hipótese alternativa é expressa como:

< ou >

H0 : μ = 1.8 HA : μ > 1.8 **Teste Cauda Superior:** nós assumimos que μ = 1.8 a menos que a média da amostra seja maior que the 1.8	H0 : μ = 1.8 HA : μ < 1.8 **Teste Cauda Inferior:** nós assumimos que μ = 1.8 a menos que a média da amostra seja menor que 1.8
	
Teste Unilateral Direito Se a média estiver dentro da região branca do gráfico, não rejeitamos a hipótese nula, caso contrário, a rejeitamos. Teste Unilateral Direito: (Superior) H0 : μ = valor numérico HA: μ > valor numérico	Teste Unilateral Esquerdo Se a média estiver dentro da região branca do gráfico, não rejeitamos a hipótese nula, caso contrário, a rejeitamos. Teste Unilateral Esquerdo: (Inferior) H0 : μ = valor numérico HA: μ < valor numérico

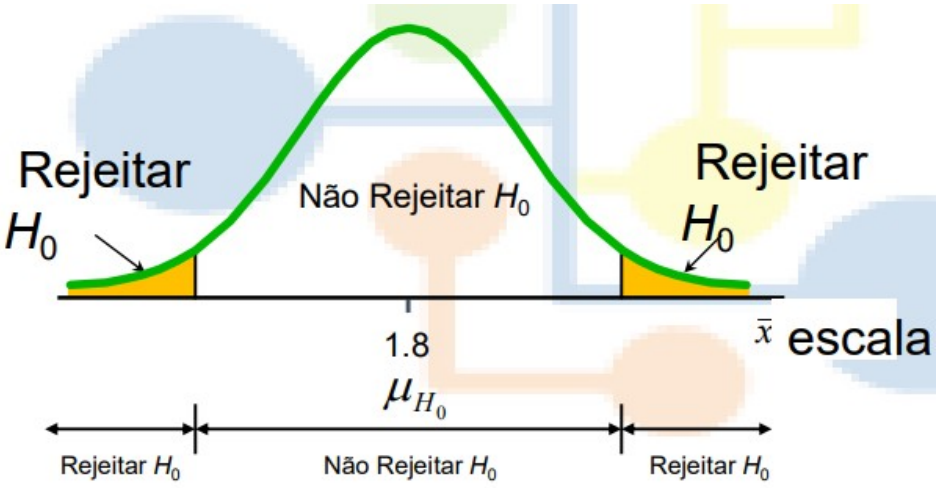
Testes de Hipótese Bilateral

O teste Bilateral é usado sempre que a hipótese alternativa é expressa como ≠ de. (diferente de)

H0 : μ = 1.8

HA : μ ≠ 1.8

Nós assumimos que μ = 1.8 a menos que a média da amostra seja ≠ que 1.8



A curva acima representa a distribuição da amostragem da média de utilização de banda larga. Assume-se que a média da população é 1.8 GB, de acordo com a hipótese nula H0 : ◡ = 1.8.

Por existirem duas regiões de rejeição no gráfico (regiões em amarelo), este é chamado teste de hipótese bilateral ou bicaudal.

Como a hipótese nula é expressa como ≠ ela pode ser maior ou menor que, por isso o teste é bilateral.

H0 : μ = Valor numérico.

HA : μ ≠ Valor numérico.

Erros Tipo I e Tipo II

O propósito do teste de hipótese é verificar a validade de uma afirmação sobre um parâmetro da população, baseado em amostragem.

Como estamos tomando amostra como base, estamos expostos ao **risco de conclusões erradas** sobre a população, por conta de **erros de amostragem**.

A hipótese nula pode ser verdadeira, caso tenhamos coletado uma amostra que não seja representativa da população. Ou talvez, a amostra tenha sido muito pequena.

Para testar a H0, é preciso definir uma regra de decisão com o objetivo de estabelecer uma zona de rejeição da hipótese, ou seja, definir um nível de significância, (alfa), sendo os mais consensuais os alfas 0.10, 0.05 e 0.01.

Grau de Confiança → Nível de Significância:

90% → 0,10

95% → 0,05

99% → 0,01

Se o valor do parâmetro da população, defendido pela H0 , cair na zona de rejeição, então esse valor é muito pouco provável de ser o valor verdadeiro da população e a H0 será rejeitada em favor da HA.

Pode acontecer, que apesar de rejeitada com base em dados de uma amostra, a H0 de fato seja verdadeira. Nesse caso, estaríamos cometendo um erro de decisão.

Esse erro é chamado de Erro Tipo I, cuja probabilidade de ocorrência depende do alfa escolhido.

Quando o valor defendido pela H0 cair fora da zona de rejeição, então consideramos que não há evidência para rejeitar H0 em prejuízo da HA. Mas aqui, também podemos estar cometendo um erro se a HA, apesar de descartada pelos dados em mãos, for de fato verdadeira. Esse erro é chamado **Erro Tipo II**.

Condição		A Hipótese Nula é Verdadeira	A Hipótese Nula é Falsa
Decisão	Decidimos rejeitar a hipótese nula.	Erro Tipo I (Rejeição de uma hipótese nula verdadeira)	Decisão correta
	Não rejeitamos a hipótese nula.	Decisão correta	Erro Tipo II (Não rejeição de uma hipótese nula falsa)

Exemplo:

A eficácia de certa vacina após um ano é de 25% (isto é, o efeito imunológico se prolonga por mais de 1 ano em apenas 25% das pessoas que a tomam). Desenvolve-se uma nova vacina, mais cara e deseja-se saber se esta é, de fato, melhor.

Que hipóteses devem ser formuladas? Que erros podemos encontrar?

Hipótese Nula H0 : p = 0,25

Hipótese Alternativa HA : p > 0,25

Erro Tipo I : aprovar a vacina quando, na realidade, ela não tem nenhum efeito superior ao da vacina em uso.

Erro Tipo II : rejeitar a nova vacina quando ela é, de fato, melhor que a vacina em uso.

A probabilidade de se cometer um Erro Tipo I depende dos valores dos parâmetros da população e é designada por α (alfa – nível de significância).

Dizemos então que o nível de significância alfa de um teste, é a probabilidade máxima com que desejamos correr o risco de um Erro Tipo I.

O valor alfa é tipicamente predeterminado e escolhas comuns são (alfa) = 0.05 e (alfa) = 0.01

A probabilidade de se cometer um Erro Tipo II é designada por β (beta).

Intervalo de Confiança e Significância Estatística

Intervalo de Confiança é uma amplitude (ou um intervalo) de valores que tem a probabilidade de conter o valor verdadeiro da população. Observe que na definição de intervalo de confiança, está associado uma probabilidade. A esta probabilidade chamamos de: Nível de Confiança, Grau de Confiança ou Coeficiente de Confiança.

Essas probabilidades podem vir a partir de escolhas comuns do grau de confiança que se deseja alcançar, dentre os mais comuns temos:

Grau de Confiança	Nível de Significância α	Valor Crítico z
90%	0,10	1,645
95%	0,05	1,96
99%	0,01	2,575

Considerando:

$$\text{Probabilidade } \{c1 <= \mu <= c2\} = 1 - a$$

Onde:

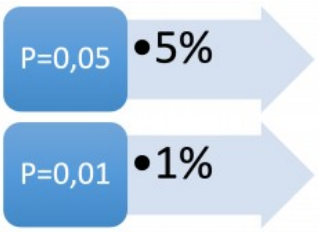
- O intervalo (c1, c2) é chamado de intervalo de confiança.
- μ = média da população.
- a é o nível de significância.
- 100 (1 - a) é nível de confiança.
- 1 - a é o coeficiente de confiança.

Um Intervalo de Confiança funciona como um indicador da precisão da sua medida. E indica qual o grau de estabilidade da sua estimativa, a qual pode ser calculada para determinar o quanto você está próximo de sua estimativa original quando realiza um ou mais experimentos. Portanto, o Intervalo de Confiança está associado a um grau de confiança que é uma medida da nossa certeza de que o intervalo contém o parâmetro populacional.

O principal objetivo da análise estatística de dados é estabelecer se os resultados possuem ou não significância estatística, de acordo com os parâmetros estabelecidos.

Ao formular uma hipótese em relação a uma determinada característica de uma população a amostra pode pertencer ou não à população de origem. Portanto as diferenças observadas são decorrentes de variações normais ou a amostra pode não pertencer a população e as diferenças encontradas representam um efeito real, não podendo ser atribuídas ao acaso. Vale ressaltar que devemos ter os níveis de significância previamente escolhidos.

O nível de significância é o limite que se estabelece para afirmar que um certo desvio é decorrente de acaso ou não.



Os níveis de significância mais aceitos são P = 0,05 e P = 0,01, ou seja, 5% e 1% respectivamente.

A partir de um nível de significância convencionado (alfa) os desvios são ocasionados pela lei do acaso, e o resultado é considerado não significativo.

Na prática é considerado satisfatório o limite de 5% de probabilidade de erro. Não sendo significativas as diferenças que tiverem uma probabilidade acima desse limite. O nível de significância deve ser estabelecido antes do experimento ser realizado e corresponde ao risco que se corre de rejeitar uma hipótese verdadeira ou aceitar uma hipótese falsa. A Significância de um resultado também é denominada de p-value ou valor-p.

Uma das mais importantes funções da Estatística no mundo atual é coletar informações sobre uma amostra e então usar esta informação para analisar a população da qual a amostra foi extraída. O que fará um Cientista de Dados com essas informações? O Cientista de Dados irá utilizar recursos de intervalo de confiança diariamente para:

- Calcular a variabilidade de tempo de atendimento a clientes (bancos, centrais de atendimento, estabelecimentos em geral).
- Calcular o tempo de aterrissagem de voo.
- Calcular o tempo de substituição de aparelhos eletrônicos em uma Central de Processamento de Dados.
- Calcular a vida útil de componentes elétricos e eletrônicos em ambientes de trabalho e domésticos.
- Calcular os resultados de pesquisas eleitorais ou outras pesquisas em geral

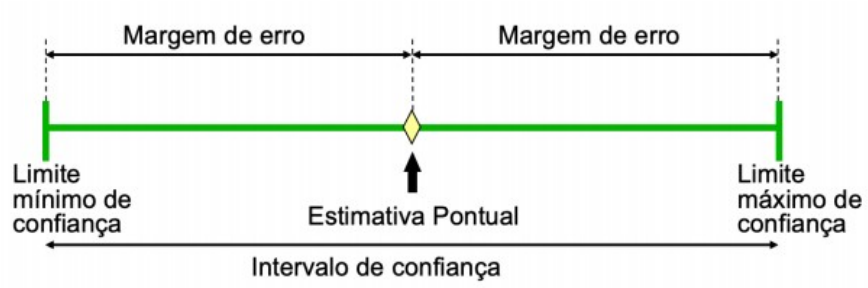
A Estatística oferece ferramentas para analisar dados e resolver problemas. E o que as empresas mais procuram nos dias de hoje é por profissionais que saibam resolver problemas. Seja um desses profissionais.

Intervalos de Confiança Para Desvio Padrão da População Conhecido

Vamos assumir que o desvio padrão da população s (sigma) é conhecido. O propósito de gerar um Intervalo de Confiança é prover uma estimativa para o valor da média da população e o Intervalo de Confiança para a média tem um limite de confiança máximo e um limite de confiança mínimo.

- LMax = limite máximo de confiança
- LMin = limite mínimo de confiança

Os limites descrevem o range em que nós temos um grau de confiança onde a média da população será encontrada. A Margem de erro, são os valores adicionados ou subtraídos da estimativa pontual, para formar o intervalo de confiança.



A margem de erro representa a largura do intervalo de confiança entre a média da amostra e seu limite máximo e entre a média e seu limite mínimo de confiança. Calculamos a Margem de Erro (ME) da seguinte forma:

$$ME_{\bar{x}} = z_{\alpha/2}\sigma_{\bar{x}}$$

Onde:

LMax = Média + Margem de erro

LMin = Média – Margem de erro

Podemos reduzir a margem de erro e ainda manter um nível de confiança de 90%, simplesmente aumentando o tamanho da amostra.

Quiz:

- O que faz a estatística descritiva?

  Utiliza métodos para coleta, organização, apresentação, análise e síntese de dados obtidos em uma população ou amostra.

- Um dos meios mais simples de descrever dados é através de tabelas de frequência, que refletem as observações feitas nos dados. **VERDADEIRO**

- Probabilidade é um valor numérico que indica a chance, ou probabilidade, de um evento específico ocorrer. Este valor numérico vai estar entre 0 e 1.

- Amostragem sistemática consiste em selecionar as unidades elementares da população em intervalos pré-fixados?

- Quando uma pesquisa é conduzida sem controle sobre quem são seus participantes (amostra) , a pesquisa é classificada como amostragem probabilística. **FALSO**

- A Estimação é o processo de se utilizar dados amostrais para estimar os valores para a média, o desvio padrão de uma população e a proporção populacional dos parâmetros.

- O Principal objetivo de uma Analista de Dados é extrair informações dos dados e transformá-las em conhecimento que possa ser usado para a tomada de decisões.

- A distribuição de probabilidade Discreta descreve quantidades aleatórias de dados que podem assumir valores finitos.

- Se você decidir contar o número de pessoas que visitam uma loja em um dia da semana, você vai estar trabalhando com Dados Discretos.

5.6. Machine Learning em Python

Introdução

Considerar uma carreira em Machine Learning e aprender tudo que for possível sobre esse assunto é uma das decisões mais inteligentes que você pode tomar na sua carreira profissional.

O que vamos estudar neste capítulo?

- Processo de Machine Learning
- Biblioteca Scikit-learn
- Coleta, Análise Exploratória e Pré-Processamento
- Feature Selection
- Algoritmos de Machine Learning – Classificação
- Algoritmos de Machine Learning – Regressão
- Métodos Ensemble
- Algoritmo XGBoost

Este capítulo não é um curso de Machine Learning, mas sim uma introdução ao tema. O curso de Machine Learning é o número 4 da Formação Cientista de Dados e aborda o tema em profundidade. Nosso objetivo aqui é que você aprenda como aplicar Machine Learning, pois mais a frente no curso aplicaremos aprendizado de máquina em dados gerados em tempo real.

Algoritmos de Machine Learning

Hoje, como Cientista de Dados, é possível construir sistemas que trituram dados com algoritmos complexos, tudo com baixo custo e alta capacidade de processamento.

Machine Learning, ou seja, a aplicação de algoritmos e ciência para extrair informações de dados, é um dos campos mais espetaculares da ciência da computação atualmente.

Tipos Algoritmos de Machine Learning:

- Aprendizagem Supervisionada
- Aprendizagem Não Supervisionada
- Aprendizagem Por Reforço

Aprendizagem Supervisionada

Como uma criança aprende?

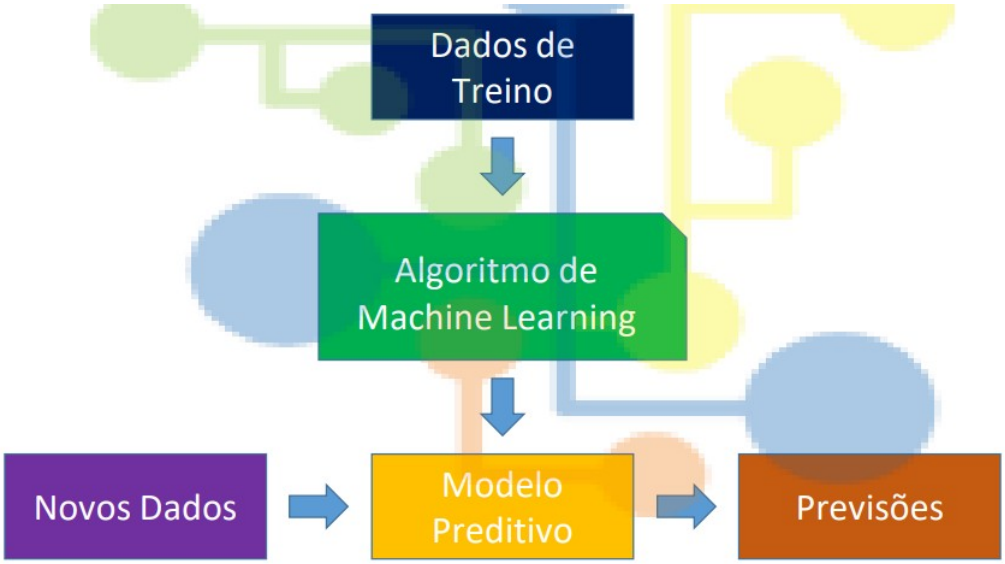
Um professor apresenta imagens, textos ou objetos informando para a criança o que aquilo representa.

Por exemplo, o professor apresenta a foto de um carro, explicando suas principais características. Mais tarde, quando a criança encontrar algo com as mesmas características será capaz de reconhecer que se trata de um carro.

Na aprendizagem supervisionada, temos as entradas (as características) e temos as saídas. O algoritmo então aprende o relacionamento nos dados e um modelo é criado. Quando o modelo é apresentado a novos dados de entrada, é capaz de prever as saídas.

Dados de Treino → Algoritmo de Machine Learning → Modelo Preditivo

Novos dados → Modelo Preditivo → Previsões



Aprendizagem Não Supervisionada

Na aprendizagem não supervisionada, temos as entradas (as características) mas não temos as saídas. O algoritmo aprende o relacionamento nos dados e gera agrupamentos (clusters).

Atributo 1	Atributo 2	Atributo 3	Saída
x1	x2	x3	Carro
x4	x5	x6	Avião
x7	x8	x9	?
x10	x11	x12	?

Aprendizagem Supervisionada

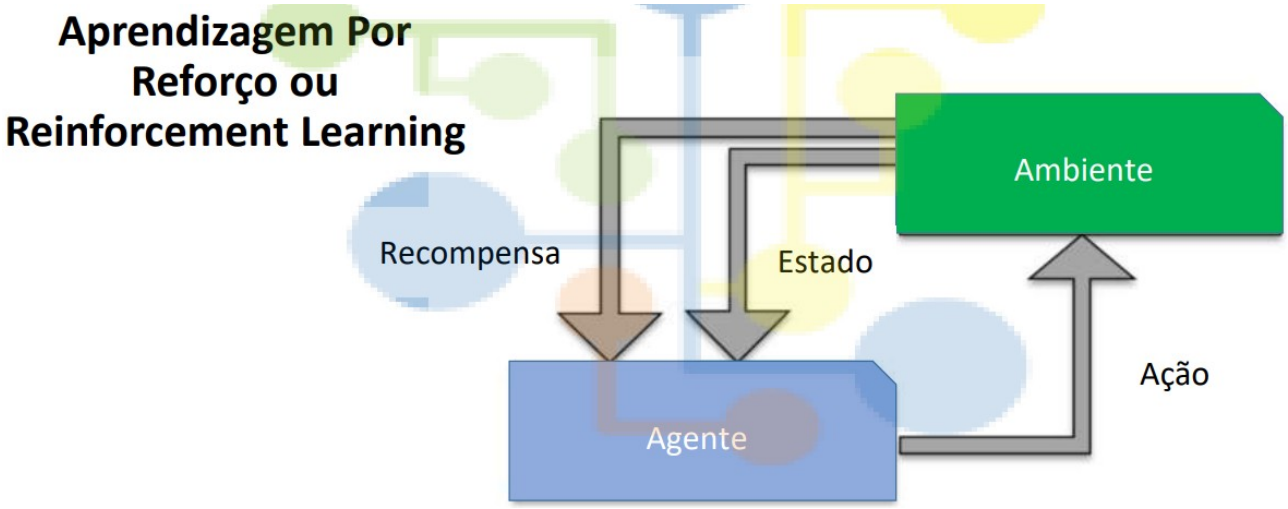
Aprendizagem Não Supervisionada

Aprendizagem Por Reforço

Como alguém aprende a andar de bicicleta? Podemos usar a aprendizagem supervisionada? E a aprendizagem não supervisionada? Qual seria o melhor método neste caso?

Tentativa e Erro.

Aprendizagem Por Reforço ou Reinforcement Learning:



Principais Algoritmos de Machine Learning

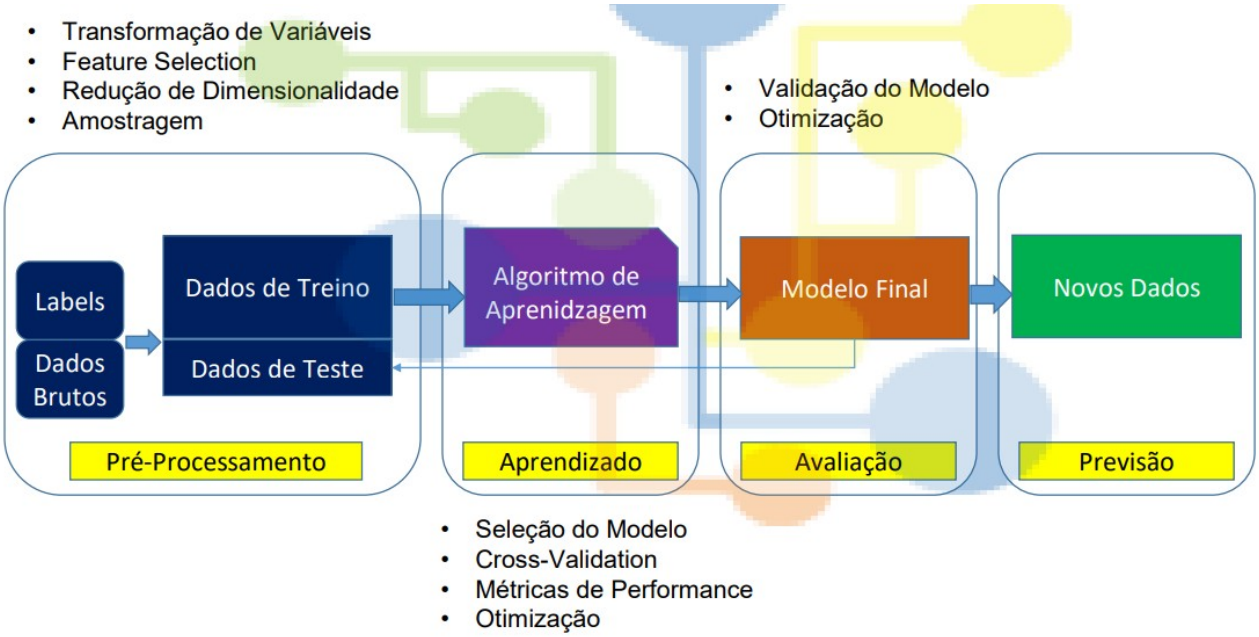
Quando trabalhamos com aprendizagem supervisionada temos basicamente 2 tipos de algoritmos:

- Regressão – previsão de valores numéricos.
- Classificação – previsão de classes ou categorias.

Principais tipos de algoritmos de Machine Learning:

- Aprendizagem Supervisionada:
  - Regressão Linear: utilizada para estimar valores reais (custo de uma casa, vendas de casas). Baseada em variáveis contínuas.
  - Regressão Logística: embora tenha regressão no nome, trata-se de um algoritmo de classificação.
  - Árvore de Decisão: melhor caminho entre entrada e saída.
  - Random Forest (Floresta Aleatória): vários algoritmos de árvore de decisão juntos. Pode ser muito útil para seleção de variáveis.
  - Support Vector Machines (SVM): pode ser usado para problemas de regressão e de classificação. Usado quando os dados não são linearmente separados, criando um outra dimensão para a realização da tarefa.
  - Naive Bayes: Algoritmo probabilístico.
  - K-Nearest Neighbors (KNN): pode ser usado para problemas de regressão e de classificação. Toma como base a distância matemática entre os pontos de dados.
- Aprendizagem Não Supervisionada:
  - K-Means: agrupa os dados de acordo com relacionamentos que mais se aproximam.
  - Algoritmo para Redução de Dimensionalidade: agrupa variáveis em componentes (resumo vetorial das variáveis que fazem parte do componente).
  - Gradient Boosting & AdaBoost

O Processo de Construção de Modelos de Machine Learning



Soluções de Machine Learning

Principais linguagens de programação para ML:	Principais frameworks para ML:
• Python • Linguagem R • Scala • Java • JavaScript • Go • C++ / C#	• Scikit-learn (Python) • Caret (R) • TensorFlow (Python, R, Java, C++) • Apache Mahout (Python, Java) • Spark Mllib (Scala, Java, Python, R) • H2O (Java, Python) • Weka (Java, Python) • PyTorch, CNTK, MXNet (Python, C++, Java)

A linguagem Python oferece duas vantagens principais sobre todas as outras soluções. Primeiro, por se tratar de uma linguagem de uso geral, ela pode ser usada com qualquer uma destas soluções. Segundo, Python possui uma das mais poderosas soluções gratuitas de Machine Learning, o Scikit-learn, que estudaremos neste capítulo.

Scikit learn:

- SciPy
- NumPy
- matplotlib
- pandas

Principais características do scikit learn:

- Excelente documentação

- Fácil e intuitivo
- Diversos datasets
- Licença BSD (permite ser usado para fins comerciais)
- Confiável

Deep Learning x Machine Learning x Inteligência Artificial

Recentemente o AlphaGo da Google (https://deepmind.com/alpha-go) derrotou o mestre Sul Coreano Lee Se-dol em uma partida do jogo de tabuleiro Go. Várias notícias foram espalhadas na mídia sobre este tema e os termos Deep Learning, Machine Learning e Inteligência Artificial foram usados de forma intercambiável para descrever como o Alpha Go venceu. Todos esses três conceitos são parte do motivo da vitória sobre Lee Se-Dol. Mas eles não são a mesma coisa.

A maneira mais fácil de pensar sobre o seu relacionamento entre esses conceitos é visualizá-los como círculos concêntricos com Inteligência Artificial—a ideia que veio primeiro—a maior, e então Machine Learning—que floresceu depois - e finalmente Deep Learning—que está liderando a explosão de Inteligência Artificial hoje—contendo parte das duas.

A Inteligência Artificial tem sido parte da nossa imaginação e fervor dentro de laboratórios de pesquisa desde que alguns cientistas lançaram o termo nas conferências de Dartmouth em 1956 e deram luz ao campo de Inteligência Artificial. Nas décadas desde então, Inteligência Artificial tem sido vista de forma alternada entre a chave do futuro mais brilhante da nossa tecnologia.

Nos últimos anos a popularidade de Inteligência Artificial explodiu, especialmente desde 2015. Muito disso tem a ver com a disponibilidade dos GPUs (Graphic Processing Units) que fazem com que processamento paralelo seja mais rápido, mais barato e mais poderoso. Também tem a ver com todo a enxurrada de dados que temos hoje na internet (todo o movimento de Big Data)—imagens, textos, transações, dados de mapas, sensores.

Vejamos as principais diferenças entre os 3 conceitos:

Inteligência Artificial

Naquela conferência no verão de 1956 o sonho dos pioneiros da Inteligência Artificial (IA) era de construir máquinas complexas—possibilitadas por computadores que emergiam na época—que possuíssem as mesmas características da inteligência humana. Esse é o conceito que pensamos como “IA genérica”—máquinas fabulosas que tem todos os nossos sentidos (e talvez até mais), toda a nossa razão e pensam como nós pensamos. Você já viu essas máquinas em filmes como amigos—C-3PO—e inimigos—O Exterminador do Futuro. Máquinas de IA genéricas ficaram nos filmes e na ficção cientifica por um bom motivo; ainda não conseguimos criar algo do tipo, pelo menos não ainda.

O que conseguimos fazer se encaixa no conceito de “IA limitada”. Tecnologias que são capazes de executar tarefas específicas tão bem quanto, ou até melhor, que nós humanos conseguimos. Exemplos de IA limitadas são tarefas como classificação de imagens em um serviço como o Pinterest ou reconhecimento de rostos no Facebook.

Esses são exemplos de IA limitadas na prática. Essas tecnologias exibem algumas facetas da inteligência humana. Mas como? De onde essa inteligência vem? Isso nos leva ao próximo círculo, Machine Learning.

Machine Learning

Machine Learning da maneira mais básica é a prática de usar algoritmos para coletar dados, aprender com eles e então fazer uma predição sobre alguma coisa. Então ao invés de implementar as rotinas de software na mão, com um set específico de instruções para completar uma tarefa em particular, a máquina é “treinada” usando uma quantidade grande de dados e algoritmos que dão e ela a habilidade de aprender como executar a tarefa.

Machine Learning veio direto das mentes do pessoal do início da IA e a abordagem com algoritmos através dos anos incluiu árvore de aprendizado, programação lógica indutiva, agrupamento, aprendizado reforçado, redes Bayesianas, entre outros. Como sabemos, nenhuma dessas soluções chegou ao objetivo final de uma IA genérica e mesmo uma IA limitada estava fora do nosso alcance com as abordagens iniciais de Machine Learning.

Da maneira que as coisas evoluíram, uma das melhores áreas de aplicação para Machine Learning por muitos anos foi a de visão computacional, apesar de ainda requerer muito trabalho manual para completar uma tarefa. Pessoas escrevem na mão classificadores como filtros detectores de bordas em imagens para que os programas consigam identificar onde um objeto começou e terminou; detectores de formato para determinar se algo na imagem tem oito lados; um classificador para reconhecer as letras “P-A-R-E”. De todos esses classificadores criados manualmente, eles construíram algoritmos que entendem uma imagem e “aprendem” a determinar se é uma placa de pare.

Legal, mas ainda nada que seja surpreendente. Especialmente em um dia com neblina quando a placa não é perfeitamente visível, ou se uma árvore tapa metade dela. Há uma razão pela qual visão computacional e detecção de imagens não chegava nem perto de rivalizar com humanos até muito recentemente, ela era muito rasa e propensa a erros. Tempo e o algoritmo de aprendizado certo fizeram toda a diferença.

Deep Learning

Outra abordagem em forma de algoritmo do início do movimento de Machine Learning, Redes Neurais Artificias surgiram e desapareceram através das décadas. Rede neurais são inspiradas pelo nosso entendimento da biologia do cérebro humano—todas as interconexões entre neurônios. Mas, diferente de um cérebro biológico onde qualquer neurônio pode se conectar com qualquer outro neurônio dentro de uma certa distância física, essas redes neurais artificiais têm camadas discretas, conexões e direções de propagação de dados.

Você pode, por exemplo, pegar uma imagem, cortá-la em uma pilha de pequenos pedaços que são recebidos pela primeira camada da rede neural. Na primeira camada neurônios individuais então passam os dados para uma segunda camada. A segunda camada faz o seu trabalho, e assim por diante, até que a camada final produza a saída.

Cada neurônio atribui um peso para os dados que entram—o quão correto ou incorreto ele é relativo à tarefa que está sendo executada. A saída final é então determinada pelo total desses pesos. Tome como exemplo a nossa placa de PARE. Atributos de uma foto de uma placa de pare são cortados e examinados pelos neurônios—o seu formato octogonal, a sua cor vermelha, as suas

letras distintas, o tamanho comum para placas de trânsito e o seu movimento (ou falta dele). O trabalho da rede neural é concluir se a imagem é de uma placa de pare ou não. Ela traz um “vetor de probabilidade”, que é um valor calculado a partir dos pesos atribuídos a imagem. Em nosso exemplo o sistema pode estar 87% confiante que a imagem é de uma placa de pare, 7% confiante que é uma placa de limite de velocidade e 5% que é um gato preso em uma árvore e assim por diante—a arquitetura da rede então diz para a rede neural se está certo ou não.

Mesmo esse exemplo está indo a frente, porque até recentemente redes neurais eram evitadas pela comunidade pesquisadora de IA. Elas estavam presentes desde o início de IA e haviam produzido muito pouco no sentido de “inteligência”. O problema era que mesmo a rede neural mais básica exigia muito computacionalmente, então era uma abordagem nem um pouco prática. Ainda assim, um grupo de pesquisa pequeno liderado por Geoffrey Hinton na Universidade de Toronto no Canadá se manteve firme, finalmente conseguindo paralelizar os algoritmos para supercomputadores executa-los e provar o conceito, mas só quando GPU’s foram incumbidos da tarefa que a promessa foi cumprida.

Se voltarmos para nosso exemplo da placa de pare, as chances são de que enquanto a rede está sendo ajustada ou “treinada”, está produzindo respostas erradas—recorrentemente. Ela precisa de treino. A rede precisa ver centenas de milhares, até milhões de imagens, até os pesos de cada informação recebida pelos neurônios estarem tão precisamente calibrados que conseguem responder de forma correta praticamente toda vez—com neblina ou sem neblina, com sol ou chuva. É nesse ponto que a rede neural aprendeu como que uma placa de pare se parece; ou a rosto de sua mãe no caso do Facebook; ou um gato, que é o que Andrew Ng fez na Google em 2012.

O grande avanço de Andrew Ng foi de pegar essas redes neurais, e essencialmente faze#las grandes, aumentar as amadas e os neurônios, e então alimentá-las com um nível massivo de dados para que fossem treinadas. No caso de Andrew Ng, eram imagens de 10 milhões de vídeos do YouTube. Andrew colocou a palavra “deep” no Deep Learning, que descreve todas as camadas nessas redes neurais.

Hoje, reconhecimento de imagens por máquinas treinadas através de Deep Learning em alguns cenários possuem uma taxa de acerto maior que a de humanos, e isso varia de gatos até identificar indicadores de câncer no sangue e tumores em exames de ressonância magnética. O AlphaGo da Google também aprendeu as regras do jogo e treinou para sua partida—calibrou sua rede neural—jogando contra si mesmo repetidamente.

Deep Learning permitiu muitas aplicações práticas de Machine Learning e por extensão o campo todo de IA. Deep Learning se quebra em diversas tarefas de maneira que todo tipo de ajuda de uma máquina é possível, mesmo as mais remotas. Carros que dirigem sozinhos, melhor saúde preventiva, recomendações de filmes, todos já estão aqui ou no horizonte. IA é o presente e o futuro. Com a ajuda de Deep Learning, IA pode até chegar no estado de ficção científica que imaginamos por tanto tempo.

Template Para Construção de Modelos de Machine Learning

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 6</font>
## Template Para Construção de Modelos de Machine Learning
Este notebook contém um template do código necessário para criar os principais algoritmos de
Machine Learning.
#>
from IPython.display import Image
Image(url = 'images/processo.png')
# Regressão Linear
#>
# Import do módulo
from sklearn import linear_model
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto linear regression
linear = linear_model.LinearRegression()
# Treinando o modelo com dados de treino e checando o score
linear.fit(x_treino, y_treino)
linear.score(x_treino, y_treino)
# Coletando os coeficientes
print('Coefficient: \n', linear.coef_)
print('Intercept: \n', linear.intercept_)
# Previsões
valores_previstos = linear.predict(x_teste)
# Regressão Logística
#>
# Import do módulo
from sklearn.linear_model import LogisticRegression
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto logistic regression
modelo = LogisticRegression()
# Treinando o modelo com dados de treino e checando o score
modelo.fit(x_treino, y_treino)
modelo.score(x_treino, y_treino)
# Coletando os coeficientes
print('Coefficient: \n', modelo.coef_)
print('Intercept: \n', modelo.intercept_)
# Previsões
valores_previstos = modelo.predict(x_teste)
# Árvores de Decisão
#>
# Import do módulo
from sklearn import tree
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto tree para regressão
modelo = tree.DecisionTreeRegressor()
# Criando o objeto tree para classificação
modelo = tree.DecisionTreeClassifier()
# Treinando o modelo com dados de treino e checando o score
modelo.fit(x_treino, y_treino)
modelo.score(x_treino, y_treino)
# Previsões
valores_previstos = modelo.predict(x_teste)
# Naive Bayes
#>
# Import do módulo
from sklearn.naive_bayes import GaussianNB
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto GaussianNB
modelo = GaussianNB()
# Treinando o modelo com dados de treino
modelo.fit(x_treino, y_treino)
```

```
# Previsões
valores_previstos = modelo.predict(x_teste)
# Support Vector Machines
#>
# Import do módulo
from sklearn import svm
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto de classificação SVM
modelo = svm.svc()
# Treinando o modelo com dados de treino e checando o score
modelo.fit(x_treino, y_treino)
modelo.score(x_treino, y_treino)
# Previsões
valores_previstos = modelo.predict(x_teste)
# K-Nearest Neighbors
#>
# Import do módulo
from sklearn.neighbors import KNeighborsClassifier
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto de classificação KNeighbors
KNeighborsClassifier(n_neighbors = 6) # Valor default é 5
# Treinando o modelo com dados de treino
modelo.fit(X, y)
# Previsões
valores_previstos = modelo.predict(x_teste)
# K-Means
#>
# Import do módulo
from sklearn.cluster import KMeans
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto KNeighbors
k_means = KMeans(n_clusters = 3, random_state = 0)
# Treinando o modelo com dados de treino
modelo.fit(x_treino)
# Previsões
valores_previstos = modelo.predict(x_teste)
# Random Forest
#>
# Import Library
from sklearn.ensemble import RandomForestClassifier
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto Random Forest
model = RandomForestClassifier()
# # Treinando o modelo com dados de treino
modelo.fit(x_treino, x_teste)
# Previsões
valores_previstos = modelo.predict(x_teste)
# Redução de Dimensionalidade
#>
# Import do módulo
from sklearn import decomposition
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Create objeto PCA
pca= decomposition.PCA(n_components = k)
# Para Factor analysis
fa= decomposition.FactorAnalysis()
# Reduzindo a dimensão do dataset de treino usando PCA
treino_reduzido = pca.fit_transform(treino)
# Reduzindo a dimensão do dataset de teste
teste_reduzido = pca.transform(teste)
# Gradient Boosting & AdaBoost
#>
# Import do módulo
from sklearn.ensemble import GradientBoostingClassifier
```

```
# Datasets de treino e de teste
x_treino = dataset_treino_variaveis_preditoras
y_treino = dataset_treino_variavel_prevista
x_teste = dataset_teste_variaveis_preditoras
# Criando o objeto Gradient Boosting
modelo = GradientBoostingClassifier(n_estimators = 100, learning_rate = 1.0, max_depth = 1,
random_state = 0)
# Treinando o modelo com dados de treino
modelo.fit(x_treino, y_treino)
# Previsões
valores_previstos = modelo.predict(x_teste)
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Definição do Problema do Negócio /
Extraindo e Carregando os Dados /
Análise Exploratória dos Dados: Estatística Descritiva, Correlação, Matplotlib, Seaborn /
Pré-Processamento – Normalização /
Pré-Processamento – Padronização e Binarização /
Feature Selection – Seleção Univariada /
Feature Selection – Eliminação Recursiva (RFE) /
Feature Selection – Método Ensemble para Seleção de Variáveis /
Redução de Dimensionalidade com Principal Component Analysis (PCA) /
Resampling e Divisão em Dados de Treino e Teste /
Cross Validation /
Avaliando Performance – Métricas Para Modelos de Classificação – Acurácia, ROC, Confusion Matrix /
Algoritmos de Classificação – Regressão Logística, LDA, Naive Bayes, NKK, CART, SVM /
Seleção do Modelo Preditivo *(Usando Programação)* /
Otimização do Modelo – Ajuste de Hyperparâmetros /
Grid Search Parameter Tuning e Random Search Parameter Tuning /
Salvando e Carregando o Modelo Treinado /
Métodos Ensemble – Bagged Decision Trees, RandomForest, AdaBoost, Gradient Boosting /
Classificação com Algorítmo XGBoost:

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 6</font>
# Machine Learning em Python - Parte 1 - Classificação
# >
from IPython.display import Image
Image(url = 'images/processo.png')
# >
import sklearn as sl
import warnings
warnings.filterwarnings("ignore")
sl.__version__
## Definição do Problema de Negócio
Vamos criar um modelo preditivo que seja capaz de prever se uma pessoa pode ou não desenvolver
diabetes. Para isso, usaremos dados históricos de pacientes, disponíveis no dataset abaixo.
Dataset: Pima Indians Diabetes Data Set
```

http://archive.ics.uci.edu/ml/datasets/diabetes
Este dataset descreve os registros médicos entre pacientes do Pima Inidians e cada registro está marcado se o paciente desenvolveu ou não diabetes.
Informações sobre os atributos:
1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)
Extraindo e Carregando os Dados
Existem diversas considerações ao se carregar dados para o processo de Machine Learning. Por exemplo: seus dados possuem um header (cabeçalho)? Caso negativo, você vai precisar definir o título para cada coluna. Seus arquivos possuem comentários? Qual o delimitador das colunas? Alguns dados estão entre aspas, simples ou duplas?
>
```
# Carregando arquivo csv usando NumPy
import numpy as np
arquivo = 'data/pima-data.csv'
arquivo_data = open(arquivo, 'rb')
dados = np.loadtxt(arquivo_data, delimiter = ",")
print(dados.shape)
```
>
```
# Carregando arquivo csv usando Pandas
import pandas as pd
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = pd.read_csv(arquivo, names = colunas)
print(dados.shape)
```
>
```
# Carregando arquivo csv usando Pandas (método que usaremos neste notebook)
from pandas import read_csv
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
print(dados.shape)
```
Análise Exploratória de Dados
Estatística Descritiva
>
```
# Visualizando as primeiras 20 linhas
dados.head(20)
```
Se o número de linhas no seu arquivo for muito grande, o algoritmo pode levar muito tempo para ser treinado. Se o número de registros for muito pequeno, você pode não ter registros suficientes para treinar seu modelo.
Se você tiver muitas colunas em seu arquivo, o algoritmo pode apresentar problemas de performance devido a alta dimensionalidade.
A melhor solução vai depender de cada caso. Mas lembre-se: treine seu modelo em um subset do seu conjunto de dados maior e depois aplique o modelo a novos dados.
>
```
# Visualizando as dimensões
dados.shape
```
O tipo dos dados é muito importante. Pode ser necessário converter strings, ou colunas com números inteiros podem representar variáveis categóricas ou valores ordinários.
>
```
# Tipo de dados de cada atributo
dados.dtypes
```
>
```
# Sumário estatístico
dados.describe()
```
Em problemas de classificação pode ser necessário balancear as classes. Classes desbalanceadas (ou seja, volume maior de um dos tipos das classes) são comuns e precisam ser tratadas durante a fase de pré-processamento. Podemos ver abaixo que existe uma clara desproporção entre as classes 0 (não ocorrência de diabetes) e 1 (ocorrência de diabetes).
>
```
# Distribuição das classes
dados.groupby('class').size()
```
A correlação é o relacionamento entre 2 variáveis. O método mais comum para calcular correlação é o método de Pearson, que assume uma distribuição normal dos dados. Correlação de -1 mostra uma correlação negativa, enquanto uma correlação de +1 mostra uma correlação positiva. Uma correlação igual a 0 mostra que não há relacionamento entre as variáveis.
Alguns algoritmos como regressão linear e regressão logística podem apresentar problemas de performance se houver atributos altamente correlacionados (colineares).
>
```
# Correlação de Pearson
dados.corr(method = 'pearson')
```
Skew (ou simetria) se refere a distribuição dos dados que é assumida ser normal ou gaussiana (bell

```
curve). Muitos algoritmos de Machine Learning consideram que os dados possuem uma distribuição
normal. Conhecendo a simetria dos dados, permite que você faça uma preparação e entregue o que o
algoritmo espera receber, aumentado desta forma a acurácia do modelo preditivo.
# >
# Verificando o skew de cada atributo
dados.skew()
### Visualização com Matplotlib
# >
import matplotlib.pyplot as plt
# Por se tratar de um conjunto de gráficos menores, pode ser mais interessante gerar os gráficos em
janela separada
%matplotlib inline
Com o histograma podemos rapidamente avaliar a distribuição de cada atributo. Os histogramas
agrupam os dados em bins e fornecem uma contagem do número de observações em cada bin. Com o
histograma, você pode rapidamente verificar a simetria dos dados e se eles estão em distribuição
normal ou não. Isso também vai ajudar na identificação dos outliers.
Podemos ver que os atributos age, pedi e test possuem uma distribuição exponencial. Podemos ver que
as colunas mass e press possuem uma distribuição normal.
# >
# Histograma Univariado
dados.hist()
plt.show()
Os Density Plots são outra forma de visualizar a distribuição dos dados para cada atributo. O plot
é como uma espécie de histograma abstrato com uma curva suave através do topo dos bins de um
histograma. Pode ser mais fácil identificar a distribuição dos dados usando um density plot.
# >
# Density Plot Univariado
dados.plot(kind = 'density', subplots = True, layout = (3,3), sharex = False)
plt.show()
Com os boxplots também podemos revisar a distribuição dos dados para cada atributo. A linha no
centro (vermelho) é o valor da mediana (quartil 50%), a linha abaixo é o quartil 25% e a linha
acima o quartil 75%. O boxplot ajuda a ter uma ideia da dispersão dos dados e os possíveis
outliers.
Podemos ver que a dispersão dos dados é bem diferente entre os atributos. As colunas age, skin e
test possuem uma simetria muito próxima a valores de dados menores.
# >
# Box and Whisker Plots
dados.plot(kind = 'box', subplots = True, layout = (3,3), sharex = False, sharey = False)
plt.show()
# >
# Matriz de Correlação com nomes das variáveis
correlations = dados.corr()
# Plot
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin = -1, vmax = 1)
fig.colorbar(cax)
ticks = np.arange(0, 9, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(colunas)
ax.set_yticklabels(colunas)
plt.show()
# >
# Matriz de Correlação genérica
correlations = dados.corr()
# Plot
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin = -1, vmax = 1)
fig.colorbar(cax)
plt.show()
Um scatterplot mostra o relacionamento entre duas variáveis como pontos em duas dimensões, sendo um
eixo para cada variável. Podemos criar um scatterplot para cada par de variáveis em nosso dataset.
A exemplo da matriz de correlação, o scatterplot matrix é simétrico.
# >
# Scatter Plot
from pandas.plotting import scatter_matrix
scatter_matrix(dados)
plt.show()
### Visualização com Seaborn
# >
import seaborn as sns
# >
# Pairplot
sns.pairplot(dados)
# >
# Boxplot com orientação vertical
```

```
sns.boxplot(data = dados, orient = "v")
# >
# Clustermap
sns.clustermap(dados)
# >
dados.describe
# >
from scipy import stats
sns.distplot(dados.pedi, fit = stats.norm);
## Preparando os Dados para Machine Learning
Muitos algoritmos esperam receber os dados em um formato específico. É seu trabalho preparar os
dados em uma estrutura que seja adequada ao algoritmo que você está utilizando.
É muito provável que você tenha que realizar tarefas de pré-processamento nos dados. Esse é um
passo necessário dentro do processo. O desafio é o fato que cada algoritmo requer uma estrutura
diferente, o que pode requerer transformações diferentes nos dados. Mas é possível em alguns casos,
obter bons resultados sem um trabalho de pré-processamento. Mas é uma boa prática criar diferentes
visões e transformações dos dados, de modo a poder testar diferentes algoritmos de Machine
Learning.
## Normalização - Método 1
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
E uma das primeiras tarefas dentro do pré-processamento, é colocar seus dados na mesma escala.
Muitos algoritmos de Machine Learning vão se beneficiar disso e produzir resultados melhores. Esta
etapa também é chamada de normalização e significa colocar os dados em uma escala com range entre 0
e 1. Isso é útil para a otimização, sendo usado no core dos algoritmos de Machine Learning, como
gradient descent. Isso também é útil para algoritmos como regressão e redes neurais e algoritmos
que usam medidas de distância, como KNN. O scikit-learn possui uma função para esta etapa, chamada
MinMaxScaler().
# >
# Transformando os dados para a mesma escala (entre 0 e 1)
# Import dos módulos
from pandas import read_csv
from sklearn.preprocessing import MinMaxScaler
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input (X) e output (Y)
X = array[:,0:8]
Y = array[:,8]
# Gerando a nova escala (normalizando os dados)
scaler = MinMaxScaler(feature_range = (0, 1))
rescaledX = scaler.fit_transform(X)
# Sumarizando os dados transformados
print("Dados Originais: \n\n", dados.values)
print("\nDados Normalizados: \n\n", rescaledX[0:5,:])
## Normalização - Método 2
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
No scikit-learn,
 normalização se refere a ajustar a escala de cada observação (linha) de modo que ela tenha
comprimento igual a 1 (chamado vetor de comprimento 1 em álgebra linear). Este método de pré-
processamento é útil quando temos datasets esparsos (com muitos zeros) e atributos com escala muito
variada. Útil quando usamos algoritmos de redes neurais ou que usam medida de distância, como KNN.
O scikit-learn possui uma função para esta etapa, chamada Normalizer().
# >
# Normalizando os dados (comprimento igual a 1)
from pandas import read_csv
from sklearn.preprocessing import Normalizer
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Gerando os dados normalizados
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# Sumarizando os dados transformados
print("Dados Originais: \n\n", dados.values)
print("\nDados Normalizados: \n\n", normalizedX[0:5,:])
## Padronização
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
Padronização é a técnica para transformar os atributos com distribuição Gaussiana (normal) e
diferentes médias e desvios padrão em uma distribuição Gaussiana com a média igual a 0 e desvio
padrão igual a 1. Isso é útil para algoritmos que esperam que os dados estejam com uma distribuição
Gaussiana, como regressão linear, regressão logística e linear discriminant analysis. Funciona bem
quando os dados já estão na mesma escala. O scikit-learn possui uma função para esta etapa, chamada
```

```
StandardScaler().
# >
# Padronizando os dados (0 para a média, 1 para o desvio padrão)
# Import dos módulos
from pandas import read_csv
from sklearn.preprocessing import StandardScaler
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Gerando o novo padrão
scaler = StandardScaler().fit(X)
standardX = scaler.transform(X)
# Sumarizando os dados transformados
print("Dados Originais: \n\n", dados.values)
print("\nDados Padronizados: \n\n", standardX[0:5,:])
## Binarização (Transformar os Dados em Valores Binários)
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Binarizer.html
Nós podemos definir um valor em nossos dados, ao qual chamamos de threshold e então definimos que
todos os valores acima do threshold serão marcados como sendo 1 e todos valores iguais ou abaixo do
threshold serão marcados como sendo 0. Isso é o que chamamos de Binarização. Isso é útil quando
temos probabilidades e queremos transformar os dados em algo com mais significado. O scikit-learn
possui uma função para esta etapa, chamada Binarizer().
# >
# Binarização
# Import dos módulos
from pandas import read_csv
from sklearn.preprocessing import Binarizer
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Gerando a binarização
binarizer = Binarizer(threshold = 0.2).fit(X)
binaryX = binarizer.transform(X)
# Sumarizando os dados transformados
print("Dados Originais: \n\n", dados.values)
print("\nDados Binarizados: \n\n", binaryX[0:5,:])
## Feature Selection
Os atributos presentes no seu dataset e que você utiliza nos dados de treino, terão grande
influência na precisão e resultado do seu modelo preditivo. Atributos irrelevantes terão impacto
negativo na performance, enquanto atributos colineares podem afetar o grau de acurácia do modelo. O
Scikit-learn possui funções que automatizam o trabalho de extração e seleção de variáveis.
A etapa de Feature Selection é onde selecionamos os atributos (variáveis) que serão melhores
candidatas a variáveis preditoras. O Feature Selection nos ajuda a reduzir o overfitting (quando o
algoritmo aprende demais), aumenta a acurácia do modelo e reduz o tempo de treinamento.
## Seleção Univariada
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
Testes estatísticos podem ser usados para selecionar os atributos que possuem forte relacionamento
com a variável que estamos tentando prever. O Scikit-learn fornece a função SelectKBest() que pode
ser usada com diversos testes estatísticos, para selecionar os atributos. Vamos usar o teste qui-
quadrado e selecionar os 4 melhores atributos que podem ser usados como variáveis preditoras.
# >
# Extração de Variáveis com Testes Estatísticos Univariados (Teste qui-quadrado neste exemplo)
# Import dos módulos
from pandas import read_csv
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Função para seleção de variáveis
best_var = SelectKBest(score_func = chi2, k = 4)
# Executa a função de pontuação em (X, y) e obtém os recursos selecionados
fit = best_var.fit(X, Y)
# Reduz X para os recursos selecionados
```

```
features = fit.transform(X)
# Resultados
print('\nNúmero original de features:', X.shape[1])
print('\nNúmero reduzido de features:', features.shape[1])
print('\nFeatures (Variáveis Selecionadas): \n\n', features)
## Eliminação Recursiva de Atributos
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html
Esta é outra técnica para seleção de atributos, que recursivamente remove os atributos e constrói o
modelo com os atributos remanescentes. Esta técnica utiliza a acurácia do modelo para identificar
os atributos que mais contribuem para prever a variável alvo. Em inglês esta técnia é chamada
Recursive Feature Elimination (RFE).
O exemplo abaixo utiliza a técnica de eliminação recursiva de atributos com um algoritmo de
Regressão Logística para selecionar as 3 melhores variáveis preditoras. O RFE selecionou as
variáveis preg, mass e pedi, que estão marcadas como True em "Atributos Selecionados" e com valor 1
em "Ranking dos Atributos".
# >
# Eliminação Recursiva de Variáveis
# Import dos módulos
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Criação do modelo
modelo = LogisticRegression()
# RFE
rfe = RFE(modelo, 3)
fit = rfe.fit(X, Y)
# Print dos resultados
print("Variáveis Preditoras:", dados.columns[0:8])
print("Variáveis Selecionadas: %s" % fit.support_)
print("Ranking dos Atributos: %s" % fit.ranking_)
print("Número de Melhores Atributos: %d" % fit.n_features_)
## Método Ensemble para Seleção de Variáveis
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html
Bagged Decision Trees, como o algoritmo RandomForest (esses são chamados de Métodos Ensemble),
podem ser usados para estimar a importância de cada atributo. Esse método retorna um score para
cada atributo.
Quanto maior o score, maior a importância do atributo.
# >
# Importância do Atributo com o Extra Trees Classifier
# Import dos Módulos
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Criação do Modelo - Feature Selection
modelo = ExtraTreesClassifier()
modelo.fit(X, Y)
# Print dos Resultados
print(dados.columns[0:8])
print(modelo.feature_importances_)
## Redução de Dimensionalidade (Feature Extraction)
## Principal Component Analysis (PCA)
O PCA foi inventado em 1901 por Karl Pearson e utiliza álgebra linear para transformar datasets em
uma forma comprimida, o que é geralmente conhecido como Redução de Dimensionalidade. Com PCA você
pode escolher o número de dimensões (chamados componentes principais) no resultado transformado.
Vamos usar PCA para selecionar 3 componentes principais.
A Análise de Componentes Principais (PCA) é um método para extração das variáveis importantes (na
forma de componentes) a partir de um grande conjunto de variáveis, disponíveis em um conjunto de
dados. Esta técnica permite extrair um número pequenos de conjuntos dimensionais a partir de um
dataset altamente dimensional. Com menos variáveis a visualização também se torna muito mais
significativa. PCA é mais útil quando se lida com 3 ou mais dimensões.
# >
# Image source: http://www.nlpca.org/pca_principal_component_analysis.html
from IPython.display import Image
Image(url = 'images/PCA2.png')
```

Cada componente resultante é uma combinação linear de n atributos. Ou seja, cada componente principal é uma combinação de atributos presentes no dataset. O Primeiro Componente Principal é a combinação linear dos atributos com máxima variância e determina a direção em que há mais alta variabilidade nos dados. Quanto maior a variabilidade capturada no primeiro componente principal, mais informação será capturada pelo componente. O Segundo Componente Principal captura a variabilidade remanescente. Todos os componentes
subsequentes possuem o mesmo conceito.

```
# >
# Image source: http://www.nlpca.org/pca_principal_component_analysis.html
from IPython.display import Image
Image(url = 'images/PCA3.png')
```

O PCA precisa ser alimentado com dados normalizados. Utilizar o PCA em dados não normalizados pode gerar resultados inesperados.
A análise de componentes principais é uma técnica da estatística multivariada que consiste em transformar um conjunto de variáveis originais em outro conjunto de variáveis denominadas de componentes principais. Os componentes principais apresentam propriedades importantes: cada componente principal é uma combinação linear de todas as variáveis originais, são independentes entre si e estimados com o propósito de reter, em ordem de estimação, o máximo de informação, em termos da variação total contida nos dados. Os componentes principais são garantidamente independentes apenas se os dados forem normalmente distribuídos (conjuntamente).
Procura-se redistribuir a variação observada nos eixos originais de forma a se obter um conjunto de eixos ortogonais não correlacionados. Esta técnica pode ser utilizada para geração de índices e agrupamento de indivíduos. A análise agrupa os indivíduos de acordo com sua variação, isto é, os indivíduos são agrupados segundo suas variâncias, ou seja, segundo seu comportamento dentro da população, representado pela variação do conjunto de características que define o indivíduo, ou seja, a técnica agrupa os indivíduos de uma população segundo a variação de suas características.
A análise de componentes principais é associada à idéia de redução de massa de dados, com menor perda possível da informação.
O objetivo é sumarizar os dados que contém muitas variáveis (p) por um conjunto menor de variáveis (k) compostas derivadas a partir do conjunto original. PCA usa uma conjunto de dados representado por uma matriz de n registros por p atributos, que podem estar correlacionados, e sumariza esse conjunto por eixos não correlacionados (componentes principais) que são uma combinação linear das p variáveis originais. As primeiras k componentes contém a maior quantidade de variação dos dados.
Em termos gerais a PCA busca reduzir o número de dimensões de um dataset, projetando os dados em um novo plano. Usando essa nova projeção os dados originais, que podem envolver diversas variáveis, podem ser interpretados utilizando menos "dimensões."
No dataset reduzido podemos observar com mais clareza tendências, padrões e/ou outliers. Mas vale lembrar que a regra: "Se não está nos dados brutos não existe!" é sempre válida. A PCA fornece apenas mais clareza aos padrões que já estão lá.

```
# >
from IPython.display import Image
Image(url = 'images/PCA.png')
```

Quanto maior a variância, maior a quantidade de informação contida no componente.

```
# >
# Feature Extraction
# Import dos módulos
from pandas import read_csv
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Normalizando os dados
scaler = MinMaxScaler(feature_range = (0, 1))
rescaledX = scaler.fit_transform(X)
# Seleção de atributos
pca = PCA(n_components = 4)
fit = pca.fit(rescaledX)
# Sumarizando os componentes
print("Variância: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

Amostragem - Resampling

Você precisa saber se seu modelo preditivo vai funcionar bem quando receber novos dados. A melhor maneira de avaliar a performance do modelo é fazer previsões em dados que você já conhece o resultado. Outra maneira de testar a performance do seu modelo é utilizar técnicas estatísticas como métodos de amostragem que permitem você estimar quão bem seu modelo irá fazer previsões em novos dados.
A avaliação do modelo é uma estimativa de quão bem o algoritmo será capaz de prever em novos dados. Isso não garante performance. Após avaliar o modelo, nós podemos treiná-lo novamente com os dados de treino e então prepará-lo para uso operacional em produção. Existem diversas técnicas para isso e estudaremos duas aqui: Conjunto de dados de treino e de teste e Cross Validation.

Dados de Treino e de Teste

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
Este é o método mais utilizado para avaliar performance de um algoritmo de Machine Learning.

```
Dividimos nossos dados originais em dados de treino e de teste. Treinamos o algoritmo nos dados de
treino e fazemos as previsões nos dados de teste e avaliamos o resultado. A divisão dos dados vai
depender do seu dataset, mas utiliza-se com frequência tamanhos entre 70/30 (treino/teste) e 65/35
(treino/teste).
Este método é bem veloz e ideal para conjuntos de dados muito grandes. O ponto negativo é a
possibilidade de alta variância.
# >
# Avaliação usando dados de treino e de teste
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo o tamanho das amostras
teste_size = 0.33
# Garante que os resultados podem ser reproduzidos
# Isso é importante para comparar a acurácia com outros algoritmos de Machine Learning.
seed = 7
# Criando os conjuntos de dados de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = teste_size, random_state
= seed)
# Criação do modelo
modelo = LogisticRegression()
# Treinamento do modelo
modelo.fit(X_treino, Y_treino)
# Score do modelo nos dados de teste
result = modelo.score(X_teste, Y_teste)
print("Acurácia nos Dados de Teste: %.3f%%" % (result * 100.0))
### Cross Validation
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
Cross Validation é uma técnica que pode ser utilizada para avaliar a performance de um modelo com
menos variância que a técnica de dividir os dados em treino/teste. Com esta técnica dividimos os
dados em partes normalmente chamadas de k-folds (por exemplo k = 5, k = 10). Cada parte é chamada
fold. O algoritmo é treinado em k-1 folds. Cada fold é usado no treinamento de forma repetida e um
fold por vez. Após executar o processo em k-1 folds, podemos sumarizar a performance em cada fold
usando a média e o desvio padrão (Eu disse que Estatística era importante no processo de Big Data
Analytics). O resultado é normalmente mais confiável e oferece maior acurácia ao modelo. A chave
deste processo está em definir o correto valor de k, de modo que o número de folds represente
adequadamente o número de repetições necessárias.
# >
from IPython.display import Image
Image(url = 'images/cross-validation.jpg')
# >
# Avaliação usando Cross Validation
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para os folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = LogisticRegression()
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Usamos a média e o desvio padrão
print("Acurácia Final: %.3f%%" % (resultado.mean() * 100.0))
## Avaliando a Performance
As métricas que você escolhe para avaliar a performance do modelo vão influenciar a forma como a
performance é medida e comparada com modelos criados com outros algoritmos.
Vamos utilizar o mesmo algoritmo, mas com métricas diferentes e assim comparar os resultados. A
função cross_validation.cross_val_score() será usada para avaliar a performance.
```

```
### Métricas para Algoritmos de Classificação
https://scikit-learn.org/stable/modules/model_evaluation.html
# >
# Acurácia
# Número de previsões corretas. É útil apenas quando existe o mesmo número de observações em cada
classe.
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = LogisticRegression()
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold, scoring = 'accuracy')
# Print dos resultados
print("Acurácia: %.3f" % (resultado.mean() * 100))
# >
# Curva ROC
# A Curva ROC permite analisar a métrica AUC (Area Under the Curve).
# Essa é uma métrica de performance para classificação binária, em que podemos definir as classes
em positiavs e negativas.
# Problemas de classificação binária são um trade-off sentre Sensitivity e Specifity.
# Sensitivity é a taxa de verdadeiros positivos (TP). Ese é o número de instâncias positivas da
primeira classe que foram previstas corretamente.
# Specifity é a taxa de verdadeiros
 negativos (TN). Esse é o número de instâncias da segunda classe que foram previstas corretamente.
# Valores acima de 0.5 indicam uma boa taxa de previsão.
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
model = LogisticRegression()
# Cross Validation
resultado = cross_val_score(model, X, Y, cv = kfold, scoring = 'roc_auc')
# Print do resultado
print("AUC: %.3f" % (resultado.mean() * 100))
# >
# Confusion Matrix
# Permite verificar a acurácia em um formato de tabela
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
```

```
X = array[:,0:8]
Y = array[:,8]
# Definindo o tamanho do conjunto de dados
teste_size = 0.33
seed = 7
# Dividindo os dados em treino e teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = teste_size, random_state
= seed)
# Criando o modelo
model = LogisticRegression()
model.fit(X_treino, Y_treino)
# Fazendo as previsões e construindo a Confusion Matrix
previsoes = model.predict(X_teste)
matrix = confusion_matrix(Y_teste, previsoes)
# Imprimindo a Confusion Matrix
print(matrix)
# >
# Relatório de Classificação
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo o tamanho do conjunto de dados
teste_size = 0.33
seed = 7
# Dividindo os dados em treino e teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = teste_size, random_state
= seed)
# Criando o modelo
modelo = LogisticRegression()
modelo.fit(X_treino, Y_treino)
# Fazendo as previsões e construindo o relatório
previsoes = model.predict(X_teste)
report = classification_report(Y_teste, previsoes)
# Imprimindo o relatório
print(report)
# Algoritmos de Classificação
Não temos como saber qual algoritmo vai funcionar melhor na construção do modelo, antes de
testarmos o algoritmo com nosso dataset. O ideal é testar alguns algoritmos e então escolher o que
fornece melhor nível de precisão. Vamos testar um conjunto de algoritmos de classificação, nas
mesmas condições.
## Regressão Logística
Algoritmo Linear. O algoritmo de Regressão Logística assume que seus dados estão em uma
Distribuição Normal para valores numéricos que podem ser modelados com classificação binária.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = LogisticRegression()
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
## Linear Discriminant Analysis
```

Algoritmo Linear. Técnica estatística para classificação binária. Também assume que os dados estão em Distribuição Normal.

```
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = LinearDiscriminantAnalysis()
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
```

KNN - K-Nearest Neighbors

Algoritmo Não-Linear que utiliza uma métrica de distância para encontrar o valor de K mais adequado as instâncias do dataset de treino.

```
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
random_state = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = random_state)
# Criando o modelo
modelo = KNeighborsClassifier()
# Cross Validation
results = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
```

Naive Bayes

Algoritmo Não-Linear. Calcula a Probabilidade de cada classe e a probabilidade condicional de cada classe dado uma variável de entrada. As probabilidades são então estimadas para os novos dados e multiplicadas, assumindo que são independentes (suposição simples ou Naive). Assume dados em distirbuição Gaussiana (Normal)

```
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
```

```
# Criando o modelo
modelo = GaussianNB()
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
## CART (Classification and Regression Trees)
Algoritmo Não-Linear. O algoritmo CART constrói uma árvore binária a partir do dataset de treino.
Cada atributo e cada valor de cada atributo são avaliados com o objetivo de reduzir a função de
custo (Cost Function).
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = DecisionTreeClassifier()
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
## SVM - Support Vector Machines
O objetivo deste algoritmo é buscar uma linha que melhor separa duas classes dentro de um conjunto
de dados. As instâncias de dados que estão mais próximas desta linha que separa as classes, são
chamadas support vectors. O SVM tem sido estendido para suportar multiclasses.
Support Vector Machines são algoritmos de classificação muito poderosos. Quando usados em conjunto
com “Random forest” e outras ferramentas de aprendizagem de máquina, dão uma dimensão muito
diferente para montagem de modelos. Assim, eles se tornam cruciais para os casos em que é
necessária um poder de previsão muito elevado. Esses algoritmos são um pouco mais difíceis de
visualizar devido à complexidade na formulação.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds
 = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = SVC()
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
# Seleção do Modelo Preditivo
Veremos que os algoritmos de Regressão Logística e Linear Discriminant Analysis apresentaram o
melhor nível de precisão.
# >
# Import dos módulos
from pandas import read_csv
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Preparando a lista de modelos
modelos = []
modelos.append(('LR', LogisticRegression()))
modelos.append(('LDA', LinearDiscriminantAnalysis()))
modelos.append(('NB', GaussianNB()))
modelos.append(('KNN', KNeighborsClassifier()))
modelos.append(('CART', DecisionTreeClassifier()))
modelos.append(('SVM', SVC()))
# Avaliando cada modelo em um loop
resultados = []
nomes = []
for nome, modelo in modelos:
    kfold = KFold(n_splits = num_folds, random_state = seed)
    cv_results = cross_val_score(modelo, X, Y, cv = kfold, scoring = 'accuracy')
    resultados.append(cv_results)
    nomes.append(nome)
    msg = "%s: %f (%f)" % (nome, cv_results.mean(), cv_results.std())
    print(msg)
# Boxplot para comparar os algoritmos
fig = plt.figure()
fig.suptitle('Comparação de Algoritmos de Classificação')
ax = fig.add_subplot(111)
plt.boxplot(resultados)
ax.set_xticklabels(nomes)
plt.show()
## Otimização do Modelo - Ajuste de Hyperparâmetros
Todos os algoritmos de Machine Learning são parametrizados, o que significa que você pode ajustar a
performance do seu modelo preditivo, através do tuning (ajuste fino) dos parâmetros. Seu trabalho é
encontrar a melhor combinação entre os parâmetros em cada algoritmo de Machine Learning. Esse
processo também é chamado de Otimização de Hyperparâmetros. O scikit-learn oferece dois métodos
para otimização automática dos parâmetros: Grid Search Parameter Tuning e Random Search Parameter
Tuning.
### Grid Search Parameter Tuning
Este método realiza metodicamente combinações entre todos os parâmetros do algoritmo, criando um
grid. Vamos experimentar este método utilizando o algoritmo de Regressão Logística.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores que serão testados
valores_grid = {'penalty': ['l1','l2'], 'C': [0.001,0.01,0.1,1,10,100,1000]}
# Criando o modelo
modelo = LogisticRegression()
# Criando o grid
grid = GridSearchCV(estimator = modelo, param_grid = valores_grid)
grid.fit(X, Y)
# Print do resultado
print("Acurácia: %.3f" % (grid.best_score_ * 100))
print("Melhores Parâmetros do Modelo:\n", grid.best_estimator_)
### Random Search Parameter Tuning
Este método gera amostras dos parâmetros dos algoritmos a partir de uma distribuição randômica
uniforme para um número fixo de iterações. Um modelo é construído e testado para cada combinação de
parâmetros.
```

```
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores que serão testados
seed = 7
iterations = 14
# Definindo os valores que serão testados
valores_grid = {'penalty': ['l1','l2'], 'C': [0.001,0.01,0.1,1,10,100,1000]}
# Criando o modelo
modelo = LogisticRegression()
# Criando o grid
rsearch = RandomizedSearchCV(estimator = modelo,
                             param_distributions = valores_grid,
                             n_iter = iterations,
                             random_state = seed)
rsearch.fit(X, Y)
# Print dos resultados
print("Acurácia: %.3f" % (rsearch.best_score_ * 100))
print("Melhores Parâmetros do Modelo:\n", rsearch.best_estimator_)
# Salvando o resultado do seu trabalho
# >
# Salvando o resultado do seu trabalho
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import pickle
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo o tamanho dos dados de treino e de teste
teste_size = 0.33
seed = 7
# Criando o dataset de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = teste_size, random_state
= seed)
# Criando o modelo
modelo = LogisticRegression()
# Treinando o modelo
modelo.fit(X_treino, Y_treino)
# Salvando o modelo
arquivo = 'modelos/modelo_classificador_final.sav'
pickle.dump(modelo, open(arquivo, 'wb'))
print("Modelo salvo!")
# Carregando o arquivo
modelo_classificador_final = pickle.load(open(arquivo, 'rb'))
modelo_prod = modelo_classificador_final.score(X_teste, Y_teste)
print("Modelo carregado!")
# Print do resultado
print("Acurácia: %.3f" % (modelo_prod.mean() * 100))
# Otimizando Performance com Métodos Ensemble
Métodos Ensemble permitem aumentar consideravelmente o nível de precisão nas suas previsões.
Veremos como criar alguns dos Métodos Ensemble mais poderosos em Python. Existem 3 métodos
principais para combinar previsões a partir de diferentes modelos:
Bagging - Para construção de múltiplos modelos (normalmente do mesmo tipo) a partir de diferentes
subsets no dataset de treino.
Boosting - Para construção de múltiplos modelos (normalmente do mesmo tipo), onde cada modelo
aprende a corrigir os erros gerados pelo modelo anterior, dentro da sequência de modelos criados.
Voting - Para construção de múltiplos modelos (normalmente de tipos diferentes) e estatísticas
simples (como a média) são usadas para combinar as previsões.
Vejamos como utilizar estes métodos.
### Bagged Decision Trees
Este método funciona bem quando existe alta variância nos dados
# >
# Import dos módulos
```

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Cria o modelo unitário (classificador fraco)
cart = DecisionTreeClassifier()
# Definindo o número de trees
num_trees = 100
# Criando o modelo bagging
modelo = BaggingClassifier(base_estimator = cart, n_estimators = num_trees, random_state = seed)
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
### Random Forest
Random Forest é uma extensão do Baggig Decision Tree. Amostras do dataset de treino são usadas com
reposição, mas as árvores são criadas de uma forma que reduz a correlação entre classificadores
individuais (Random Forest é um conjunto de árvores de decisão).
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Definindo o número de trees
num_trees = 100
max_features = 3
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = RandomForestClassifier(n_estimators = num_trees, max_features = max_features)
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
### AdaBoost
Algoritmos baseados em Boosting Ensemble criam uma sequência de modelos que tentam corrigir os
erros dos modelos anteriores dentro da sequência. Uma vez criados, os modelos fazem previsões que
podem receber um peso de acordo com sua acurácia e os resultados são combinados para criar uma
previsão única final.
O AdaBoost atribui pesos às instâncias no dataset, definindo quão fácil ou difícil
 elas são para o processo de classificação, permitindo que o algoritmo tenha mais ou menos atenção
às instâncias durante o processo de construção dos modelos.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
```

```
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Definindo o número de trees
num_trees = 30
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = AdaBoostClassifier(n_estimators = num_trees, random_state = seed)
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
### Gradient Boosting
Também chamado Stochastic Gradient Boosting, é um dos métodos Ensemble mais sofisticados.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Definindo o número de trees
num_trees = 100
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando o modelo
modelo = GradientBoostingClassifier(n_estimators = num_trees, random_state = seed)
# Cross Validation
resultado = cross_val_score(modelo, X, Y, cv = kfold)
# Print do resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
### Voting Ensemble
Este é um dos métodos Ensemble mais simples. Este método cria dois ou mais modelos separados a
partir do dataset de treino. O Classificador Voting então utiliza a média das previsões de cada
sub-modelo para fazer as previsões em novos conjuntos de dados. As previsões de cada sub-modelo
podem receber pesos, através de parâmetros definidos manualmente ou através de heurística. Existem
versões mais avançadas do Voting, em que o modelo pode aprender o melhor peso a ser atribuído aos
sub-modelos. Isso é chamado Stacked Aggregation, mas ainda não está disponível no Scikit-learn.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores para o número de folds
num_folds = 10
seed = 7
# Separando os dados em folds
kfold = KFold(num_folds, True, random_state = seed)
# Criando os modelos
estimators = []
modelo1 = LogisticRegression()
estimators.append(('logistic', modelo1))
modelo2 = DecisionTreeClassifier()
```

```
estimators.append(('cart', modelo2))
modelo3 = SVC()
estimators.append(('svm', modelo3))
# Criando o modelo ensemble
ensemble = VotingClassifier(estimators)
# Cross Validation
resultado = cross_val_score(ensemble, X, Y, cv = kfold)
# Resultado
print("Acurácia: %.3f" % (resultado.mean() * 100))
## Algoritmo XGBoost - Extreme Gradient Boosting
O algoritmo XGBoost é uma extensão do GBM (Gradient Boosting Method) que permite trabalhar com
multithreading em uma única máquina e processamento paralelo em um cluster de vários servidores. A
principal vantagem do XGBoost sobre o GBM é sua capacidade de gerenciar dados esparsos. O XGBoost
automaticamente aceita dados esparsos como input sem armazenar zeros na memória.
Principais vantagensdo XGBoost:
1- Aceita dados esparsos (o que permite trabalhar com matrizes esparsas), sem a necessidade de
conversão para matrizes densas.
2- Constrói uma árvore de aprendizagem utilizando um moderno método de split (chamado quatile
sketch), o que resulta em tempo de processamento muito menor que métodos tradicionais.
3- Permite computação paralela em uma única máquina (através do uso de multithreading) e
processamento paralelo em máquinas distribuídas em cluster.
Basicamente o XGBoost utiliza os mesmos parâmetros do GBM e permite tratamento avançado de dados
missing.
O XGBoost é muito utilizado por Cientistas de Dados que vencem competições no Kaggle. Repositório
no Github: https://github.com/dmlc/XGBoost
### Instalar XGBoost a partir do PyPi
!pip install xgboost
# >
!pip install xgboost
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
# Carregando os dados
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dados = read_csv(arquivo, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo o tamanho dos dados de treino e de teste
teste_size = 0.33
seed = 7
# Criando o dataset de treino e de teste
X_treino, X_teste, y_treino, y_teste = train_test_split(X, Y, test_size = teste_size, random_state
= seed)
# Criando o modelo
modelo = XGBClassifier()
# Treinando o modelo
modelo.fit(X_treino, y_treino)
# Pront do modelo
print(modelo)
# Fazendo previsões
y_pred = modelo.predict(X_teste)
previsoes = [round(value) for value in y_pred]
# Avaliando as previsões
accuracy = accuracy_score(y_teste, previsoes)
print("Acurácia: %.2f%%" % (accuracy * 100.0))
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

O Que é Normalização e Quando Aplicar?

A normalização é uma técnica frequentemente aplicada à preparação dos dados em aprendizado de máquina. O objetivo da normalização é alterar os valores das colunas numéricas no conjunto de dados para uma escala comum, sem distorcer as diferenças nos intervalos de valores. Não precisamos aplicar normalização a todo conjunto de dados. É necessário apenas quando os recursos (variáveis) tiverem intervalos diferentes.

Por exemplo, considere o conjunto de dados contendo dois recursos, idade (x1) e receita (x2). Onde a faixa etária varia de 0 a 100 anos, enquanto a renda varia de 0 a 20.000 ou mais. A renda é cerca de 1.000 vezes maior do que a idade e com uma variação de valores muito maior. Então, esses dois recursos estão em intervalos muito diferentes. Quando fazemos análises adicionais, como regressão linear multivariada, por exemplo, a renda atribuída influenciará muito mais o resultado devido ao seu valor maior. E isso causa problemas durante o treinamento do algoritmo.

A normalização também é chamada simplesmente de Scaler Min-Max e basicamente reduz o intervalo dos dados de forma que o intervalo seja fixo entre 0 e 1 (ou -1 a 1, se houver valores negativos). Funciona melhor para casos em que a padronização (que veremos no próximo item de aprendizagem) pode não funcionar tão bem. Se a distribuição não for gaussiana ou o desvio padrão for muito pequeno, o Scaler Min-Max funciona melhor. Aqui a fórmula que define a normalização:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Quando a Normalização é Importante?

A normalização é principalmente necessária no caso de algoritmos que usam medidas de distância como clustering, sistemas de recomendação que usam semelhança de cosseno, etc. Isto é feito de forma que uma variável que está em uma escala maior não afeta o resultado apenas porque está em uma escala maior.

Abaixo listamos alguns algoritmos de Machine Learning que requerem a normalização dos dados:

1. KNN com medida de distância euclidiana se quiser que todos os recursos contribuam igualmente no modelo.
2. Regressão Logística, SVM, Perceptrons, Redes Neurais.
3. K-Means
4. Análise discriminante linear, análise de componentes principais, análise de componentes principais do kernel.

Classificadores baseados em modelo gráfico, como Fisher LDA ou Naive Bayes, bem como Árvores de Decisão e métodos baseados em árvore, como Random Forest, são invariantes ao dimensionamento de recursos, mas ainda assim pode ser uma boa ideia redimensionar os dados.

A normalização eliminará a capacidade de interpretação do modelo e, portanto, dependerá, em última instância, da necessidade do negócio

O Que é Padronização e Quando Aplicar?

Padronização (ou normalização do escore Z ou em inglês Standardization ou ainda Standard Scaler) é o processo de redimensionamento dos recursos (variáveis) para que eles tenham as propriedades de uma distribuição normal com $\mu = 0$ e $\sigma = 1$, onde μ é a média e σ é o desvio padrão da média; as pontuações padrão (também chamadas de escores z) das amostras são calculadas da seguinte forma:

$$z = \frac{x - \mu}{\sigma}$$

É amplamente utilizado em SVMs, regressão logística e redes neurais.

Normalização x Padronização

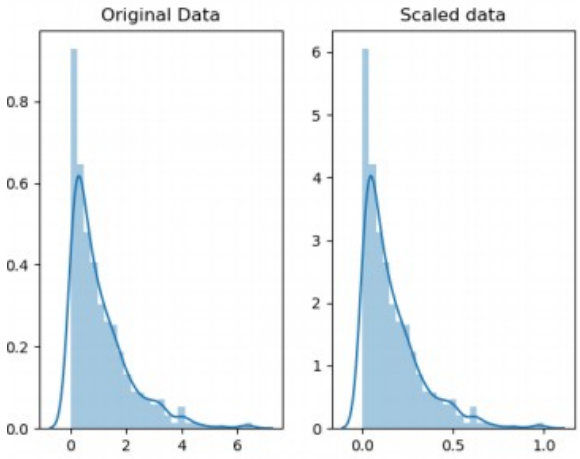
Conforme vimos nas aulas anteriores, a Normalização transforma os dados em um intervalo, digamos entre 0 e 1 ou 1 e 10, de forma que os números estejam na mesma escala. Por exemplo, podemos converter os dados de centímetros para metros para que tenhamos todos na mesma escala. A Normalização pode ser formulada como:

x <- (x - min (x)) / (max (x) - min (x))

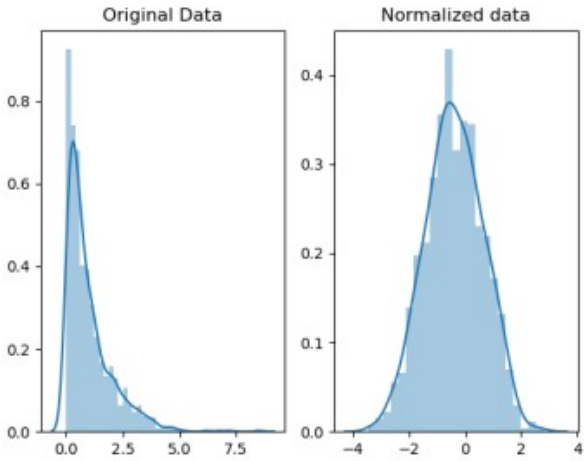
A Padronização significa transformar os dados de tal forma que eles tenham média zero e desvio padrão igual a 1. Portanto, aqui temos os dados em escala de forma padronizada, de modo que a distribuição seja aproximadamente uma distribuição normal, sendo representado da seguinte forma:

x <- (x - mean (x)) / sd (x)

Ambas as técnicas têm suas desvantagens. Se você tiver valores outliers em seu conjunto de dados, a Normalização dos dados certamente aumentará os dados "normais" para um intervalo muito pequeno. E, geralmente, a maioria dos conjuntos de dados tem outliers. Ao usar a Padronização, seus novos dados não são limitados (ao contrário da Normalização). Portanto, a Normalização é geralmente evitada quando o conjunto de dados tem outliers (desde que inclua o valor máximo). Nesses casos, preferimos a Padronização. Os gráficos abaixo resumem as diferenças quando aplicamos Normalização e Padronização:



Observe no eixo x do gráfico acima como a escala dos dados é diferentes, embora a distribuição dos dados seja a mesma. Isso é o que chamamos de Normalização.



Observe no eixo x do gráfico acima como a distribuição dos dados agora segue uma distribuição normal depois que aplicamos a Padronização. Uma distribuição normal é caracterizada por média 0 e desvio padrão 1.

Algumas considerações importantes:

1. A Normalização torna o treinamento menos sensível à escala de recursos, para que possamos resolver melhor os coeficientes.
2. O uso de um método de Normalização melhorará a análise de múltiplos modelos.
3. A Normalização assegurará que um problema de convergência não tenha uma variância massiva, tornando a otimização viável.
4. A Padronização tende a tornar o processo de treinamento bem melhor, porque a condição numérica dos problemas de otimização é melhorada

Métricas Para Modelos de Regressão / Principais Métricas Para Modelos de Regressão – MSE, MAE, R2 / Algoritmos de Regressão – Regressão Linear, Ridge, Lasso, ElasticNET, KNN, CART, SVM / Seleção, Avaliação e Otimização do Modelo:

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 6</font>
# Machine Learning em Python - Parte 2 - Regressão
# >
from IPython.display import Image
Image(url = 'images/processo.png')
# >
import sklearn as sl
import warnings
warnings.filterwarnings("ignore")
sl.__version__
## Definição do Problema de Negócio
Vamos criar um modelo preditivo que seja capaz de prever o preço de casas com base em uma série de
variáveis (características) sobre diversas casas em um bairro de Boston, cidade dos EUA.
Dataset: https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html
## Avaliando a Performance
https://scikit-learn.org/stable/modules/model_evaluation.html
As métricas que você escolhe para avaliar a performance do modelo vão influenciar a forma como a
```

```
performance é medida e comparada com modelos criados com outros algoritmos.
### Métricas para Algoritmos de Regressão
Métricas Para Avaliar Modelos de Regressão
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)
- R Squared (R²)
- Adjusted R Squared (R²)
- Mean Square Percentage Error (MSPE)
- Mean Absolute Percentage Error (MAPE)
- Root Mean Squared Logarithmic Error (RMSLE)
# >
from IPython.display import Image
Image(url = 'images/mse.png')
# >
from IPython.display import Image
Image(url = 'images/rmse.png')
# >
from IPython.display import Image
Image(url = 'images/mae.png')
# >
from IPython.display import Image
Image(url = 'images/r2.png')
Como vamos agora estudar as métricas para regressão, usaremos outro dataset, o Boston Houses.
#### MSE
É talvez a métrica mais simples e comum para a avaliação de regressão, mas também provavelmente a
menos útil. O MSE basicamente mede o erro quadrado médio de nossas previsões. Para cada ponto,
calcula a diferença quadrada entre as previsões e o valor real da variável alvo e, em seguida,
calcula a média desses valores.
Quanto maior esse valor, pior é o modelo. Esse valor nunca será negativo, já que estamos elevando
ao quadrado os erros individuais de previsão, mas seria zero para um modelo perfeito.
# >
# MSE - Mean Squared Error
# Similar ao MAE, fornece a magnitude do erro do modelo.
# Quanto maior, pior é o modelo!
# Ao extrairmos a raiz quadrada do MSE convertemos as unidades de volta ao original,
# o que pode ser útil para descrição e apresentação. Isso é chamado RMSE (Root Mean Squared Error)
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = LinearRegression()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
#### MAE
# >
# MAE
# Mean Absolute Error
# É a soma da diferença absoluta entre previsões e valores reais.
# Fornece uma ideia de quão erradas estão nossas previsões.
# Valor igual a 0 indica que não há erro, sendo a previsão perfeita.
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
```

```
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = LinearRegression()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mae = mean_absolute_error(Y_test, Y_pred)
print("O MAE do modelo é:", mae)
### R^2
# >
# R^2
# Essa métrica fornece uma indicação do nível de precisão das previsões em relação aos valores
observados.
# Também chamado de coeficiente de determinação.
# Valores entre 0 e 1, sendo 0 o valor ideal.
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = LinearRegression()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
r2 = r2_score(Y_test, Y_pred)
print("O R2 do modelo é:", r2)
# Algoritmos de Regressão
## Regressão Linear
Assume que os dados estão em Distribuição Normal e também assume que as variáveis são relevantes
para a construção do modelo e que não sejam colineares, ou seja, variáveis com alta correlação
(cabe a você, Cientista de Dados, entregar ao algoritmo as variáveis realmente relevantes).
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = LinearRegression()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
## Ridge Regression
```

```
Extensão para a regressão linear onde a loss function é modificada para minimizar a complexidade do
modelo.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = Ridge()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
## Lasso Regression
Lasso (Least Absolute Shrinkage and Selection Operator) Regression é uma modificação da regressão
linear e assim como a Ridge Regression, a loss function é modificada para minimizar a complexidade
do modelo.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Lasso
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = Lasso()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
## ElasticNet Regression
ElasticNet é uma forma de regularização da regressão que combina as propriedades da regressão Ridge
e LASSO. O objetivo é minimizar a complexidade do modelo, penalizando o modelo usando a soma dos
quadrados dos coeficientes.
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import ElasticNet
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X,
```

```
 Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = ElasticNet()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
## KNN
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = KNeighborsRegressor()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
## CART
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = DecisionTreeRegressor()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
## SVM
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
```

```
Y = array[:,13]
# Divide os dados em treino e teste
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 5)
# Criando o modelo
modelo = SVR()
# Treinando o modelo
modelo.fit(X_train, Y_train)
# Fazendo previsões
Y_pred = modelo.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
```

Otimização do Modelo - Ajuste de Parâmetros

Todos os algoritmos de Machine Learning são parametrizados, o que significa que você pode ajustar a performance do seu modelo preditivo, através do tuning (ajuste fino) dos parâmetros. Seu trabalho é encontrar a melhor combinação entre os parâmetros em cada algoritmo de Machine Learning. Esse processo também é chamado de Otimização Hyperparâmetro. O scikit-learn oferece dois métodos para otimização automática dos parâmetros: Grid Search Parameter Tuning e Random Search Parameter Tuning.

Grid Search Parameter Tuning

Este método realiza metodicamente combinações entre todos os parâmetros do algoritmo, criando um grid. Vamos experimentar este método utilizando o algoritmo de Regressão Ridge. No exemplo abaixo veremos que o valor 1 para o parâmetro alpha atingiu a melhor performance.

```
# >
# Import dos módulos
from pandas import read_csv
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores que serão testados
valores_alphas = np.array([1,0.1,0.01,0.001,0.0001,0])
valores_grid = dict(alpha = valores_alphas)
# Criando o modelo
modelo = Ridge()
# Criando o grid
grid = GridSearchCV(estimator = modelo, param_grid = valores_grid)
grid.fit(X, Y)
# Print do resultado
print("Melhores Parâmetros do Modelo:\n", grid.best_estimator_)
```

Random Search Parameter Tuning

Este método gera amostras dos parâmetros dos algoritmos a partir de uma distribuição randômica uniforme para um número fixo de interações. Um modelo é construído e testado para cada combinação de parâmetros. Neste exemplo veremos que o valor muito próximo de 1 para o parâmetro alpha é o que vai apresentar os melhores resultados.

```
# >
# Import dos módulos
from pandas import read_csv
import numpy as np
from scipy.stats import uniform
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:8]
Y = array[:,8]
# Definindo os valores que serão testados
valores_grid = {'alpha': uniform()}
seed = 7
# Criando o modelo
modelo = Ridge()
iterations = 100
rsearch = RandomizedSearchCV(estimator = modelo,
                             param_distributions = valores_grid,
                             n_iter = iterations,
                             random_state = seed)
```

```
rsearch.fit(X, Y)
# Print do resultado
print("Melhores Parâmetros do Modelo:\n", rsearch.best_estimator_)
# Salvando o resultado do seu trabalho
# >
# Import dos módulos
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge
import pickle
# Carregando os dados
arquivo = 'data/boston-houses.csv'
colunas = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO','B',
'LSTAT', 'MEDV']
dados = read_csv(arquivo, delim_whitespace = True, names = colunas)
array = dados.values
# Separando o array em componentes de input e output
X = array[:,0:13]
Y = array[:,13]
# Definindo os valores para o número de folds
teste_size = 0.35
seed = 7
# Criando o dataset de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = teste_size, random_state
= seed)
# Criando o modelo
modelo = Ridge()
# Treinando o modelo
modelo.fit(X_treino, Y_treino)
# Salvando o modelo
arquivo = 'modelos/modelo_regressor_final.sav'
pickle.dump(modelo, open(arquivo, 'wb'))
print("Modelo salvo!")
# Carregando o arquivo
modelo_regressor_final = pickle.load(open(arquivo, 'rb'))
print("Modelo carregado!")
# Print do resultado
# Fazendo previsões
Y_pred = modelo_regressor_final.predict(X_test)
# Resultado
mse = mean_squared_error(Y_test, Y_pred)
print("O MSE do modelo é:", mse)
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Quiz:

Aprendizagem Supervisionada – Esses algoritmos consistem em variáveis target / variáveis de saída (ou variáveis dependentes) que são previstas por um grupo de variáveis preditoras (variáveis independentes). Usando este grupo de variáveis, nós geramos uma função que mapeia entradas para saídas desejáveis. O processo de treinamento continua até o modelo atingir um determinado nível de precisão desejável nos dados de teste.

Aprendizagem Não-Supervisionada – Nestes algoritmos, nós não temos variáveis target, ou seja, variáveis de saída para serem estimadas. São feitos agrupamentos na população de dados em diferentes grupos, amplamente utilizados para segmentar as observações com características específicas.

A fase de pré-processamento não tem importância no processo de Machine Learning e pode ser facilmente ignorada! (FALSO)

A biblioteca scikit-learn provê algoritmos de Machine Learning para classificação, regressão, redução de dimensionalidade e clustering.

O scikit-learn possui licença BSD, o que significa que poder usado livremente para fins comerciais sem restrições.

5.7. Processamento Big Data com Apache Spark

Introdução

Como Vamos Estudar o Spark?

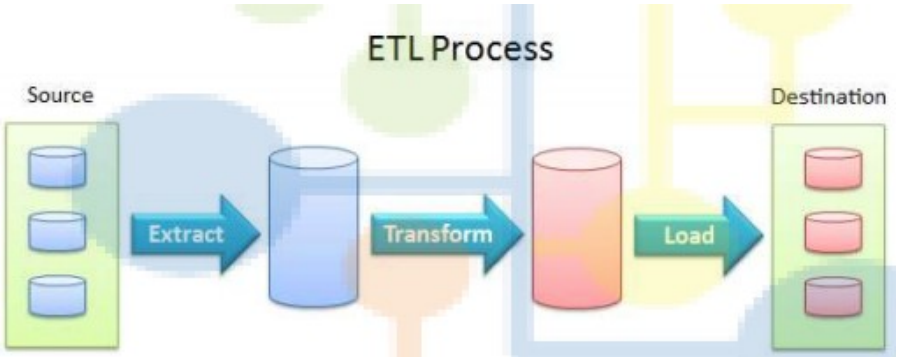
- Capítulo 7 - Arquitetura do Spark, Transformações, Ações, PySpark
- Capítulo 8 - Spark SQL
- Capítulo 9 - Spark Streaming e Análise de Dados em Tempo Real
- Capítulo 10 - Machine Learning em Streaming de Dados com Spark MLlib

Solução – Desafio – Pipeline com PCA e Regressão Logística

```
# Modelo preditivo de classificação para prever o valor de uma variável binária (true ou false) a
partir de dados numéricos

# Import dos módulos
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore")

# Dados de treino
n_train = 10
np.random.seed(0)
df_treino = pd.DataFrame({"var1": np.random.random(n_train), \
                          "var2": np.random.random(n_train), \
                          "var3": np.random.random(n_train), \
                          "var4": np.random.randint(0,2,n_train).astype(bool),\
                          "target": np.random.randint(0,2,n_train).astype(bool)})

# Dados de teste
n_test = 3
np.random.seed(1)
df_teste = pd.DataFrame({"var1": np.random.random(n_test), \
                         "var2": np.random.random(n_test), \
                         "var3": np.random.random(n_test), \
                         "var4": np.random.randint(0,2,n_test).astype(bool),\
                         "target": np.random.randint(0,2,n_test).astype(bool)})

# Reduzindo a dimensionalidade para 3 componentes
pca = PCA(n_components = 3)

# Aplique o PCA aos datasets
newdf_treino = pca.fit_transform(df_treino.drop("target", axis = 1))
newdf_teste = pca.transform(df_teste.drop("target", axis = 1))

# Crie dataframes do pandas com o resultado do item anterior
features_treino = pd.DataFrame(newdf_treino)
features_teste = pd.DataFrame(newdf_teste)

# Crie um modelo de regressão logística
regr = LogisticRegression()

# Usando o recurso de pipeline do scikit-learn para encadear 2 algoritmos em um mesmo modelo, use o
resultado do PCA como entrada para Regressão Logística
pipe = Pipeline([('pca', pca), ('logistic', regr)])
pipe.fit(features_treino, df_treino["target"])

# Faça previsões com o modelo treinado
predictions = pipe.predict(features_teste)

# Imprimindo as previsões
print("\nPrevisões do modelo:")
print(predictions)
```

Apache Spark e Big Data

Para tratamento de dados não estruturados, utiliza-se processos de ETL (Extract, Transform, Load)



Como armazenar e processar todos esses dados, se o volume aumenta de forma exponencial?

Clusters são conjuntos de computadores (servidores) conectados (nodes), que executam como se fossem um único sistema.

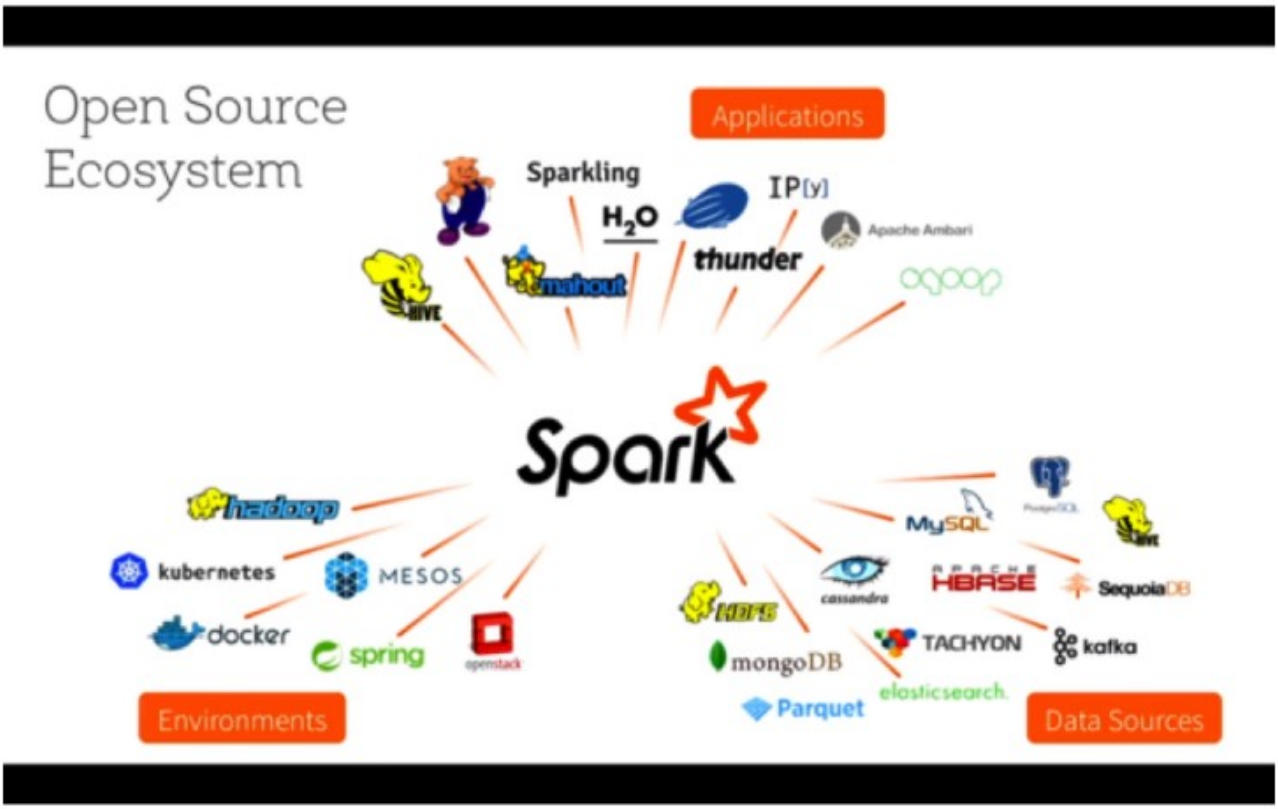
Apache Spark é um sistema de análise de dados distribuídos e altamente escalável, que permite processamento em memória e desenvolvimento de aplicações em Java, Scala, Python e R.

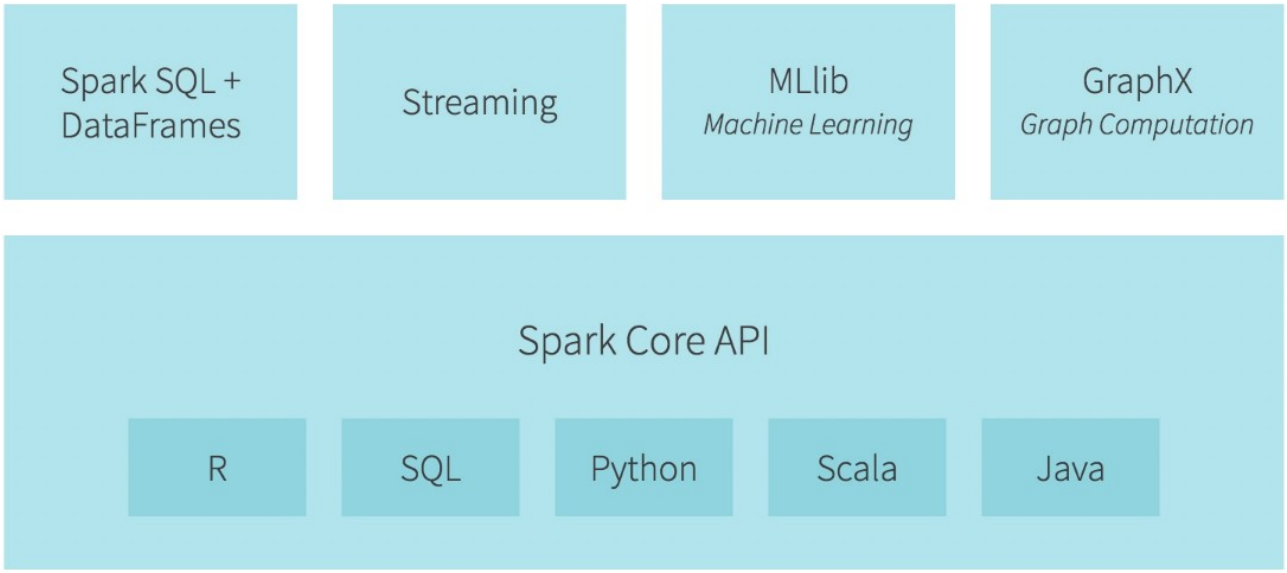
Apache Spark é um framework open-source para processamento de Big Data construído para ser veloz, fácil de usar e para análises sofisticadas.

Apache Spark é uma ferramenta de análise de Big Data, escalável e eficiente.

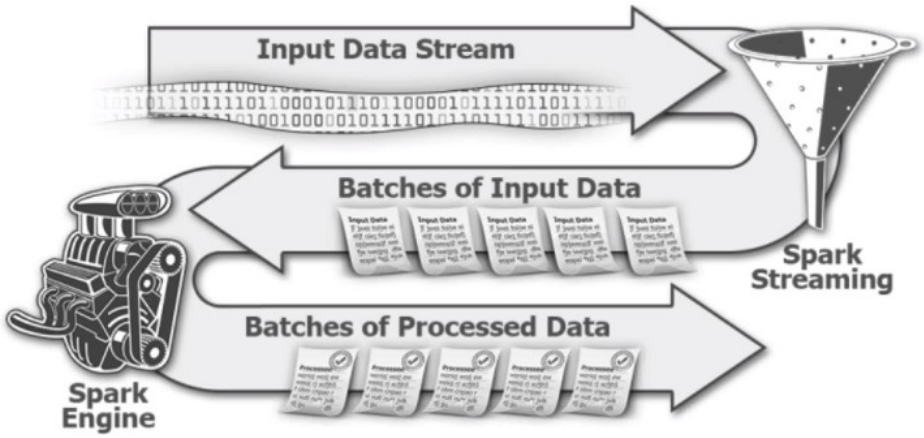
O Apache Spark é desenvolvido em liguagem Scala e é executado numa máquina virtual Java.

Ecossistema e Componentes do Apache Spark





Spark Streaming:



Streaming: fluxo contínuo de dados.

Usado para coletar e processar dados em tempo real.

Spark Mllib (Spark Machine Learning Library):

Consistem em algoritmos de aprendizagem de máquina, algumas ferramentas estatísticas, bem como algoritmos de regressão, classificação, filtros colaborativos, clusterização (aprendizagem não supervisionada), redução de dimensionalidade, extração de atributos, otimização, etc.

Spark SQL:

Módulo do Apache Spark para processar dados estruturados.

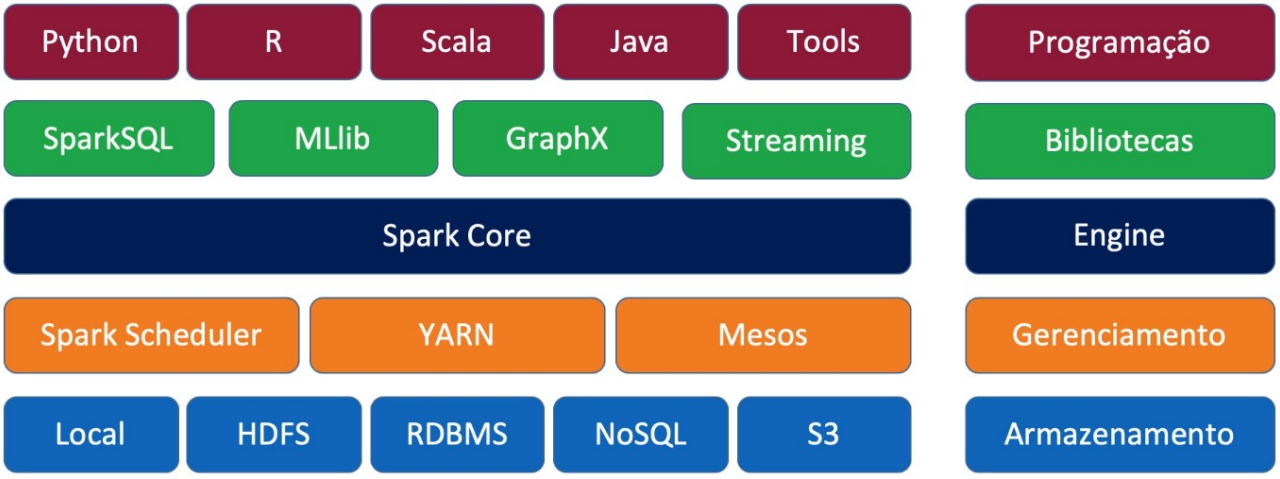
Spark GraphX:

Permite que o Spark realize, de forma rápida e em memória, o processamento grafos.

Databricks:

É uma das principais soluções em nuvem para o Apache Spark.

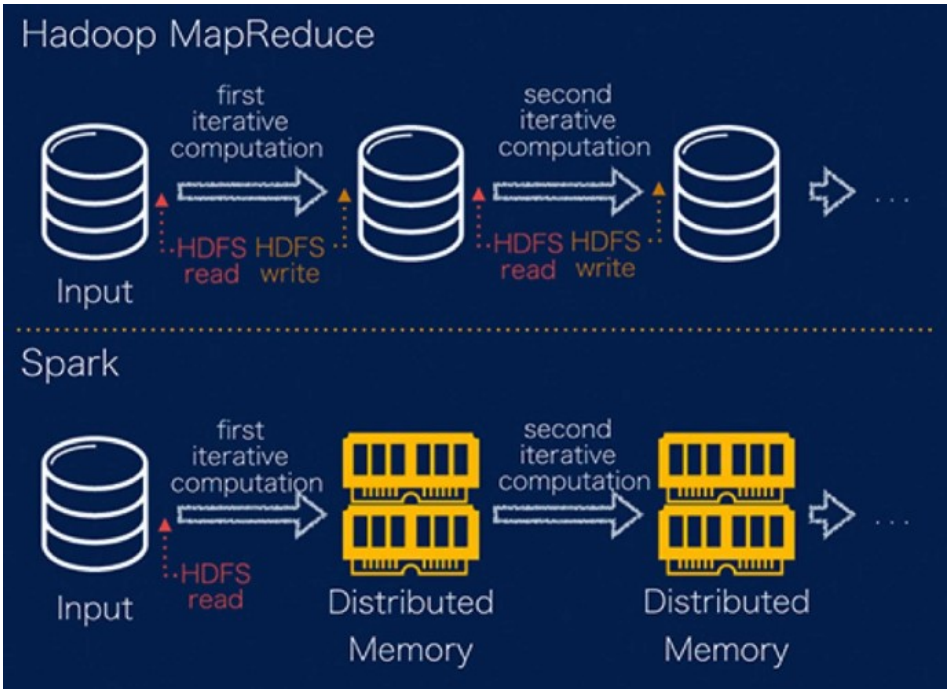
Resumindo o ecossistema Apache Spark:



Quando Devemos Usar o Spark?

- Integração de Dados (trazer dados estruturados para o hadoop) e ETL
- Análises Interativas (executar queries dentro do terminal)
- Computação em Batch de Alta Performance
- Análises Avançadas de Machine Learning
- Processamento de Dados em Tempo Real

Principais Características do Apache Spark



- Spark realiza operações de MapReduce (mapear e reduzir dados)
- Spark pode utilizar o HDFS (armazenamento distribuído)
- Spark permite construir um workflow de Analytics (pipeline de análise de dados)
- Spark utiliza a memória do computador de forma diferente e eficiente
- Spark é veloz
- Spark é flexível (na prática, tudo é programação)
- Spark é gratuito (não tem custo de licença)

Verificando a Instalação do Spark

Para execução do Spark é necessário ter o java, o python, além do próprio spark.

Para executar o Spark, abra o prompt de comando e entre no diretório onde se encontram os notebooks do jupyter. Em seguida, execute o comando pyspark (para encerrar, tecle ctrl+c).

Introdução ao PySpark /
Executando Aplicação PySpark /
Operação de MapReduce com PySpark

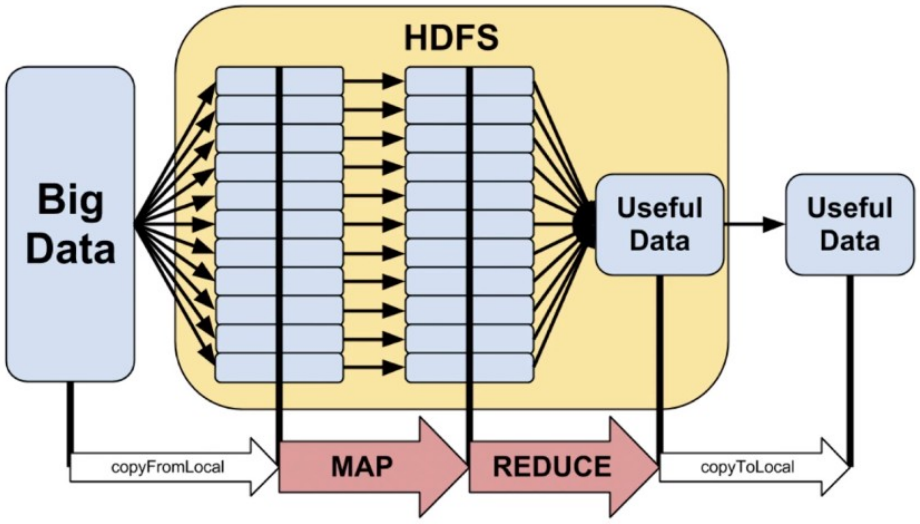
```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 7</font>
### ********** Atenção: **********
Utilize Java JDK 11 e Apache Spark 2.4.2
*Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório
metastore_db no mesmo diretório onde está este Jupyter notebook*
## Introdução ao PySpark
# >
import sys
print(sys.version)
# >
print(sc) # SparkContext
# >
print(sc.version)
# >
# Testando o Spark e criando uma RDD
lst = [25, 90, 81, 37, 776, 3320]
testData = sc.parallelize(lst)
# >
?sc.parallelize # distribui uma coleção local Python (uma lista) e converte em uma RDD, para gravar
num cluster de computadores.
# >
type(testData)
# >
testData.count()
# >
testData.collect()
## Executando Aplicação PySpark
RDD's são coleções distribuídas de itens. RDD's podem ser criadas a partir do Hadoop (arquivos no
HDFS), através da transformação de outras RDD's, a partir de bancos de dados (relacionais e não-
relacionais) ou a partir de arquivos locais.
# >
# Criando uma RDD a partir de um arquivo csv
sentimentoRDD = sc.textFile("data/sentimentos.csv")
# >
type(sentimentoRDD) # pyspark.rdd.RDD
# >
# Ação - Contando o número de registros
sentimentoRDD.count()
# >
# Listando os 5 primeiros registros
sentimentoRDD.take(5)
# >
# Transformando os dados - transformação para letras maiúsculas
transfRDD = sentimentoRDD.map(lambda x : x.upper())
transfRDD.take(5)
# >
sentimentoRDD.take(5) # uma RDD é um objeto imutável
# >
arquivo = sc.textFile("data/sentimentos.csv")
# >
type(arquivo)
# >
arquivo.count() # contagem de elementos da RDD
# >
arquivo.first() # primeiro elemento da RDD
# >
linhasComSol = arquivo.filter(lambda line: "Sol" in line) # filtro
# >
type(linhasComSol) # pyspark.rdd.PipelinedRDD
# >
linhasComSol.count()
Primeiro a função map() determina o comprimento de cada linha do arquivo, criando uma RDD. A função
reduce() é chamada para encontrar a linha com maior número de caracteres. O argumento para as
funções map() e reduce() são funções anônimas criadas com lambda (da linguagem Python).
# >
arquivo.map(lambda line: len(line.split())).reduce(lambda a, b: a if (a > b) else b)
Esta linha pode ser reescrita da seguinte forma:
# >
def max(a, b):
    if a > b:
        return a
```

```
    else:
        return b
arquivo.map(lambda line: len(line.split())).reduce(max)
## Operação de MapReduce
As operações de MapReduce foram popularizadas pelo Hadoop e podem ser feitas com Spark até 100x
mais rápido.
# >
contaPalavras = arquivo.flatMap(lambda line: line.split()).map(lambda palavra: (palavra,
1)).reduceByKey(lambda a, b: a+b)
# >
contaPalavras.collect()
Acesse o monitoramento do Spark em: http://localhost:4040/
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Hadoop MapReduce x Apache Spark

Hadoop MapReduce e Apache Spark são os dois frameworks mais populares para computação em cluster e análise de dados de larga escala (Big Data).

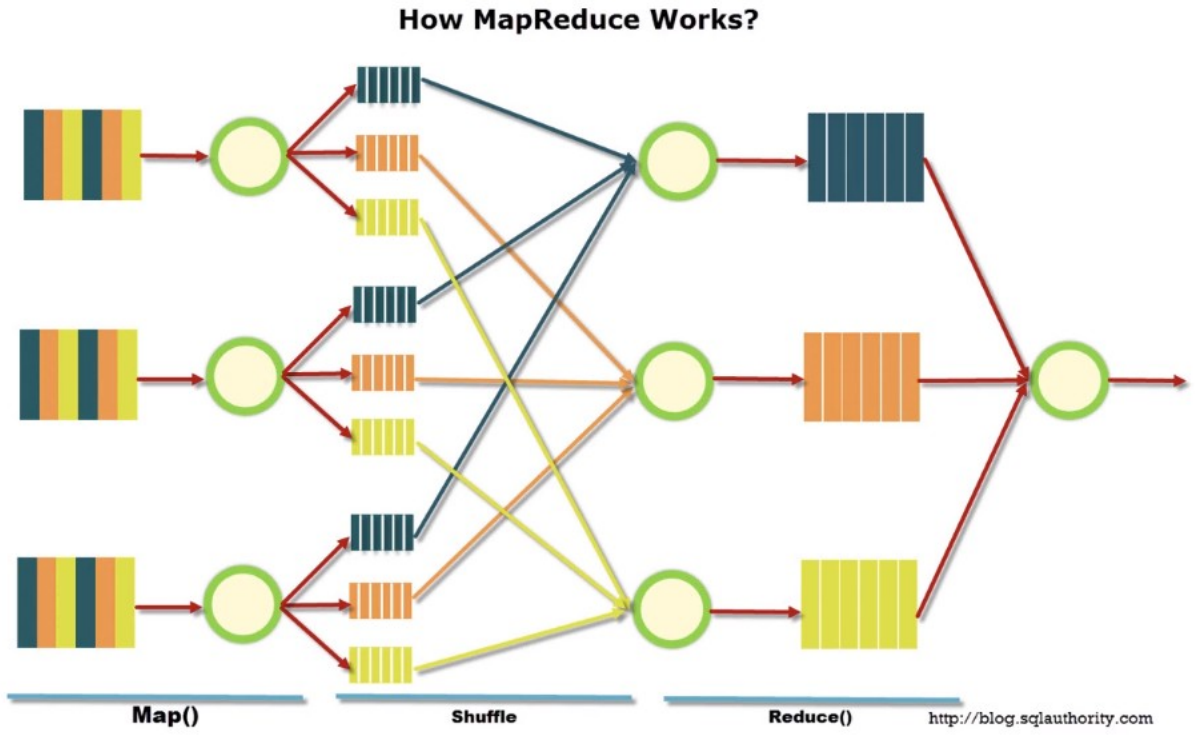
Estes dois frameworks escondem a complexidade existente no tratamento de dados com relação a paralelismo entre tarefas e tolerância a falha por meio da exposição de uma simples API com informações para os usuários.

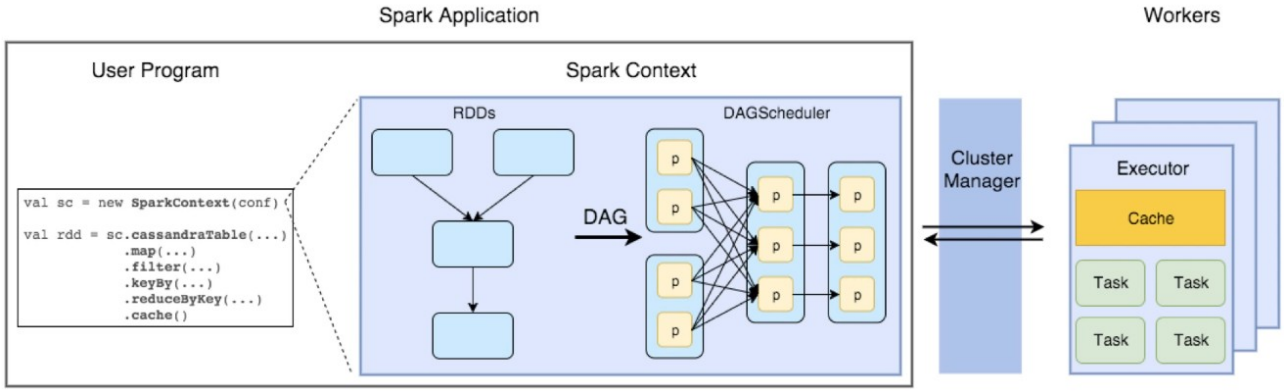


O Spark realiza o processamento distribuído, de forma similar ao Hadoop MapReduce, porém com muito mais velocidade.

O Spark não possui sistema de armazenamento, podendo usar o HDFS como fonte/destino de dados.

O Processo de MapReduce no Apache Spark





RDD's = Resilient Distributed Datasets

Spark Context é a área de memória onde será executado o Spark.

- O Spark suporta mais do que apenas as funções de Map e Reduce.
- Hadoop MapReduce grava os resultados intermediários em disco, enquanto o Spark grava os resultados intermediários em memória, o que é muito mais rápido.
- O Spark fornece APIs concisas e consistentes em Scala, Java e Python (e mais recentemente em R).
- O Spark oferece shell interativo para Scala, Python e R.
- O Spark pode utilizar o HDFS como uma de suas fontes/destinos de dados.

O Cientista de Dados é responsável por definir as regras de manipulação e análise de dados.

O Engenheiro de Dados é responsável por garantir o processamento distribuído, cuidando do pipeline, fontes de dados e destino, bem como pela segurança dos dados.

O Spark e o Hadoop MapReduce tem uma característica principal em comum: são responsáveis por gerenciar o processamento distribuído. Porém o Spark faz isso de forma muito mais rápida e eficiente que o Hadoop MapReduce.

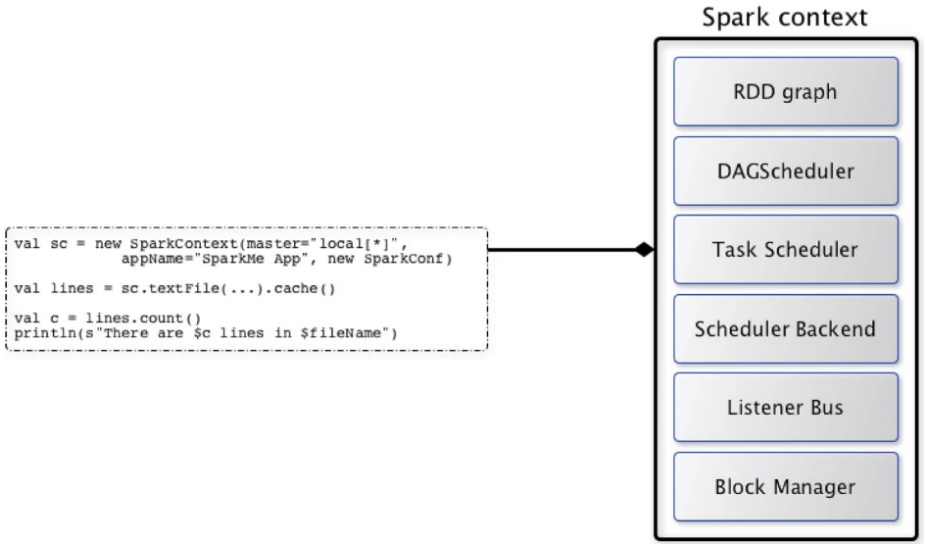
Profissionais Que Trabalham com Apache Spark

Exstem basicamente 3 perfis de profissionais que vão trabalhar com Spark:

- Cientistas de Dados
- Engenheiros de Dados
- Administradores

Anatomia de Uma Aplicação Spark

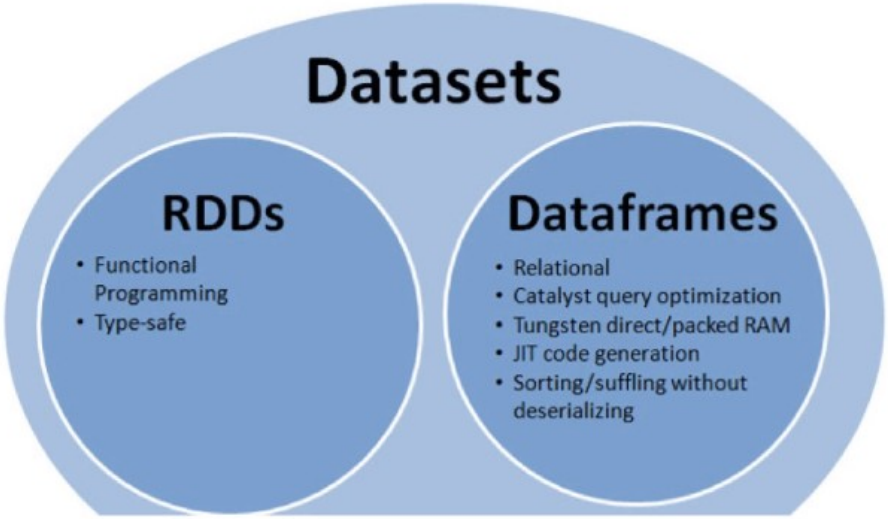
Cada aplicação Spark inicia uma instância de um Spark Context.



Sem um Spark Context, nada pode ser feito no Spark. Cada aplicação Spark é uma instância de um Spark Context.

O Spark Context é basicamente uma espécie de cliente que estabelece a conexão com o ambiente de execução do Spark e age como o processo principal da sua aplicação.

Com o Spark Context criado, podemos então definir os nossos RDD’s e Dataframes, que são os objetos que vão armazenar os dados para o processamento pelo Spark.



E como criamos um Spark Context?

1. Shell – área de trabalho via linha de comando

2. Spark Context – conexão ao ambiente Spark criado quando iniciamos o pyspark

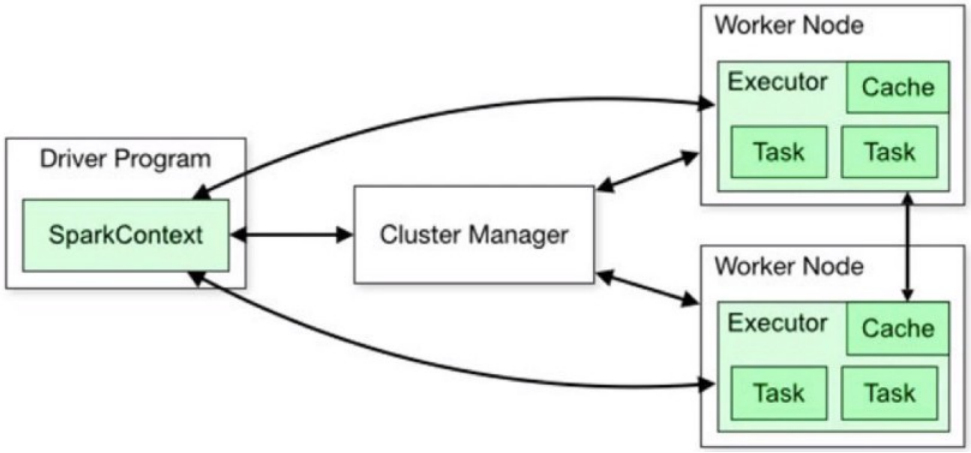
Ou seja, uma vez que iniciamos o PySpark, é criado um Spark Context, que nos permite criar RDD's e Dataframes e realizar transformações e ações. Cada operação Spark gera um job que então é executado ou agendado para ser executado ao longo do cluster de computadores ou localmente em nossa máquina.

Quando utilizamos o PySpark e o Jupyter Notebook não precisamos nos preocupar com a criação do Spark Context.

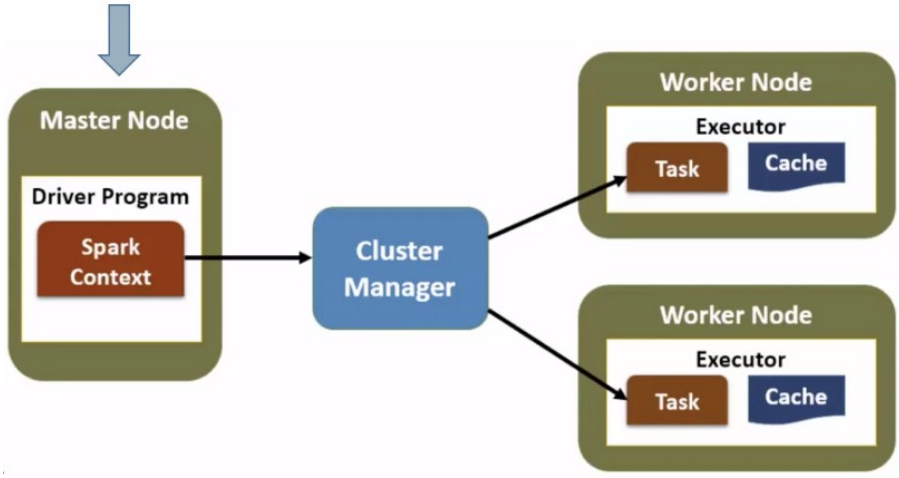
Mas quando criamos uma aplicação de análise de dados (um arquivo .py por exemplo) e utilizamos o spark-submit para iniciar nossa aplicação, precisamos explicitamente criar um Spark Context.

Arquitetura Spark

Arquitetura Spark Master/Worker:



Arquitetura Spark:



- Spark Context: ponto de partida.
  - Deve ser selecionado uma API suportada pelo Apache Spark (Scala, Java, Python ou R) e criado um Spark Context.
  - Roda no computador mestre / gestor de todo o cluster spark.
- Cluster Manager: gerenciamento de recursos de cluster para execução de um JOB.
- Worker Node: roda o processo executor (quem realiza a tarefa).
  - Worker = host = node = computador.
- Executor: é uma JVM (Java Virtual Machine)

Spark Modes

- Modo Batch: Um programa é agendado para executar em intervalos específicos (através do scheduler).
- Modo Iterativo: Utiliza o shell para executar comandos no cluster. O shell age como um driver program e provê um SparkContext.
- Modo Streaming: Um programa que executa continuamente para processar os dados à medida que eles chegam, em tempo real.

Deploy Mode e Fontes de Dados

Spark Mode → Modo de processamento de dados no Spark (Batch, Interativo ou Streaming).

Deploy Mode → Modo de execução do Spark. Em cada Deploy Mode podemos usar um ou mais Spark Modes.

Tipos de Deploy Mode:

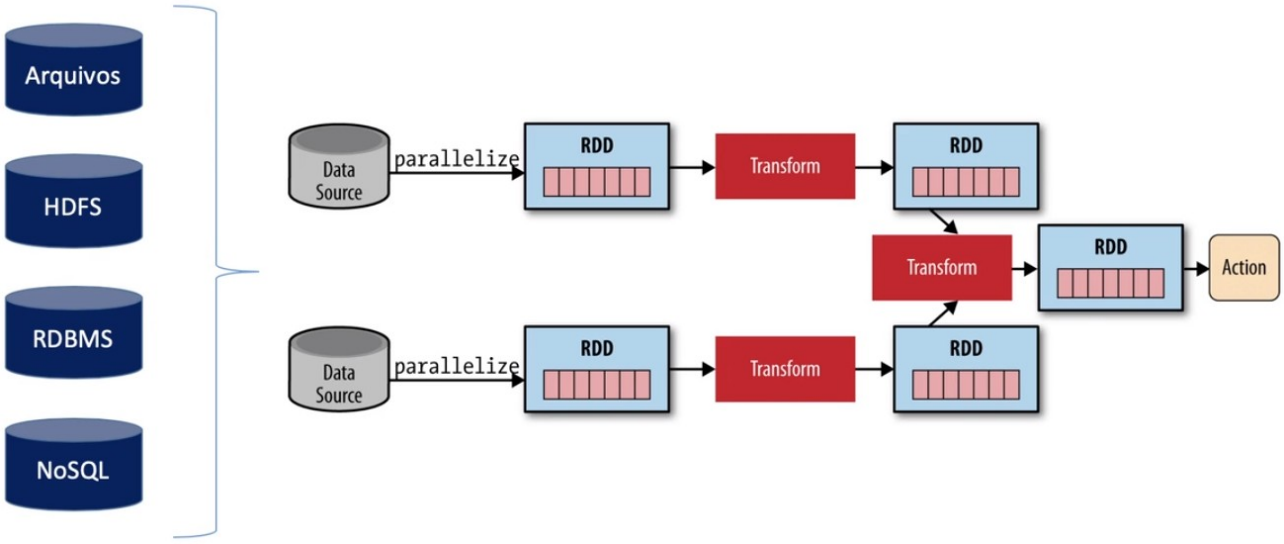
- Local (Standalone ou Cluster) – Única JVM
- Cluster em Nuvem (Databricks, Amazon EC2, IBM Bluemix) – Cluster Gerenciado

Possíveis fontes de dados para o Apache Spark:

RDDs – Resilient Distributed Datasets

RDD é uma coleção de objetos, distribuída e imutável. Cada conjunto de dados no RDD é dividido em partições lógicas, que podem ser computados em diferentes nodes do cluster.

RDD's são imutáveis!



Existem 2 formas de criar o RDD:

1. Paralelizando uma coleção existente (função sc.parallelize)
2. Referenciando um dataset externo (HDFS, RDBMS, NoSQL, S3)

As RDD’s são a essência do funcionamento do Spark.

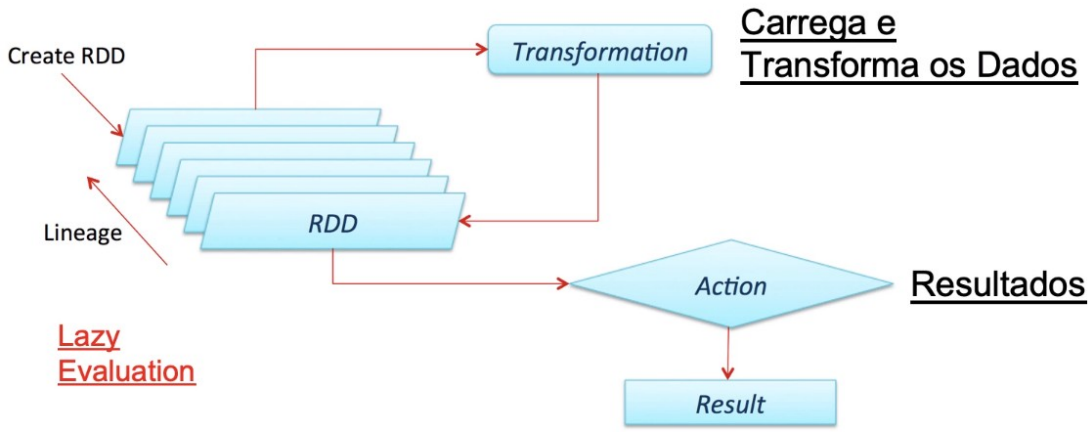
Xxx

Por padrão, os RDD’s são computados cada vez que executamos uma Ação. Entretanto, podemos “persistir” o RDD na memória (ou mesmo no disco) de modo que os dados estejam disponíveis ao longo do cluster e possam ser processados de forma muito mais rápida pelas operações de análise de dados criadas por você, Cientista de Dados.

O RDD suporta dois tipos de operações:

Transformações	Ações
◦ map(func) ◦ flatMap(func) ◦ groupByKey() ◦ reduceByKey(func) ◦ mapValues(func) ◦ sample(…) ◦ union(other) ◦ distinct() ◦ sortByKey()	◦ reduce(func) ◦ collect() ◦ count() ◦ first() ◦ take(n) ◦ saveAsTextFile(path) ◦ countByKey() ◦ foreach(func)

As “Ações” aplicam as “Transformações” nos RDD’s e retornam resultado.



- Spark é baseado em RDD’s. Criamos, transformamos e armazenamos RDD’s com Spark.
- RDD representa uma coleção de elementos de dados particionados que podem ser operados em paralelo.
- RDD’s são objetos imutáveis. Eles não podem ser alterados uma vez criados.
- RDD’s podem ser colocados em cache e permitem persistência (mesmo objeto usado entre sessões diferentes).
- Ao aplicarmos Transformações em RDD’s criamos novos RDD’s.

Portanto, as 3 características principais dos RDD’s são:

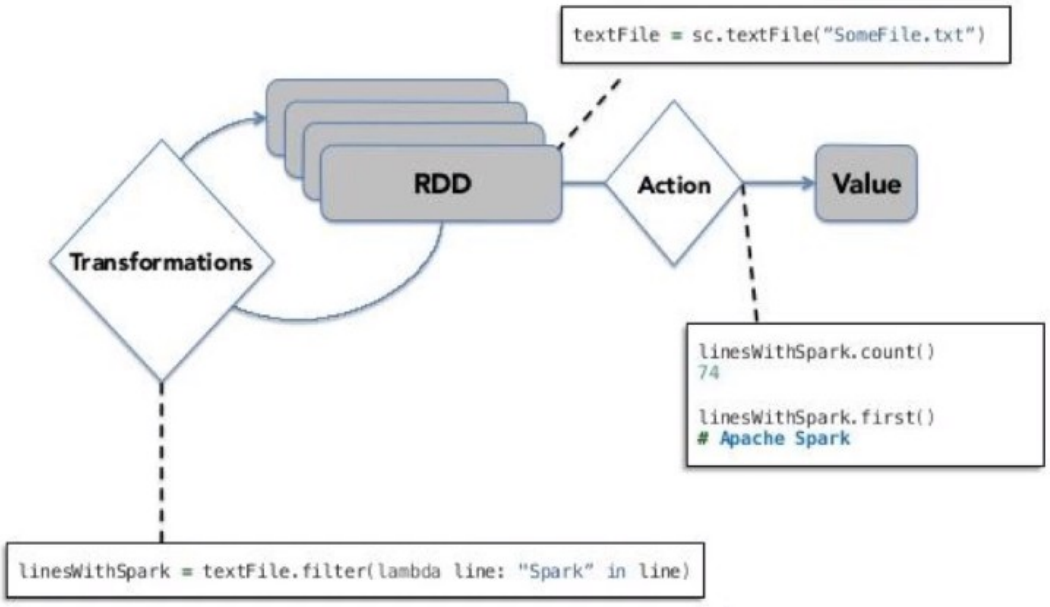
1. Imutabilidade – Importante quando se realiza processamento paralelo.
2. Particionado e Distribuído – Permite processar arquivos através de diversos computadores.

3. Armazenamento em Memória – Processamento muito mais veloz, permitindo armazenar os resultados intermediários em memória.

O Que São Transformações?

Transformações são “operações preguiçosas” (lazy operations) executadas sobre os RDD’s e que criam um ou mais RDD’s.

Dizemos que as transformações são operações lazy, porque elas não são executadas imediatamente, mas sim no momento em que as operações de ação são executadas.



Após executar operações de Transformação no RDD, o RDD resultante será diferente do RDD original e poderá ser menor (se usadas as funções filter, count, distinct, sample) ou maior (se usadas as funções flatMap, union, cartesian).

Transformações:

- Realizam as operações em um RDD e criam um novo RDD.
- Operações são feitas em um elemento por vez.
- Lazy Evaluation.
- Pode ser distribuída através de múltiplos nodes.

Algumas operações de Tranformação podem ser colocadas no que o Spark chama de Pipeline, que é um encadeamento de transformações visando aumentar a performance.

Tipos de Transformações:

Narrow	Wide
Resultado de funções como map() e filter() e os dados vem de uma única partição.	Resultado de funções como groupByKey() e reduceByKey() e os dados podem vir de diversas partições.

Principais Operações de Transformação

- Map
  - Conceito de MapReduce
  - Age sobre cada elemento e realiza a mesma operação
- flatMap
  - Funciona como a função Map, mas retorna mais elementos
- Filter
  - Filtra um RDD para retornar elementos
- Set
  - São realizadas em duas RDD's, com operações de união e interseção
- mapPartitions
  - Quando utilizamos como fontes de dados bancos de dados como Hbase ou Cassandra, temos dados armazenados com pares chave-valor. A transformação mapPartitions garante a atomicidade dos dados, evitando problemas de overhead na manipulação de dados e garantindo performance.

Lista com Todas as Operações de Transformação:

http://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

O Que São Ações?

São operações executadas sobre os RDD's que geram um resultado.

Leitura de um arquivo (RDD) > Divisão do arquivo em linhas (Transformação) > Mapeamento de cada palavra (Transformação) > Redução por agregação (Transformação) > Ordenação (Transformação) > Impressão das palavras com maior ocorrência (Ação).

Ações são operações síncronas mas podemos usar a função AsyncRDDActions() para tornar as operações assíncronas.

Podemos pensar nas ações como válvulas. Os dados estão prontos para serem processados e operações de transformação já foram definidas, mas somente quando abrirmos as válvulas, ou seja, executarmos as ações, o processamento será realmente iniciado.

Técnicas de otimização de performance (armazenam resultados intermediários em memória):

Caching	Persistence
Cache()	persistence()

Nós devemos colocar os RDD's em cache, sempre que for necessário executar duas ou mais Ações no conjunto de dados. Isso melhora a performance.

Lista com Todas as Operações de Ação:

http://spark.apache.org/docs/latest/rdd-programming-guide.html#actions

Transformações – PySpark /
Operações Set, Outer Join e Distinct /
Transformação e Limpeza /
Ações – PySpark

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 7</font>
### ********** Atenção: **********
Utilize Java JDK 11 e Apache Spark 2.4.2
*Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório
metastore_db no mesmo diretório onde está este Jupyter notebook*
Acesse http://localhost:4040 sempre que quiser acompanhar a execução dos jobs
# Transformações
# >
# Criando uma lista em Python
lista1 = [124, 901, 652, 102, 397]
# >
type(lista1)
# >
# Carregando dados de uma coleção
lstRDD = sc.parallelize(lista1)
# >
type(lstRDD) # pyspark.rdd.RDD
# >
lstRDD.collect() # recupera RDD
# >
lstRDD.count()
# >
# Carregando um arquivo e criando um RDD.
autoDataRDD = sc.textFile("data/carros.csv")
# >
type(autoDataRDD) # pyspark.rdd.RDD
# >
# Operação de Ação.
autoDataRDD.first()
# >
autoDataRDD.take(5)
# >
# Cada ação gera um novo processo de computação dos dados.
# Mas podemos persistir os dados em cache para que ele possa ser usado por outras ações, sem a
necessidade
# de nova computação.
autoDataRDD.cache() # data/carros.csv MapPartitionsRDD[3] at textFile at
NativeMethodAccessorImpl.java:0
# >
for line in autoDataRDD.collect():
    print(line)
# >
# Map() e criação de um novo RDD - Transformação - Lazy Evaluation
tsvData = autoDataRDD.map(lambda x : x.replace(",","\t"))
tsvData.take(5)
# >
autoDataRDD.first()
# >
# Filter() e criação de um novo RDD - Transformação - Lazy Evaluation
toyotaData = autoDataRDD.filter(lambda x: "toyota" in x)
# >
# Ação
toyotaData.count()
# >
# Ação
toyotaData.take(20)
# >
# Pode salvar o conjunto de dados, o RDD.
# Nesse caso, o Spark solicita os dados ao processo Master e então gera um arquivo de saída.
savedRDD = open("data/carros_v2.csv","w")
savedRDD.write("\n".join(autoDataRDD.collect()))
savedRDD.close()
## Operações Set
# >
# Set operations
palavras1 = sc.parallelize(["Big Data","Data Science","Analytics","Visualization"])
palavras2 = sc.parallelize(["Big Data","R","Python","Scala"])
# >
# União
for unions in palavras1.union(palavras2).distinct().collect():
```

```
    print(unions)
# >
# Interseção
for intersects in palavras1.intersection(palavras2).collect():
    print(intersects)
# >
rdd01 = sc.parallelize(range(1,10))
rdd02 = sc.parallelize(range(10,21))
rdd01.union(rdd02).collect()
# >
rdd01 = sc.parallelize(range(1,10))
rdd02 = sc.parallelize(range(5,15))
rdd01.intersection(rdd02).collect()
## Left/Right Outer Join
# >
names1 = sc.parallelize(("banana", "uva", "laranja")).map(lambda a: (a, 1))
names2 = sc.parallelize(("laranja", "abacaxi", "manga")).map(lambda a: (a, 1))
names1.join(names2).collect()
# >
names1.leftOuterJoin(names2).collect()
# >
names1.rightOuterJoin(names2).collect()
## Distinct
# >
# Distinct
lista1 = [124, 901, 652, 102, 397, 124, 901, 652]
lstRDD = sc.parallelize(lista1)
for numbData in lstRDD.distinct().collect():
    print(numbData)
## Transformação e Limpeza
# >
# RDD Original
autoDataRDD.collect()
# >
# Transformação e Limpeza
def LimpaRDD(autoStr) :
    # Verifica a indexação
    if isinstance(autoStr, int) :
        return autoStr
    # Separa cada índice pela vírgula (separador de colunas)
    attList = autoStr.split(",")
    # Converte o número de portas para um num
    if attList[3] == "two" :
        attList[3] = "2"
    elif attList[3] == "four":
        attList[3] = "4"
    # Convert o modelo do carro para uppercase
    attList[5] = attList[4].upper()
    return ",".join(attList)
# >
# Transformação
RDD_limpo = autoDataRDD.map(LimpaRDD)
# >
print(RDD_limpo) # PythonRDD[73] at RDD at PythonRDD.scala:53
# >
# Ação
RDD_limpo.collect()
## Ações
# >
# reduce() - soma de valores
lista2 = [124, 901, 652, 102, 397, 124, 901, 652]
lstRDD = sc.parallelize(lista2)
lstRDD.collect()
lstRDD.reduce(lambda x,y: x + y)
# >
# Encontrando a linha com menor número de caracteres
autoDataRDD.reduce(lambda x,y: x if len(x) < len(y) else y)
# >
# Criando uma função para redução
def getMPG(autoStr) :
    if isinstance(autoStr, int) :
        return autoStr
    attList = autoStr.split(",")
    if attList[9].isdigit() :
        return int(attList[9])
    else:
        return 0
# >
# Encontrando a média de MPG para todos os carros
```

```
media_mpg = round(autoDataRDD.reduce(lambda x,y : getMPG(x) + getMPG(y)) / (autoDataRDD.count() -
1), 2)
print(media_mpg)
# >
times = sc.parallelize(("Flamengo", "Vasco", "Botafogo", "Fluminense", "Palmeiras", "Bahia"))
times.takeSample(True, 3)
# >
times = sc.parallelize(("Flamengo", "Vasco", "Botafogo", "Fluminense", "Palmeiras", "Bahia",
"Bahia", "Flamengo"))
times.map(lambda k: (k,1)).countByKey().items()
# >
autoDataRDD.saveAsTextFile("data/autoDataRDD.txt")
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Mini-Projeto 1 – Analisando Dados do Uber com Spark

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 7</font>
### ********** Atenção: **********
Utilize Java JDK 11 e Apache Spark 2.4.2
*Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório
metastore_db no mesmo diretório onde está este Jupyter notebook*
# Mini-Projeto 1 - Analisando Dados do Uber com Spark
Dataset: https://github.com/fivethirtyeight/uber-tlc-foil-response
Esse conjunto de dados contém dados de mais de 4,5 milhões de captações Uber na cidade de Nova York
de abril a setembro de 2014 e 14,3 milhões de captações Uber de janeiro a junho de 2015. Dados em
nível de viagem sobre 10 outras empresas de veículos de aluguel (FHV) bem como dados agregados para
329 empresas de FHV, também estão incluídos. Todos os arquivos foram recebidos em 3 de agosto, 15
de setembro e 22 de setembro de 2015.
1- Quantos são e quais são as bases de carros do Uber (onde os carros ficam esperando passageiros)?
2- Qual o total de veículos que passaram pela base B02617?
3- Qual o total de corridas por base? Apresente de forma decrescente.
# >
from pandas import read_csv
# >
# Criando um objeto Pandas
uberFile = read_csv("data/uber.csv")
# >
type(uberFile)
# >
# Visualizando as primeiras linhas
uberFile.head(10)
# >
# Tranformando o dataframe (Pandas) em um Dataframe (Spark)
uberDF = sqlContext.createDataFrame(uberFile)
# >
type(uberDF)
# >
# Criando o RDD a partir do arquivo csv
uberRDD = sc.textFile("data/uber.csv")
# >
type(uberRDD)
# >
# Total de registros
uberRDD.count()
# >
# Primeiro registro
uberRDD.first()
# >
# Dividindo o arquivo em colunas, separadas pelo caracter ","
uberLinhas = uberRDD.map(lambda line: line.split(","))
# >
type(uberLinhas)
# >
# Número de bases de carros do Uber
uberLinhas.map(lambda linha: linha[0]).distinct().count() -1
# linha [0]: coluna de indice 0 / primeira coluna
# -1: exclui o cabeçalho
# >
# Bases de carros do Uber
```

```
uberLinhas.map(lambda linha: linha[0]).distinct().collect()
# >
# Total de veículos que passaram pela base B02617
uberLinhas.filter(lambda linha: "B02617" in linha).count()
# >
# Gravando os dados dos veículos da base B02617 em um novo RDD
b02617_RDD = uberLinhas.filter(lambda linha: "B02617" in linha)
# >
# Total de dias em que o número de corridas foi superior a 16.000
b02617_RDD.filter(lambda linha: int(linha[3]) > 16000).count()
# >
# Dias em que o total de corridas foi superior a 16.000
b02617_RDD.filter(lambda linha: int(linha[3]) > 16000).collect()
# >
# Criando um novo RDD
uberRDD2 = sc.textFile("data/uber.csv").filter(lambda line: "base" not in line).map(lambda
line:line.split(","))
# >
# Aplicando redução para calcular o total por base
uberRDD2.map(lambda kp: (kp[0], int(kp[3])) ).reduceByKey(lambda k,v: k + v).collect()
# >
# Aplicando redução para calcular o total por base, em ordem decrescente
uberRDD2.map(lambda kp: (kp[0], int(kp[3])) ).reduceByKey(lambda k,v: k + v).takeOrdered(10, key =
lambda x: -x[1])
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

5.8. Apache Spark SQL

Introdução

O que vamos estudar neste capítulo?

- Computação em Nuvem e em Cluster
- Linguagem SQL
- Spark SQL com RDD's
- Spark SQL com Dataframes
- Spark SQL com Arquivos CSV
- Spark SQL com Arquivos JSON
- Spark SQL com Bancos de Dados Relacionais
- Spark SQL com Bancos de Dados Não-Relacionais
- Tabelas Temporárias com Spark SQL
- Como Construir um Cluster Spark em Nuvem

Computação em Nuvem e em Cluster

Não temos como processar e armazenar Big Data em apenas uma máquina.

E a solução criada para poder armazenar e processar todo esse volume de dados, é a Cloud Computing. O conceito de Cloud Computing ou Computação em Nuvem é relativamente simples.

Um cluster é um sistema que compreende 2 ou mais computadores (chamados nodes) que trabalham em conjunto para executar aplicações ou realizar outras tarefas.

Por que não usar máquinas de baixo custo, em vez de supercomputadores, e depois unir essas máquinas de baixo custo em clusters, de modo que pudéssemos criar um único sistema com milhares de máquinas, milhares de CPU’s e muita, muita memória RAM?

Google: 1 a 5% dos HD’s vão falhar por ano e cerca de 0,2% de falhas com memória.

Ok, resolvemos o problema de armazenamento e processamento, distribuindo os dados por clusters (conjuntos de computadores) espalhados por datacenters em todo mundo e oferecendo este serviço pela internet! Mas como vamos processar esses dados distribuídos? Com MapReduce.

Problema 1: Armazenamento e Processamento → Cloud e Cluster Computing

Problema 2: Processamento Distribuído → MapReduce

Problema 3: Performance → Computação na memória dos computadores em cluster (Apache Spark)

Linguagem SQL – Introdução

Usamos Linguagem SQL para consultar e manipular dados em bancos de dados.

ANSI (American National Standards Institute) – Padroniza implementações da linguagem SQL.

Linguagem SQL – Instruções SQL

Os comandos SQL se dividem em 3 tipos principais:

- Linguagem de Manipulação de Dados ou DML (Data Manipulation Language)
- Linguagem de Definição de Dados ou DDL (Data Definition Language)
- Linguagem de Controle de Dados ou DCL (Data Control Language)

Instruções SELECT (DQL):

- Cláusulas (FROM, WHERE, GROUP BY, HAVING, ORDER BY, DISTINCT)
- Operadores Lógicos (AND, OR, NOT)
- Operadores de Comparação (<, >, <>, <=, =, >=, BETWEEN, LIKE)
- Funções de Soma (AVG, COUNT, SUM, MIN, MAX)

Instruções DML:

- INSERT: utilizado para inserir registros em uma tabela.

  Exemplo: INSERT into CLIENTE(ID, NOME) values(1000,'Obama');

- UPDATE: utilizado para alterar valores de uma ou mais linhas de uma tabela.

  Exemplo: UPDATE CLIENTE set NOME = 'Angela' WHERE ID = 1000;

- DELETE: utilizado para excluir um ou mais registros de uma tabela.

  Exemplo: DELETE FROM CLIENTE WHERE ID = 1000;

Instruções DDL:

- CREATE: utilizado para criar objetos no banco de dados.

  Exemplo (criar uma tabela): CREATE TABLE FORNECEDOR (ID INT PRIMARY KEY, NOME VARCHAR(50));

- ALTER: utilizado para alterar a estrutura de um objeto.

  Exemplo (adicionar uma coluna em uma tabela existente): ALTER TABLE FORNECEDOR ADD TELEFONE CHAR(10);

- DROP: utilizado para remover um objeto do banco de dados.

  Exemplo (remover uma tabela): DROP TABLE FORNECEDOR;

Instruções DCL:

- GRANT: autoriza um usuário a executar alguma operação.

  Exemplo (dar permissão de consulta na tabela fornecedor para o usuário Obama): GRANT select ON fornecedor TO obama;

- REVOKE: restringe ou remove a permissão de um usuário executar alguma operação.

Exemplo (não permitir que o usuário obama crie tabelas no banco de dados): REVOKE CREATE TABLE FROM obama;

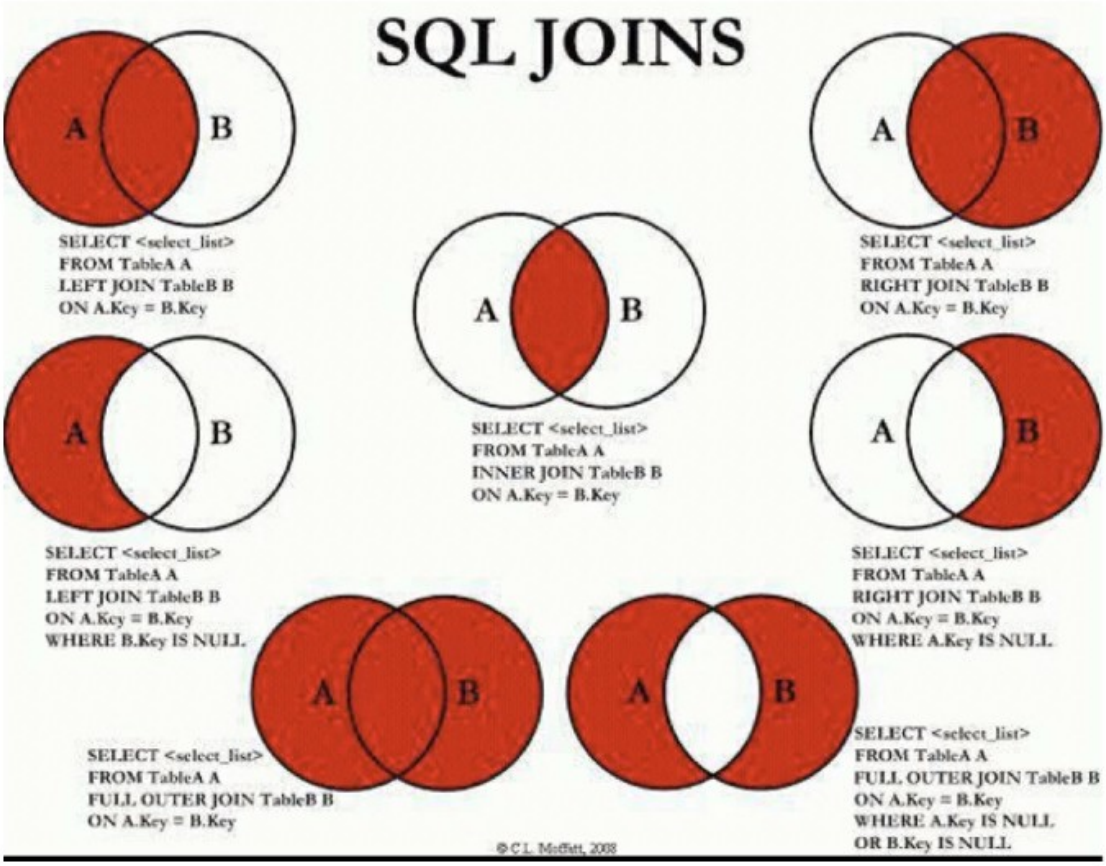
Instruções DTL (Linguagem de controle de transações):

- BEGIN TRANSACTION
- COMMIT
- ROLLBACK

Linguagem SQL – Terminologia

- Relação – essência do modelo de dados relacional.
- Tabela – conjunto de linhas e colunas onde se armazenam os dados.
- Coluna – atributo da entidade.
- Linhas – registros ou entidades da tabela.
- Chave Primária (Primary Key) – identificador único do registro na tabela.

SQL Joins:



O Que é o Apache Spark SQL?

É parte integrante do framework apache spark, utilizado para processamento de dados estruturados e que permite executar consultas SQL em grandes conjuntos de dados.

É possível executar tarefas de ETL sobre os dados, em diferentes formatos, como arquivos JSON, CSV, bancos de dados relacionais e não-relacionais.

Principais Funcionalidades do Spark SQL:

- Dataframes
- API Data Source
- Servidor JDBC Interno (Java Database Conection)
- Funcionalidades para Data Science

Componentes do Apache Spark SQL

O Spark SQL possui 3 componentes principais: DataFrame, Spark Session e SQL Context.

Dataframe:

- RDDs já existentes
- Arquivos de dados estruturados
- Conjunto de dados JSON
- Tabelas Hive
- Banco de dados externos
- Mais eficientes que os RDDs

Dataframe suporta as operações:

- filter
- join
- groupby
- agg
- Operações podem ser aninhadas

Spark Session:

- Criamos um Spark Session para acessar as funcionalidades do Spark SQL.
- Dataframes são criados a partir de Spark Sessions.
- Permite registrar um dataframe como uma tabela temporária e executar queries SQL sobre ele (muito útil no processamento de streams de dados).

SQL Context:

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

Fontes de Dados JDBC:

Tabelas Temporárias:

Uma query executada numa tabela temporária retornará um outro dataframe.

Uma das principais vantagens do Spark SQL é que você não precisa reaprender nada. Podemos usar os mesmos conceitos de SQL que usamos em bancos de dados relacionais, extrair dados para o Spark e então nos beneficiarmos do processamento paralelo e distribuído fornecido pelo framework Spark.

Ajustando a Versão do Java JDK

JVM é um ambiente de execução.
Java Code (.java) > JAVAC compiler > Byte Code (.class) > JVM > SO (Windows, Linux, Mac).
Instala-se o JVM instalando o JDK.

Spark SQL – Spark Session e SQL Context /

(Antes de trabalhar com Spark SQL, é necessário criar a Spark session e o SQL context)

Spark SQL – RDDs e a Função Row /

Spark SQL – Criando Dataframes /

Spark SQL – Aplicando SQL a Tabelas Temporárias /

Spark SQL e Arquivo CSV /

Spark SQL e Machine Learning /

Spark SQL com Arquivos JSON /

Spark SQL e Tabelas Termporárias /

Spark SQL com Banco de Dados Relacional – Manipulando os Dados com Spark SQL /

Leitura dos Dados do MongoDB com Spark

Gravação dos Dados no MongoDB com Spark

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 8</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 e Apache Spark 2.4.2
Java JDK 1.8:
https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
*Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório
metastore_db no mesmo diretório onde está este Jupyter notebook*
# Spark SQL
O Spark SQL é usado para acessar dados estruturados com Spark.
#### Acesse http://localhost:4040 sempre que quiser acompanhar a execução dos jobs.
#### Pacotes adicionais podem ser encontrados aqui: https://spark-packages.org/ (usaremos um destes
pacotes para conexão com o MongoDB).
## Spark SQL - Spark Session e SQL Context
# ->
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
from pyspark.sql import Row
# ->
print(sc)
# ->
# Spark Session - usado para trabalhar com o Spark
## getOrCreate: Se já existe a sessao spark, se conecta a ela; Caso contrario, uma nova  sera
criada
spSession = SparkSession.builder.master("local").appName("DSA-SparkSQL").getOrCreate()
# ->
# Criando o SQL Context para trabalhar com Spark SQL
sqlContext = SQLContext(sc)
# ->
# Importando o arquivo e criando um RDD
linhasRDD1 = sc.textFile("data/carros.csv")
# ->
linhasRDD1.count()
# ->
# Removendo a primeira linha - Transformação 1 (transformacao executada, mas nao processada)
linhasRDD2 = linhasRDD1.filter(lambda x: "FUELTYPE" not in x)
# ->
linhasRDD2.count() # transformacao sendo processada agora
# ->
# Dividindo o conjunto de dados em colunas - Transformação 2
linhasRDD3 = linhasRDD2.map(lambda line: line.split(","))
# ->
# Dividindo o conjunto de dados em colunas - Transformação 3
## a funcao Row é usada para montar linhas claramente definidas (o RDD nao está claramente
definido)
linhasRDD4 = linhasRDD3.map(lambda p: Row(make = p[0], body = p[4], hp = int(p[7])))
# ->
print(linhasRDD4) # não existe ainda, porque não for processado
# ->
?Row
# ->
```

```
linhasRDD4.collect() # processando RDD
# ->
# Criando um dataframe a partir do RDD
linhasDF = spSession.createDataFrame(linhasRDD4)
# ->
linhasDF.show() # mostra o dataframe criado
# ->
type(linhasDF)
# ->
# Mesma coisa que: SELECT * FROM linhasDF
linhasDF.select("*").show() # comando select através de uma função.
# ->
# Mesma coisa que: SELECT * FROM linhasDF ORDER BY make
linhasDF.orderBy("make").show()
# ->
# Registrando o dataframe como uma Temp Table (criando uma tabela temporária na memória do
computador,
# para utilização da linguagem SQL no padrão ANSI)
linhasDF.createOrReplaceTempView("linhasTB")
# ->
!java -version
# ->
# Executando queries SQL ANSI
## show() imprime na tela
spSession.sql("select * from linhasTB where make = 'nissan'").show()
# ->
# Executando queries SQL ANSI
spSession.sql("select make, body, avg(hp) from linhasTB group by make, body").show()
## Spark SQL e Arquivos CSV
# ->
# criando o dataframe direto do arquivo csv (sem criação de RDD)
carrosDF = spSession.read.csv("data/carros.csv", header = True)
# ->
type(carrosDF)
# ->
carrosDF.show()
# ->
# Registrando o dataframe como uma Temp Table
carrosDF.createOrReplaceTempView("carrosTB")
# ->
# Executando queries SQL ANSI
spSession.sql("select make, hp, price from carrosTB where CYLINDERS = 'three'").show()
# ->
carrosTT = spSession.sql("select make, hp, price from carrosTB where CYLINDERS = 'three'")
# ->
carrosTT.show()
## Aplicando Machine Learning
# ->
# Carregando o arquivo CSV e mantendo o objeto em cache
## textFile importa o csv e cria um RDD
## cache coloca o RDD em memória, para um processamento mais rápido
carros = sc.textFile("data/carros.csv")
carros.cache()
# ->
# Remove a primeira linha (header/cabeçalho) do RDD
## first coleta a primeira linha do RDD
primeiraLinha = carros.first()
linhas = carros.filter(lambda x: x != primeiraLinha)
linhas.count()
# ->
# Importando função row
from pyspark.sql import Row
# ->
# Convertendo para um vetor de linhas
def transformToNumeric(inputStr) :
    attList = inputStr.split(",") # separa colunas por vírgula
    doors = 1.0 if attList[3] == "two" else 2.0
    body = 1.0 if attList[4] == "sedan" else 2.0
    # Filtrando colunas não necessárias nesta etapa
    valores = Row(DOORS = doors, BODY = float(body), HP = float(attList[7]), RPM =
float(attList[8]), MPG = float(attList[9]))
    return valores
# ->
# Aplicando a função aos dados e persistindo o resultado em memória
## o resultado será uma lista de linhas
autoMap = linhas.map(transformToNumeric)
autoMap.persist() # mantém na memória
autoMap.collect() # coleta o resultado
# ->
```

```
# Criando o Dataframe
carrosDf = spSession.createDataFrame(autoMap)
carrosDf.show()
# ->
# Sumarizando as estatísticas do conjunto de dados (describe)
## toPandas converte para um dataframe do pandas
summStats = carrosDf.describe().toPandas()
summStats
# ->
# Extraindo as médias (criando uma lista)
## iloc faz localização por índice
medias = summStats.iloc[1,1:5].values.tolist()
medias
# ->
# Extraindo o desvio padrão (criando uma lista)
desvios_padroes = summStats.iloc[2,1:5].values.tolist()
desvios_padroes
# ->
# Inserindo a média e o desvio padrão em uma variável do tipo broadcast
bcMedias = sc.broadcast(medias)
bcDesviosP = sc.broadcast(desvios_padroes)
# ->
# Importando a Função Vectors
## linalg = algébra linear
from pyspark.ml.linalg import Vectors
# ->
# Função para normalizar os dados e criar um vetor denso
def centerAndScale(inRow) :
    global bcMedias
    global bcDesviosP
    meanArray = bcMedias.value
    stdArray = bcDesviosP.value
    retArray = []
    for i in range(len(meanArray)):
        retArray.append( (float(inRow[i]) - float(meanArray[i])) / float(stdArray[i]) )
    return Vectors.dense(retArray)
# ->
# Aplicando a normalização aos dados
## rdd converte o dataframe num RDD temporário, para poder usar o map e aplicar uma transformação
(nesse caso, a normalização)
csAuto = carrosDf.rdd.map(centerAndScale)
csAuto.collect()
# ->
# Criando um Spark Dataframe com as features (atributos)
autoRows = csAuto.map(lambda f: Row(features = f))
autoDf = spSession.createDataFrame(autoRows)
autoDf.select("features").show(10)
# ->
# Importando o algoritmo K-Means para clusterização
from pyspark.ml.clustering import KMeans
kmeans = KMeans(k = 3, seed = 1) # k=3 é a divisão dos dados em 3 grupos
modelo = kmeans.fit(autoDf)
previsoes = modelo.transform(autoDf)
previsoes.show()
# ->
# Plot dos resultados
import pandas as pd
import matplotlib.pylab as plt
%matplotlib inline
# ->
# Função para leitura dos dados e plotagem (desnormalizando os dados para poder montar o grafico)
def unstripData(instr) :
    return ( instr["prediction"], instr["features"][0], instr["features"][1],instr["features"]
[2],instr["features"][3])
# ->
# Organizando os dados para o Plot
unstripped = previsoes.rdd.map(unstripData)
predList = unstripped.collect()
predPd = pd.DataFrame(predList)
# ->
plt.cla()
plt.scatter(predPd[3], predPd[4], c = predPd[0])
## Spark SQL e Arquivos JSON
Neste site você pode validar a estrutura de um arquivo JSON: http://jsonlint.com/
# ->
# Importando o arquivo JSON (criando uma tabela automaticamente pelo uso da função json)
funcDF = spSession.read.json("data/funcionarios.json")
# ->
funcDF.show()
```

```
# ->
# printSchema mostra o padrão de organização do arquivo (para dados estruturados)
funcDF.printSchema()
# ->
type(funcDF)
# ->
# Operações com Dataframe Spark SQL - select()
funcDF.select("nome").show()
# ->
# Operações com Dataframe Spark SQL - filter()
funcDF.filter(funcDF["idade"] == 50).show()
# ->
# Operações com Dataframe Spark SQL - groupBy()
funcDF.groupBy("sexo").count().show()
# ->
# Operações com Dataframe Spark SQL - groupBy()
## agg usado para agregar a média do salario e o máximo de idade
funcDF.groupBy("deptid").agg({"salario": "avg", "idade": "max"}).show()
# ->
# Registrando o dataframe como uma Temp Table
funcDF.registerTempTable("funcTB")
# ->
# Executando queries SQL ANSI
spSession.sql("select deptid, max(idade), avg(salario) from funcTB group by deptid").show()
## Temp Tables
# ->
# Registrando o dataframe como temp Table
funcDF.createOrReplaceTempView("funcTB")
# ->
spSession.sql("select * from funcTB where salario = 9700").show()
# ->
# Criando Temp Table
sqlContext.registerDataFrameAsTable(funcDF, "funcTB2")
# ->
type(funcTB2) # erro: funcTB2 não é um objeto definido (é um objeto criado apenas na memória, em
tempo de execução)
# ->
# Persistindo a Temp Table  (aqui é onde se está criando um objeto definido, a partir da outro em
memória)
funcTB3 = spSession.table("funcTB2")
# ->
type(funcTB3) # tipo do objeto é um dataframe
# ->
# Comparando o Dataframe com a tabela temporária criada
sorted(funcDF.collect()) == sorted(funcTB3.collect())
# ->
# Aplicando o filtro
sqlContext.registerDataFrameAsTable(funcDF, "funcTB2")
funcTB3 = spSession.table("funcTB2")
funcTB3.filter("idade = '42'").first()
# ->
# Drop Temp Table
sqlContext.dropTempTable("funcTB2")
## Banco de Dados Relacional
Extraindo Dados do MySQL. Primeiro precisamos baixar o driver JDBC. Haverá um driver JDBC para cada
banco de dados que você conectar (Oracle, SQL Server, etc...)
1- Download do Driver JDBC para o MySQL: http://dev.mysql.com/downloads/connector/j/
2- Baixar o arquivo .zip
3- Descompactar o arquivo e copiar o arquivo mysql-connector-java-8.0.16.jar para a pasta
/opt/Spark/jars ou para SO Windows em C:\Spark\jars
# ->
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
spSession = SparkSession.builder.master("local").appName("DSA-SparkSQL").getOrCreate()
sqlContext = SQLContext(sc)
# ->
# Criando a conexão com banco de dados mysql:
## format para definição do formato jdbc
## option para as opções de conexão
mysql_df = spSession.read.format("jdbc").options(
    url = "jdbc:mysql://localhost/carros",
 # banco de dados carros na máquina local
    serverTimezone = "UTC",
    driver = "com.mysql.jdbc.Driver", # classe java no arquivo .jar
    dbtable = "carrosTB", # nome da tabela
    user = "root", # nome do usuário
    password = "dsa1234").load() # senha do usuário
# ->
mysql_df.show()
```

```
# ->
mysql_df.registerTempTable("carrostb") # converte dataframe numa tabela temporária
# ->
spSession.sql("select * from carrostb where hp = '68'").show() # aplica linguagem sql
## Banco de Dados Não-Relacional
Spark Connector: https://docs.mongodb.com/spark-connector/current/
Mongo Spark: https://spark-packages.org/package/mongodb/mongo-spark
$SPARK_HOME/bin/pyspark --packages org.mongodb.spark:mongo-spark-connector_2.11:2.4.0
# ->
# Imports
from pyspark.sql import SparkSession
### Leitura
# ->
# Cria a sessão
## \ permite quebrar a linha e continuar o comando
my_spark = SparkSession \
    .builder \
    .appName("myApp") \
    .config("spark.mongodb.input.uri", "mongodb://localhost/test_db.test_collection") \
    .config("spark.mongodb.output.uri", "mongodb://localhost/test_db.test_collection") \
    .getOrCreate()
# ->
# Carrega os dados do MongoDB no Spark
dados = spark.read.format("com.mongodb.spark.sql.DefaultSource").load()
# ->
dados.printSchema()
# ->
dados.count()
# ->
dados.head()
# ->
dados.show()
### Gravação
# ->
# criando um dataframe com spark
registro = spark.createDataFrame([("Camisa T-Shirt",  50)], ["item", "qty"])
# ->
# Grava os dados do Spark no MongoDB
## mode("append") adiciona ao final
## save() funciona como um COMMIT
registro.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").save()
# ->
dados.show()
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Instalando e Configurando o Banco de Dados MySQL no Windows

Faça o download:

No Google: “mysql download”.
Clique no lik MySQL Community Edition (GPL – livre para utilização) >
Clique em MySQL Community Server >
Selecione MySQL Installer MSI.

Após o download, inicie o processo de instalação:

Em Choosing Setup Type, clique em Custom
Selecione MySQL Server (versão – X64
Selecione Application > MySQL Workbench (software para acessar o banco de dados)
Selecione MySQL Shell (caso necessite de uma linha de comando)
Selecione Documentation > MySQL Documentation e Samples and Examples;
Em Group Replicacion, selecione Standalone MySQL Server;
Em Type and Networking, selecione Development Computer;
Em Authentication Method, selecione Use Strong Password Encryption for Authentication;

Em Accounts and Roles:

Em Root account password, digite uma senha de administrador

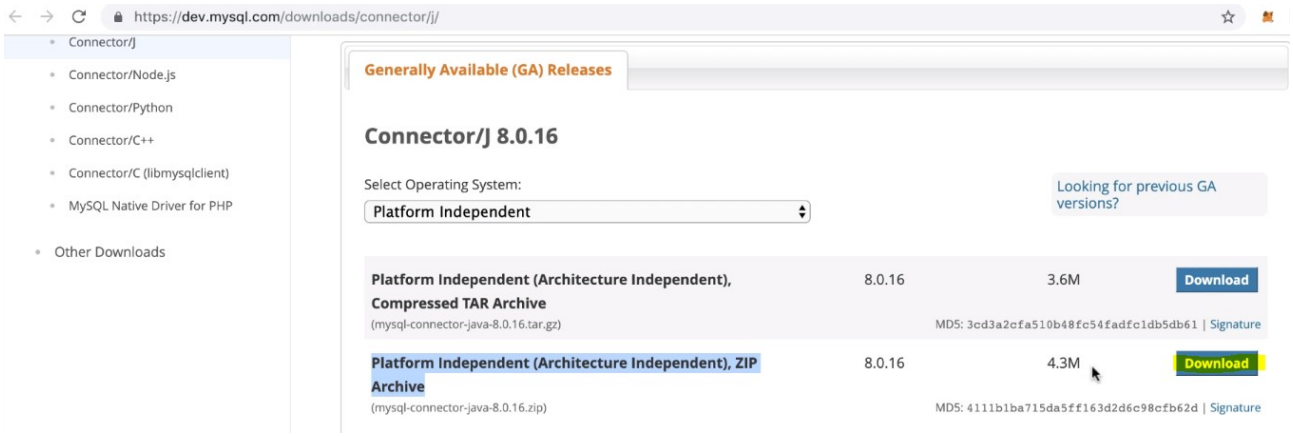
Em Add User, digite um nome para o usuário e selecione, em Role, DB Admin;

Mantenha tudo como padrão nas próximas telas e clique em Execute, para iniciar a instalação/configuração.

Para abrir o MySQL Workbench, basta pesquisar no Windows.

Spark SQL com Banco de Dados Relacional – Instalando o Driver JDBC

1- Download do Driver JDBC para o MySQL: http://dev.mysql.com/downloads/connector/j/



2- Baixar o arquivo .zip

3- Descompactar o arquivo e copiar o arquivo mysql-connector-java-8.0.16.jar para a pasta /opt/Spark/jars ou para SO Windows em C:\Spark\jars

Spark SQL com Banco de Dados Relacional – Criando o Banco de Dados e Importando os Dados

Acessar o MySQL Workbench e criar um banco de dados.

Instalando e Configurando o MongoDB no Windows

Baixando o MongoDB:

Acesse o site do mongodb;
Clique em em GetMongoDB (canto superior);
Selecione a aba Server:

Em Version, selecione a versão corrente (current release);

Em Package, selecione o pacote MSI;

Clique em Download.

Baixando o Studio 3T:

Acesse o site robomongo.org;
Clique em Download Studio 3T;
Clique em Download for Windows.

Instalando o MongoDB:

Clique no arquivo de instalação;
Selecione Custom como tipo de instalação;
Clique em Next;
Selecione Run service as Network Service user;
Desmarque a opção Install MongoDB Compass (interface gráfica do MongoDB).

Instalando o Studio 3T:

Descompacte o arquivo;
Execute o arquivo exe;
Instale no padrão.

Iniciando o MongoDB:

Clique com o botão direito no menu iniciar > Executar
Digite services.msc
Verifique se o MongoDB Server está em execução
(caso prefira, clique 2x no serviço e mude a inicialização de automática para manual)

Executando o Studio 3T:

Procure o programa e abra-o, normalmente;
Na tela inicial, clique em Connect > New Connection;
    Digite o nome MongoDB
    Em Server, digite localhost
    Em Port, digite 27017 (porta padrão)
    Clique em Save;
Clique em Connect.

Criando Banco de Dados no MongoDB e Conectando com PySpark

Trabalhando com Spark e MongoDB

1- Abra o prompt de comando ou terminal e inicialize o MongoDB com o comando: mongod.
(obs.: tecle ctrl+c para encerrar)
2- Abra outro prompt de comando ou terminal e digite para abrir o shell: mongo
(obs.: digite exit para encerrar)
3- Visualize os bancos de dados criados com o comando: show databases;
4- Digite: use test_db;
5- Crie uma coleção com o comando: db.createCollection("test_collection");
6- Carregue alguns dados na coleção criada:

```
db.test_collection.insertMany([
  { item: "Camisa Polo", qty: 25, tags: ["branco", "vermelho"], size: { h: 14, w: 21, uom: "cm" } },
  { item: "Vestido Bordado", qty: 85, tags: ["cinza"], size: { h: 27.9, w: 35.5, uom: "cm" } },
  { item: "Moleton", qty:45, tags: ["verde", "azul"], size: { h: 19, w: 22.85, uom: "cm" } }
]);
```

7-- Abra outro prompt de comando ou terminal e digite (no diretório onde se encontram os arquivos jupyter):

$SPARK_HOME/bin/pyspark --packages org.mongodb.spark:mongo-spark-connector_2.12:2.4.0

Ajustando a Versão do Conector

Use este comando com Apache Spark 2.4.3:
$SPARK_HOME/bin/pyspark --packages org.mongodb.spark:mongo-spark-connector_2.11:2.4.0
Use este comando com Apache Spark 2.4.2 ou inferior:
$SPARK_HOME/bin/pyspark --packages org.mongodb.spark:mongo-spark-connector_2.12:2.4.0

Dependendo de quando você fizer o download do Apache Spark, o suporte a linguagem Scala já pode ter sido atualizado e portanto você pode usar o comando $SPARK_HOME/bin/pyspark --packages org.mongodb.spark:mongo-spark-connector_2.12:2.4.0 para todos os casos.

Agrupamento com Pares RDDs / Acumuladores e Broadcast / Particionamento de RDDs

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 8</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 e Apache Spark 2.4.2
Java JDK 1.8:
https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
*Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório
metastore_db no mesmo diretório onde está este Jupyter notebook*
Acesse http://localhost:4040 sempre que quiser acompanhar a execução dos jobs
## Agrupamento com Pares RDD (Pair RDD)
Tipo especial de RDD que armazena pares chave-valor. Útil quando é necessáirio armazenar dados que
possuem uma chave e diversos valores (por exemplo, todas as transações de um cliente, geradas em
tempo real).
#### mapValues()
#### countByKey()
#### groupByKey()
#### reduceByKey()
#### aggregateByKey()
# ->
# Importando arquivo csv e criando um RDD
carros = sc.textFile("data/carros.csv")
# ->
carros.take(20) # Ação para imprimir na tela (20 primeiros registros)
# ->
# Criando uma Pair RDD
## [0] refere-se ao índice da coluna MAKE e [7] da coluna HP
## Operação de transformação (lazy evaluation - nada foi executado ainda, apenas definido)
carrosPairRDD = carros.map(lambda x: (x.split(",")[0], x.split(",")[7]))
## Agora ao executar uma ação, a transformação é executada:
carrosPairRDD.take(20)
# ->
# Removendo o cabeçalho
header = carrosPairRDD.first()
carrosPairRDD2 = carrosPairRDD.filter(lambda line: line != header)
# ->
# Encontra o valor de HP por marca de carro e adiciona 1 a cada registro "Make/HP"
## mapValues adiciona um valor para cada reg encontrado (count agrupado)
carrosPairRDD3 = carrosPairRDD2.mapValues(lambda x: (x, 1))
carrosPairRDD3.collect()
# ->
# Aplica redução por key (reduceByKey)
# E calcula o total de HP por fabricante e o total de automóveis por fabricante
fabricantes =  carrosPairRDD3.reduceByKey(lambda x, y: (int(x[0]) + int(y[0]), x[1] + y[1]))
fabricantes.collect()
# ->
# Calculando a média de HP dividindo pela contagem total
fabricantes.mapValues(lambda x: int(x[0])/int(x[1])).collect()
## Acumuladores e Broadcast
O Spark faz uma cópia do código que você escreveu para processar os dados e executa essas cópias,
uma por node do cluster. Qualquer variável criada no código é local ao node. O Spark gera cópias
dessas variáveis locais, uma em cada node, que agem de forma independente. Mas e se precisamos que
```

```
a mesma variável seja manipulada de forma única através de todo o cluster? Usamos Acumuladores e
Broadcast.
Variável Broadcast - read-only, é compartilhada em todo o cluster.
Variável Accumulator - é compartilhada em todo o cluster, mas pode ser atualizada em cada node do
cluster.
# ->
# Inicializando variáveis Accumulator
sedanCount = sc.accumulator(0)
hatchbackCount = sc.accumulator(0)
# ->
# Inicializando variáveis Broadcast
sedanText = sc.broadcast("sedan")
hatchbackText = sc.broadcast("hatchback")
# ->
def splitLines(line) :
    # variáveis definidas como global podem ser vistas por todo o programa.
    global sedanCount
    global hatchbackCount
    # Usa a variável Broadcast para comparar e adiciona a contagem ao accumulator
    if sedanText.value in line:
        sedanCount +=1
    if hatchbackText.value in line:
        hatchbackCount +=1
    return line.split(",")
# ->
# Map()
splitData = carros.map(splitLines)
# ->
# Ação para executar a transformação (lazy evaluation)
splitData.count()
print(sedanCount, hatchbackCount)
## Partições
Sempre que criamos RDD's, esses objetos são divididos em partições e essas partições são
distribuídas através dos nodes do cluster. Por default, os RDD's são sempre particionados. Essas
partições precisam ser configuradas quando se trabalha com grandes clusters.
# ->
# Recuperando o número de partições de um RDD
fabricantes.getNumPartitions()
# ->
# Especificando o Número de Partições
collData = sc.parallelize([3,5,4,7,4], 3) # transformando uma lista num RDD
collData.cache()
collData.count()
# ->
collData.getNumPartitions()
# ->
# defaultParallelism é um parâmetro, que se pode alterar, que define o número de partições padrão
de paralelismo.
print (sc.defaultParallelism)
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Quiz

Para resolver o problema da computação em cluster, uma nova tecnologia foi criada, o MapReduce, desenvolvido pelo Google em 2004 e depois entregue a Fundação Apache como um projeto open-source e atualmente um dos principais projetos do Hadoop.

O Spark SQL permite conexão a diferentes fontes de dados através de conexão JDBC.

As funções filter, group by e join são suportadas em um dataframe.

A função grant é uma instrução SQL DCL, não suportada em dataframes no Spark SQL.

O Spark SQL permite a execução de joins entre tabelas.

O conceito por trás do MapReduce é o de ”dividir para conquistar”, ou seja, primeiro separamos os dados e os mapeamos e depois agrupamos novamente, reduzindo até atingir o resultado que buscamos. Essa tecnologia tem se mostrado muito útil para processar Big Data. computação em cluster é praticamente o padrão para processamento de grandes conjuntos de dados.

5.9. Real-Time Analytics com Spark Streaming

Introdução

A vida não acontece em batches! Exemplos: compras no cartão de crédito, frequência de batimentos cardíacos, sinais de GPS, etc.

Com Spark podemos processar dados a medida em que são gerados.

São muitas as vantagens na manipulação de dados como fluxos.

Streaming de dados não é apenas para projetos altamente especializados. Computação baseada em Streaming está se tornando a regra para empresas orientadas a dados.

Novas tecnologias e projetos de arquitetura permitem construir sistemas flexíveis!

Sensores de IOT (Internet das Coisas) = Dados Contínuos

E o que vamos estudar sobe Spark Streaming?

- Arquitetura Spark Streaming
- DStreams
- Windowing
- Integração com outros sistemas – Kafka, Flume, Amazon Kinesis
- Processamento de Linguagem Natural – NLTK
- Análise de Sentimentos do Twitter em Tempo Real (próximo capítulo)
- Deploy em Cluster na Nuvem (próximo capítulo)

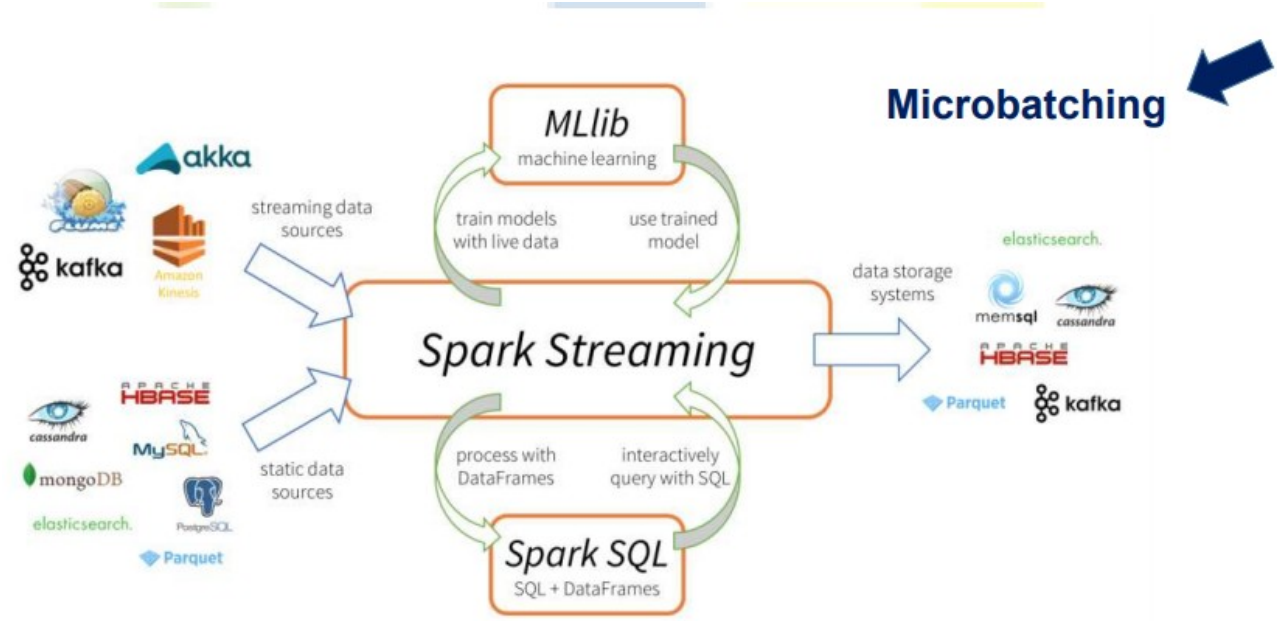
Batch x Streaming

Existem 2 formas de processar e analisar dados:

Batch	Streaming
Você inicia o processamento de um arquivo ou dataset finito, o Spark processa as tarefas configuradas e conclui o trabalho.	Você processa um fluxo de dados contínuo (Stream); a execução não para até que haja algum erro ou você termine a aplicação manualmente.
• Análise exploratória de dados • Analisar dados de Data Warehouses • Treinar um modelo de aprendizado de máquina sobre grandes conjuntos de dados • Outras tarefas analíticas feitas com Hadoop MapReduce	• Monitoramento de serviços • Processamento de eventos em tempo real para alimentar dashboards • Processamento de dados de cliques e eventos em web sites • Processamento de dados de sensores de Internet das Coisas • Processamento de dados vindos de serviços como: Twitter, Kafka, Flume, AWS Kinesis

Big Data never stops!

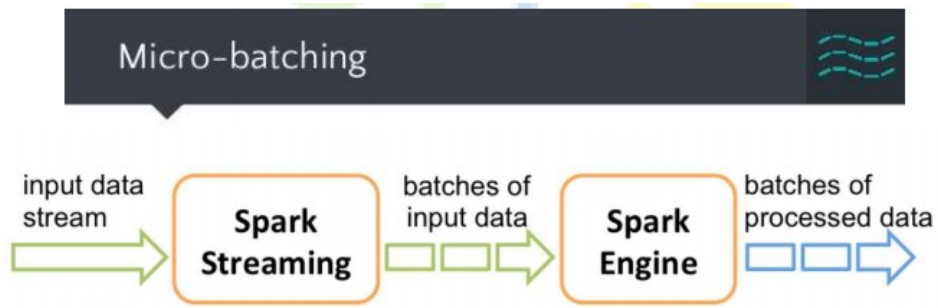
Apache Spark Streaming



O Spark Streaming é um sistema de processamento de streaming de dados, ou seja, um fluxo continuo de dados, que é tolerante a falhas e escalável, o que significa que é possível aumentar a quantidade de máquinas em cluster e, assim, expandir a capacidade de processamento e análise do Spark Streaming.

O conceito de Microbatching significa que um programa em batch é executado em intervalos frequentes, para processar todos os dados ao longo do streaming, simulando um processamento em tempo real.

Quer dizer que o Spark Streaming não é ”real” real-time? Exatamente.



O Spark Streaming pode receber dados de diversas fontes:

- Flat Files (à medida que são criados)
- TCP/IP • Apache Flume
- Apache Kafka
- AWS Kinesis
- Mídias Sociais (Facebook, Twitter, etc...)
- Bancos NoSQL

- Bancos Relacionais

Existem 4 áreas principais nas quais o Spark Streaming vem sendo usado:

- Streaming ETL
- Detecção de Anomalias
- Enriquecimento de Dados
- Sessões Complexas e Aprendizado Contínuo

Mas também existem outras áreas de aplicação do Spark Streaming:

- Detecção de Fraudes em Tempo Real
- Filtro de Spam • Detecção de Invasão de Redes
- Análise de Mídias Sociais em Tempo Real
- Análise de Stream de Cliques em Sites, gerando Sistemas de Recomendação
- Recomendação de Anúncios em Tempo Real
- Análise do Mercado de Ações

Em Spark Streaming são realizadas 4 atividades principais:

1. Coleta e análise dos dados direto da fonte e à medida que são gerados
2. Transformação, sumarização e análise
3. Machine Learning
4. Previsões em tempo real

Uma importante vantagem de usar o Spark para Big Data Analytics é a possibilidade de combinar processamento em batch e processamento de streaming em um único sistema.

Streaming de Dados – A Velocidade Com Que Você Passa o Cartão de Crédito

No combate a fraudes com cartão de crédito, uma propriedade tem sido usada com frequência. A velocidade com que o cartão é passado na máquina leitora é usado como um indicador da probabilidade de atividade fraudulenta. A ideia por trás da velocidade do cartão é bastante simples.

São 2 os objetivos que precisam ser alcançados em uma arquitetura de detecção de fraude em tempo real, utilizando dados gerados em tempo real:

Objetivo 1: Quando um cliente utiliza o cartão de crédito para uma transação, o vendedor precisa saber em tempo real a resposta para a frase: É uma fraude?

Objetivo 2: Precisamos manter um histórico de decisões de fraude feitas pelo sistema. Esse histórico deve estar disponível para outros sistemas da empresa.

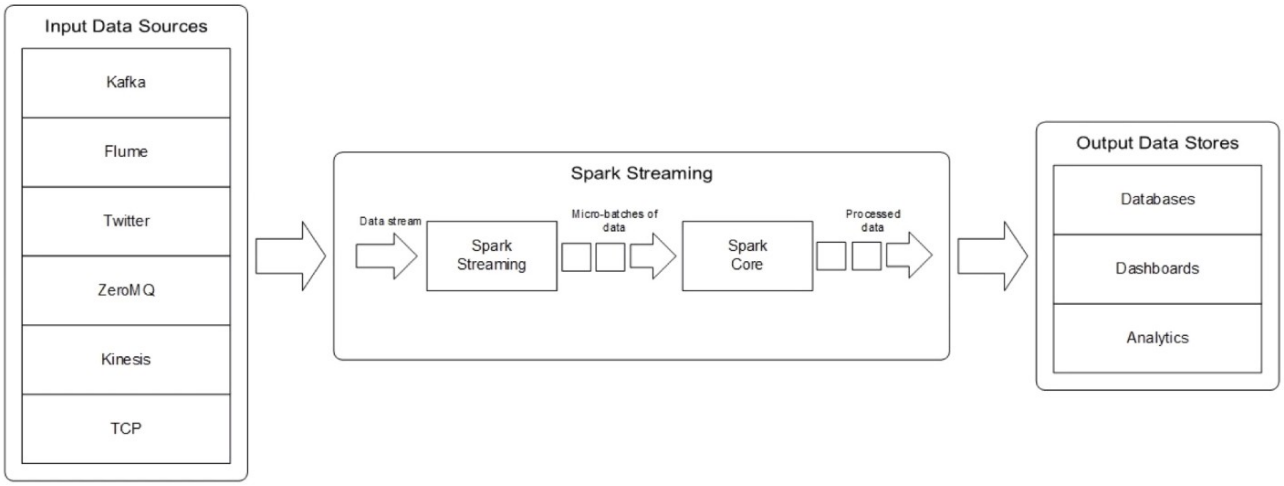
Um Streaming de dados é como um dataset contínuo, infinito. Vamos retirando porções deste dataset, analisando, tomando decisões e armazenando para análise futuras. Trabalhar com Streaming de dados é sem dúvida um grande desafio.

Arquitetura Apache Spark Streaming

Processamento iterativo – várias tarefas em sequência

Processamento interativo – análise exploratória de dados

O Spark Streaming pode receber dados de diferentes fontes, simultaneamente, e esses dados vão sendo processados e agrupados em microbatches. Aplicamos então as transformações ou procedimentos de análise nos dados, para gerar o resultado.



Internamente, o Spark Streaming recebe dados em tempo real e divide os dados em batches, chamados de microbatches, em um intervalo de tempo pré-definido e, então, trata cada batch como um RDD (Resilient Distributed Dataset).

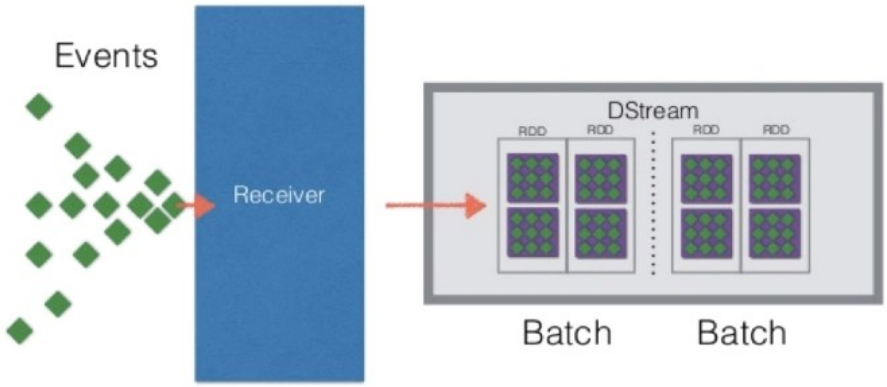
Podemos processar os RDDs usando operações de Map, Reduce, Join e Window. Os resultados em cada uma das operações dos RDDs são retornados, também, em batches e podemos armazenar os dados em sistemas de armazenamento para análise posterior, ou mesmo gerar dashboards em tempo real.

Principais Frameworks para processamento de Streaming de Dados:

- Apache Samza • Apache Storm
- Apache Flink
- Apache Spark Streaming
- AWS Kinesis (tem custo associado)

O Que São Dstreams (Discretized Streams)?

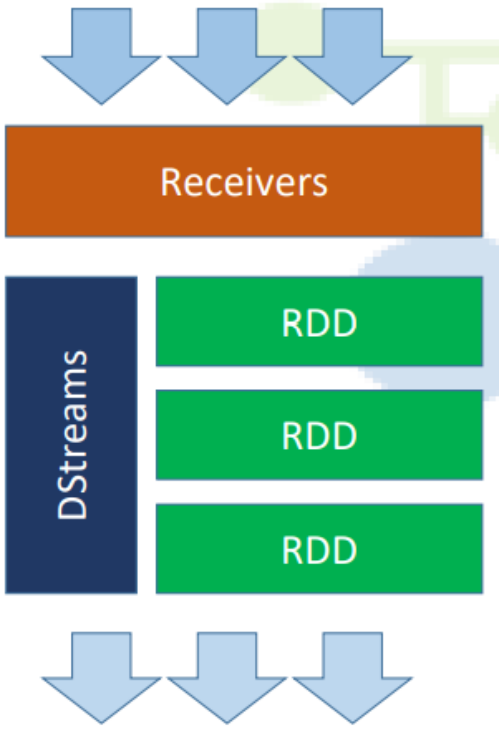
Um DStreams é uma sequência de dados que são coletados ao longo do tempo. Internamente, cada DStreams é representado por uma sequência de RDDs, coletados em cada unidade de tempo.



The DStream is (Discretized) into batches, the timing of which is set in the Spark Streaming Context. Each Batch is made up of RDDs.

Um DStream pode ser criado a partir de diversas fontes e uma vez criados eles oferecem 2 tipos de operações: transformações (que geram um novo DStream) e operações de output, ou seja, ações, que gravam os dados em um sistema de armazenamento ou alguma outra fonte externa.

Os DStreams oferecem muitas das operações que podem ser realizadas com os RDD's, mais as operações relacionadas ao tempo, como sliding windows.

<table>
<tr><td rowspan="3"> 
</td><td>Streams</td></tr>
<tr><td>DStreams – batches de dados para um determinado período de tempo

Processamento Paralelo</td></tr>
<tr><td>Transformação e output para outros sistemas</td></tr>
</table>

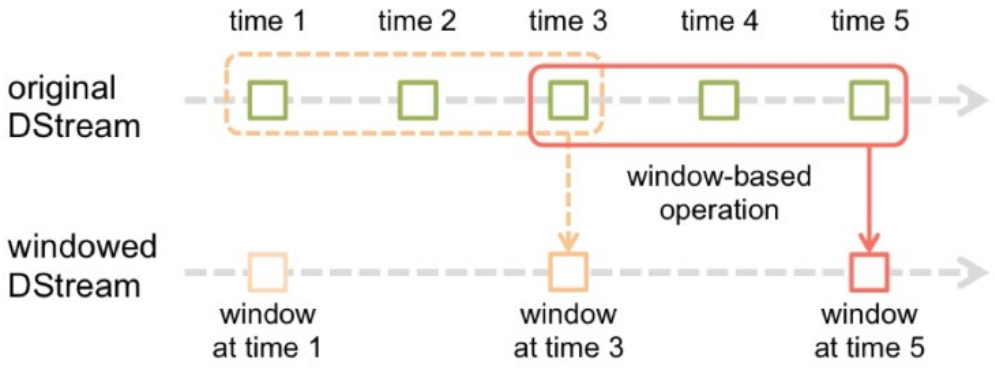
Podemos aplicar aos Dstreams as operações:
• Map
• FlatMap
• Filter
• reduceByKey
• Join • Window

Mas precisamos tomar alguns cuidados especiais, como manter o controle de estado dos dados (Stateful Data). Exemplo: coletar streams de cliques em um web site e manter a associação dos dados com a sessão ou ip do usuário.

O DStream permite converter um conjunto de dados contínuo em um conjunto discreto de RDD's (DStream significa Discretized Streams).

Windowing – Agregando Stream de Dados ao Longo do Tempo

Computing Windowed é um recurso que pode ser usado para aplicar operações de transformação sobre os dados, em uma janela específica.



Configurações possíveis:

- Window length – duração da window (3 unidades de tempo nesse caso)
- Sliding interval – intervalo entre as windows

ssc = StreamingContext(sc, INTERVALO_BATCH)
window(windowDuration: Duration, slideDuration: Duration): DStream[T]

Windowing permite computar os resultados ao longo de períodos de tempo maiores que o batch interval.

Os 3 tipos de intervalo:

Batch Interval	Sliding Interval	Window Interval
Frequência com que os dados são capturados em um DStream	Frequência com que uma window é aplicada	Intervalo de tempo capturado para computação e geração de resultados

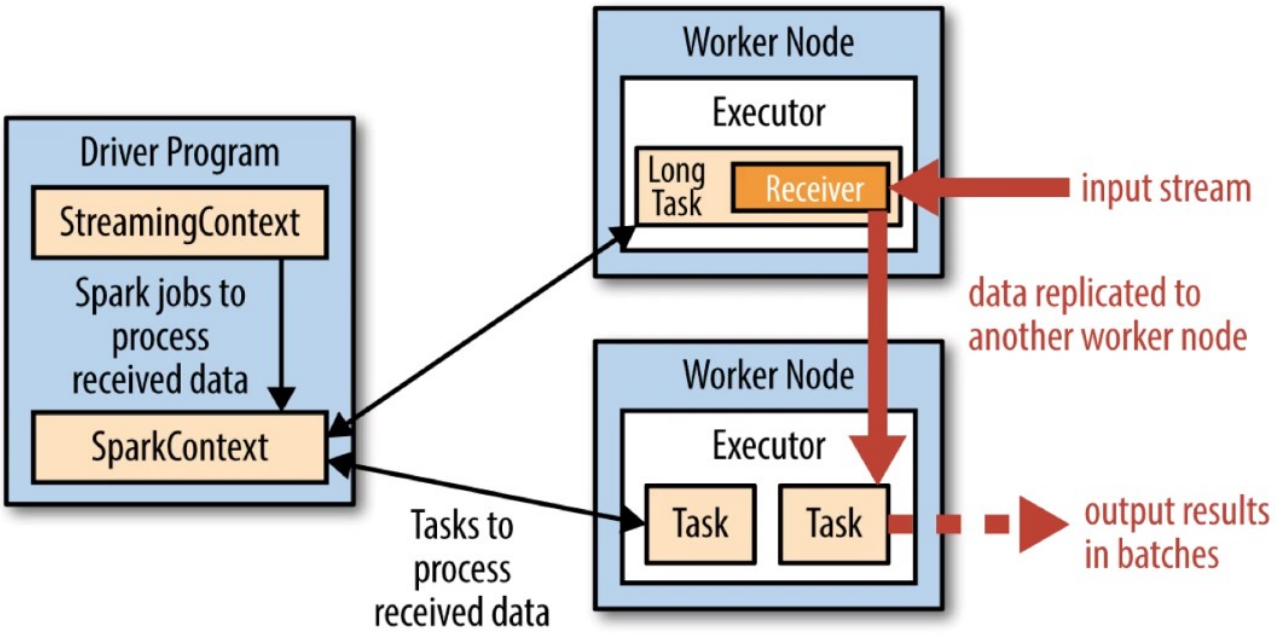
E existem diversas funções de transformação específicas para se trabalhar com Window:

Transformation	Meaning
window(*windowLength*, *slideInterval*)	Return a new DStream which is computed based on windowed batches of the source DStream.
countByWindow(*windowLength*, *slideInterval*)	Return a sliding window count of elements in the stream.
reduceByWindow(*func*, *windowLength*, *slideInterval*)	Return a new single-element stream, created by aggregating elements in the stream over a sliding interval using *func*. The function should be associative and commutative so that it can be computed correctly in parallel.
reduceByKeyAndWindow(*func*, *windowLength*, *slideInterval*, [*numTasks*])	When called on a DStream of (K, V) pairs, returns a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function *func* over batches in a sliding window. **Note:** By default, this uses Spark's default number of parallel tasks (2 for local mode, and in cluster mode the number is determined by the config property `spark.default.parallelism`) to do the grouping. You can pass an optional `numTasks` argument to set a different number of tasks.
reduceByKeyAndWindow(*func*, *invFunc*, *windowLength*, *slideInterval*, [*numTasks*])	A more efficient version of the above `reduceByKeyAndWindow()` where the reduce value of each window is calculated incrementally using the reduce values of the previous window. This is done by reducing the new data that enters the sliding window, and "inverse reducing" the old data that leaves the window. An example would be that of "adding" and "subtracting" counts of keys as the window slides. However, it is applicable only to "invertible reduce functions", that is, those reduce functions which have a corresponding "inverse reduce" function (taken as parameter *invFunc*). Like in `reduceByKeyAndWindow`, the number of reduce tasks is configurable through an optional argument. Note that checkpointing must be enabled for using this operation.
countByValueAndWindow(*windowLength*, *slideInterval*, [*numTasks*])	When called on a DStream of (K, V) pairs, returns a new DStream of (K, Long) pairs where the value of each key is its frequency within a sliding window. Like in `reduceByKeyAndWindow`, the number of reduce tasks is configurable through an optional argument.

Tolerância a Falhas

Uma das principais características do Spark Streaming é a sua tolerância a falhas.

Processo do Spark Streaming:



Todos os dados são replicados para no mínimo 2 worker nodes.

Um diretório de checkpoint pode ser usado para armazenar o estado do streaming de dados, no caso em que é necessário reiniciar o streaming.

ssc.checkpoint()

Pontos principais de falhas:

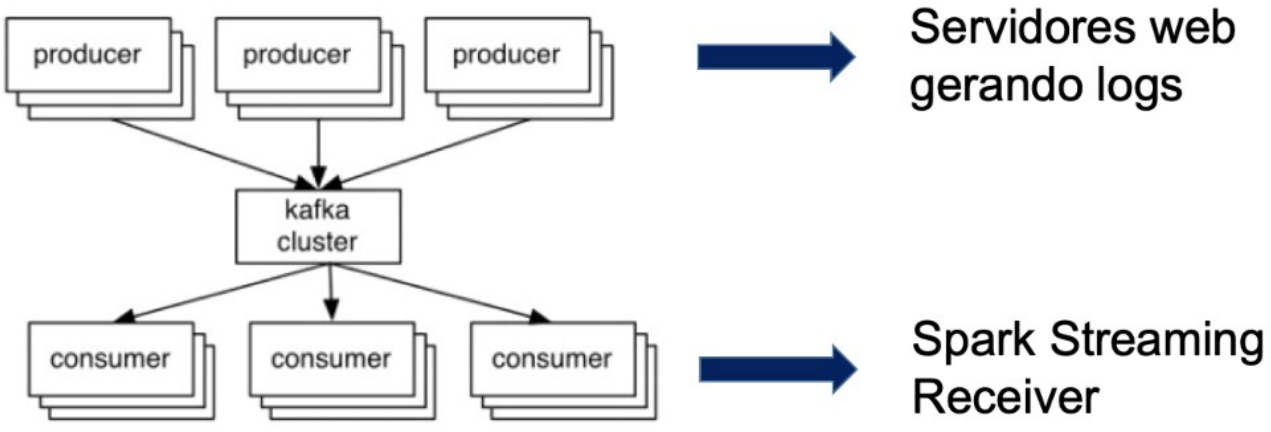
1. Falha no Receiver
    - Alguns receivers são melhores que outros.
    - Receivers como Twitter, Kafka e Flume são permitem recuperação de dados. Se o receiver falha, os dados do streaming são perdidos.
    - Outros receivers garantem a recuperação dos dados em caso de falhas: HDFS, Directely-consumed Kafka, Pull-based Flume.
2. Falha no Driver Context (script)
    - Embora os dados sejam replicados para os worker nodes, o DriverContext é executado no node Master e este pode ser um ponto único de falha.
    - Podemos usar checkpoint() para recuperar dados em caso de falhas e usamos a função StreamingContext.getOrCreate() para continuar o processamento de onde ele foi interrompido em caso de falha.
    - Em caso de falha no seu script sendo executado no DriverContext, podemos reiniciar automaticamente o processo de streaming, usando o Zookeeper (no modo supervise). O Zookeeper é um cluster manager usado pelo Spark.

Integração Com Outros Sistemas – Apache Kafka

O principal objetivo de usar o Spark Streaming é o processamento de dados em tempo real.

O Apache Kafka é um sistema para gerenciamento de fluxo de dados em tempo real, gerados a partir de websites ou outras aplicações.

É um software para sistema de mensageiria.



Além de ser um projeto open source de alta qualidade, possui a capacidade de lidar com um fluxo de alta velocidade de dados, característica cada vez mais procurada em IOT, por exemplo.

Principais motivos para a criação do Kafka:

1. Transportar dados entre diversos sistemas de dados
2. Enriquecer a análise de dados

Soluções Similares ao Kafka:

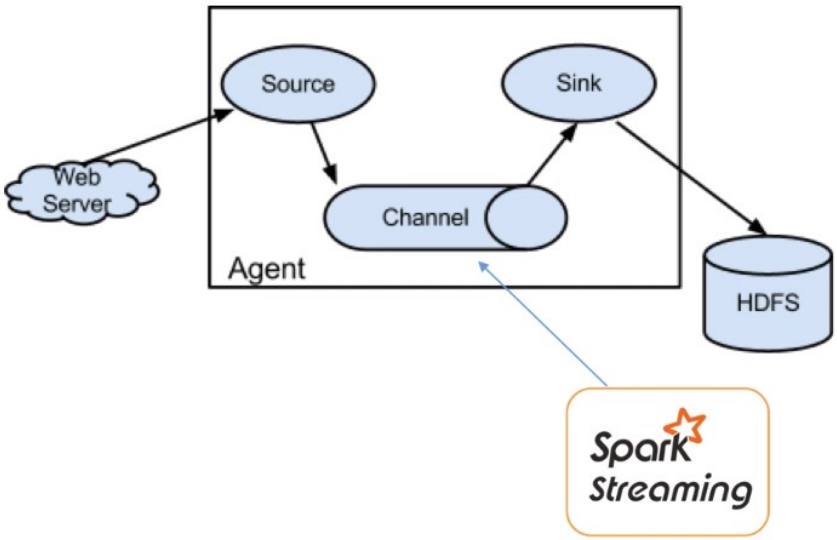
- IBM InfoSphere Streams
- Informatica’s Ultra Messaging Streaming Edition
- SAS’s Event Stream Processing Engine (ESP)
- Tibco’s StreamBase
- DataTorrent
- Splunk
- Loggly
- Logentries
- Glassbeam

É necessário instalar o pacote spark-streaming-kafka!

Integração Com Outros Sistemas – Apache Flume

Embora muito similar ao Apache Kafka, o Apache Flume tem o foco na coleta de logs de servidores, que podem ser muito valiosos em processos de análise de dados, detecção de fraudes ou invasão de redes, por exemplo.

Serviço distribuído e confiável, para eficientemente coletar, agregar e mover grandes quantidades de arquivos de log. Ele busca os arquivos de logs dos servidores e estabelece um canal, para gravar esses dados no HDFS, por exemplo, para uma análise posterior, ou entregar esses dados para processamento no Spark Streaming.



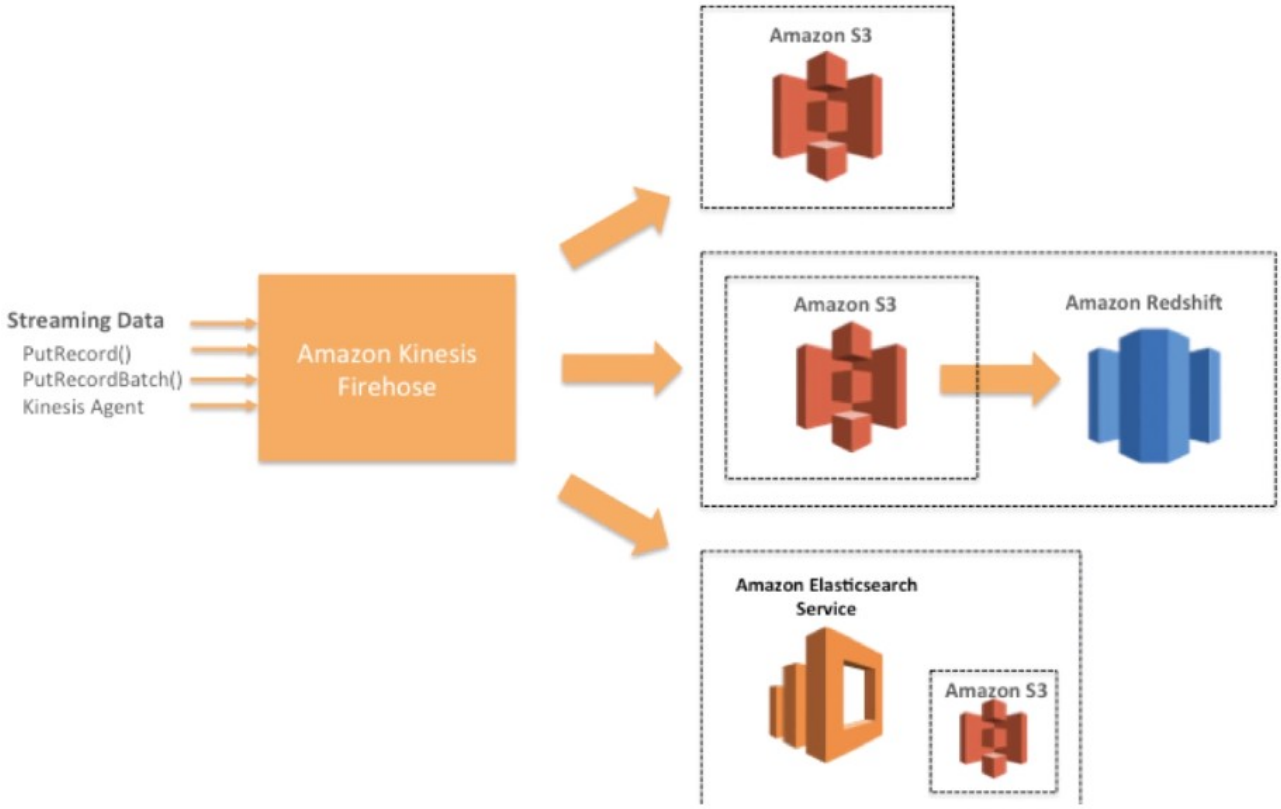
Existem 2 modos de configuração do Apache Flume:

Push-Based Flume – modo padrão, não confiável. Ou seja, caso hajam falhas, os dados serão perdidos.
É necessário instalar o pacote spark-streaming-flume – Push-Based

Pull-Based Flume – requer etapas adicionais de configuração, mas permite que se recuperem os dados em casos de falhas na infraestrutura de streaming.
É necessário instalar o pacote spark-streaming-flume-sink – Pull-Based

Integração Com Outros Sistemas – AWS Kinesis

Muito similar ao Kafka. Embora possua um custo de utilização, sua principal vantagem é poder utilizar uma excelente infraestrutura na nuvem, fornecida pela Amazon.



Com o Kinesis você pode coletar dados de diversas fontes e, então, entregá-los ao Apache Spark, por exemplo, para processamento paralelo e distribuído, ou então para aplicar modelos de aprendizagem de máquina de Machine Learning.

Para integração entre Spark Streaming e Kinesis, é necessário instalar o pacote spark-streaming-kinesis-asl (Requer Licença da Amazon!)

Introdução ao Processamento de Linguagem Natural – PLN

O computador espera que a linguagem humana seja precisa, não ambígua e altamente estruturada.

Talvez você não saiba, mas você utiliza PLN em aplicações como: • Corretores Ortográficos (Microsoft Word) • Engines de Reconhecimento de Voz (Siri, Google Assistente, Cortana) • Classificadores de Spam • Mecanismos de Busca (Google, Bing) • IBM Watson

Normalmente, o sistema de PLN é abordado do ponto de vista da abordagem do conhecimento de 4 áreas: Análise Morfológica, Análise Semântica, Análise Sintática e Análise Pragmática.

Morfológica	Sintática	Semântica	Pragmática
Estuda a construção das palavras, com seus radicais e afixos, que correspondem a partes estáticas e variantes das palavras, como as inflexões verbais.	Diz respeito ao estudo das relações formais entre as palavras.	É o processo de mapeamento de sentenças de uma linguagem, visando a representação do seu significado, baseado nas construções obtidas na análise sintática.	Diz respeito ao processamento da forma que a linguagem é usada para comunicação, como o significado é obtido na análise semântica e o contexto.

A Análise de Sentimento é o uso de PLN, análise de textos, mineração de dados e computação linguística, para identificar e extrair informação subjetiva em elementos de texto. É muito utilizada para revisões de mídias sociais, serviços de atendimento aos clientes e campanhas de marketing.

Em termos gerais, a análise de sentimentos tem como objetivo determinar a atitude de alguém, em relação a algum tópico, ou polaridade contextual em documentos.

Principais Frameworks para PLN:

- GATE (General Architecture for Text Engineering)
- Mallet (Machine Learning for Language Toolkit)
- OpenNLP
- UIMA
- Gensim
- SpaCy
- Natural Language Toolkit (NLTK)

NLTK – Processamento de Linguagem Natural em Python

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 9</font>
# Processamento de Linguagem Natural com Python - NLTK
## Instalação do pacote NLTK
http://www.nltk.org/install.html
# ->
# Instalação do módulo NLTK
!pip install nltk
# ->
import nltk
# ->
# Instalando os arquivos de dados do NLTK
# Clique em Download quando solicitado
nltk.download()
### Leia a definição e execute as células para compreender o código de cada uma e o conceito que
está sendo demonstrado
## Tokenization
Processo de dividir uma string em listas de pedaços ou "tokens". Um token é uma parte inteira. Por
exemplos: uma palavra é um token em uma sentença. Uma sentença é um token em um parágrafo.
### Dividindo um parágrafo em frases
# ->
paragrafo = "Oi. Bom saber que você está aprendendo PLN. Obrigado por estar conosco."
# ->
```

```
from nltk.tokenize import sent_tokenize
# ->
# Dividindo o parágrafo em frases
sent_tokenize(paragrafo)
# ->
import nltk.data
# ->
# Utilizando dados do pacote NLTK
tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')
# Load no windows
#tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
# ->
tokenizer.tokenize(paragrafo)
# ->
# Dados em espanhol
spanish_tokenizer = nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')
# Load no windows
#spanish_tokenizer = nltk.data.load('tokenizers/punkt/spanish.pickle')
# ->
spanish_tokenizer.tokenize('Hola amigo. Estoy bien.')
# ->
spanish_tokenizer
### Dividindo uma frase em palavras
# ->
from nltk.tokenize import word_tokenize
# ->
word_tokenize('Data Science Academy')
# ->
from nltk.tokenize import TreebankWordTokenizer
# ->
tokenizer = TreebankWordTokenizer()
# ->
tokenizer.tokenize('Hello World.')
# ->
word_tokenize("can't")
# ->
from nltk.tokenize import WordPunctTokenizer
# ->
tokenizer = WordPunctTokenizer()
# ->
tokenizer.tokenize("Can't is a contraction.")
# ->
from nltk.tokenize import RegexpTokenizer
# ->
tokenizer = RegexpTokenizer("[\w']+")
# ->
tokenizer.tokenize("Can't is a contraction.")
# ->
from nltk.tokenize import regexp_tokenize
# ->
regexp_tokenize("Can't is a contraction.", "[\w']+")
# ->
tokenizer = RegexpTokenizer('\s+', gaps = True)
# ->
tokenizer.tokenize("Can't is a contraction.")
### Treinando um Tokenizer
# ->
from nltk.tokenize import PunktSentenceTokenizer
from nltk.corpus import webtext
# ->
# /Users/dmpm/nltk_data/corpora/webtext
texto = webtext.raw('overheard.txt')
# ->
sent_tokenizer = PunktSentenceTokenizer(texto)
# ->
sents1 = sent_tokenizer.tokenize(texto)
# ->
sents1[0]
# ->
from nltk.tokenize import sent_tokenize
# ->
sents2 = sent_tokenize(texto)
# ->
sents2[0]
# ->
sents1[678]
# ->
sents2[678]
# ->
```

```
# Inserindo caminho em sistema Windows
with open('/Users/dmpm/nltk_data/corpora/webtext/overheard.txt', encoding = 'ISO-8859-2') as f:
    texto = f.read()
# Path para o Windows
# with open('C:/Users/usuario/AppData/Roaming/nltk_data/corpora/webtext/overheard.txt', encoding =
'ISO-8859-2') as f:
#    texto = f.read()
# ->
sent_tokenizer = PunktSentenceTokenizer(texto)
# ->
sents = sent_tokenizer.tokenize(texto)
# ->
sents[0]
# ->
sents[678]
## Stopwords
Stopwords são palavras comuns que normalmente não contribuem para o significado de uma frase, pelo
menos com relação ao propósito da informação e do processamento da linguagem natural. São palavras
como "The" e "a" ((em inglês) ou "O/A" e "Um/Uma" ((em português). Muitos mecanismos de busca
filtram estas palavras (stopwords), como forma de economizar espaço em seus índices de pesquisa.
# ->
from nltk.corpus import stopwords
# ->
english_stops = set(stopwords.words('english'))
# ->
words = ["Can't", 'is', 'a', 'contraction']
# ->
[word for word in words if word not in english_stops]
# ->
portuguese_stops = set(stopwords.words('portuguese'))
# ->
palavras = ["Aquilo", 'é', 'um', 'gato']
# ->
[palavra for palavra in palavras if palavra not in portuguese_stops]
# ->
stopwords.fileids()
# ->
stopwords.words('portuguese')
### Wordnet
WordNet é um banco de dados léxico (em Inglês). É uma espécie de dicionário criado especificamente
para processamento de linguagem natural.
# ->
from nltk.corpus import wordnet
# ->
syn = wordnet.synsets('cookbook')[0]
# ->
syn.name()
# ->
syn.definition()
# ->
wordnet.synsets('cooking')[0].examples()
## Collocations
Collocations são duas ou mais palavras que tendem a aparecer frequentemente juntas, como "Estados
Unidos" ou "Rio Grande do Sul". Essas palavras podem gerar diversas combinações e por isso o
contexto também é importante no processamento de linguagem natural.
# ->
from nltk.corpus import webtext
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
# ->
words = [w.lower() for w in webtext.words('grail.txt')]
# ->
bcf = BigramCollocationFinder.from_words(words)
# ->
bcf.nbest(BigramAssocMeasures.likelihood_ratio, 4)
# ->
from nltk.corpus import stopwords
# ->
stopset = set(stopwords.words('english'))
# ->
filter_stops = lambda w: len(w) < 3 or w in stopset
# ->
bcf.apply_word_filter(filter_stops)
# ->
bcf.nbest(BigramAssocMeasures.likelihood_ratio, 4)
## Stemming Words
Stemming é a técnica de remover sufixos e prefixos de uma palavra, chamada stem. Por exemplo, o
stem da palavra cooking é cook. Um bom algoritmo sabe que "ing" é um sufixo e pode ser removido.
Stemming é muito usado em mecanismos de buscas para indexação de palavras. Ao invés de armazenar
```

```
todas as formas de uma palavras, um mecamismo de busca armazena apenas o stem da palavra, reduzindo
o tamanho do índice e aumentando a performance do processo de busca.
# ->
from nltk.stem import PorterStemmer
# ->
stemmer = PorterStemmer()
# ->
stemmer.stem('cooking')
# ->
stemmer.stem('cookery')
# ->
from nltk.stem import LancasterStemmer
# ->
stemmer = LancasterStemmer()
# ->
stemmer.stem('cooking')
# ->
stemmer.stem('cookery')
# ->
from nltk.stem import RegexpStemmer
# ->
stemmer = RegexpStemmer('ing')
# ->
stemmer.stem('cooking')
# ->
from nltk.stem import SnowballStemmer
# ->
SnowballStemmer.languages
# ->
spanish_stemmer = SnowballStemmer('portuguese')
# ->
spanish_stemmer.stem('Tudo bem')
### Corpus
Corpus é uma coleção de documentos de texto e Corpora é o plural de Corpus. Esse termo vem da
palavra em Latim para corpo (nesse caso, o corpo de um texto). Um Corpus customizado é uma coleção
de arquivos de texto organizados em um diretório.
Se você for treinar seu próprio modelo como parte de um processo de classificação de texto (como
análise de texto), você ter;a que criar seu próprio Corpus e treiná-lo.
# ->
from nltk.corpus.reader import WordListCorpusReader
# ->
# Criando um Corpus (arquivo palavras.txt no mesmo diretório do Jupyter Notebook)
reader = WordListCorpusReader('.', ['palavras.txt'])
# ->
reader.words()
# ->
reader.fileids()
# ->
reader.raw()
# ->
from nltk.tokenize import line_tokenize
# ->
line_tokenize(reader.raw())
# ->
from nltk.corpus import brown
# ->
brown.categories()
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Quizz

O Apache Spark Streaming é um sistema de processamento de streaming, tolerante a falhas e escalável, o que significa que rapidamente podemos aumentar a quantidade de nodes em um cluster e assim expandir sua capacidade.

O Spark Streaming funciona com o conceito de microbatching para simular análise de fluxo em tempo real. Isto significa que um programa em batch é executado em intervalos frequentes para processar todos os dados ao longo do streaming. Embora esta abordagem seja inadequada para aplicações de baixa latência (aquelas que realmente requerem "real" real-time), é uma maneira

inteligente de utilizar pequenos processos em lote (microbatching) para se aproximar de uma atividade em tempo real e funciona bem para muitas situações.

Um DStream é uma sequência de dados que são coletados ao longo do tempo e que podem ser processados enquanto estiverem sendo coletados.

Os DStreams oferecem muitas das operações que podem ser realizadas com os RDD’s, mais as operações relacionadas ao tempo, como sliding windows.

Cada RDD dentro de um DStream armazena dados para cada unidade de tempo e podemos processar isso em paralelo e gerar outputs para cada unidade de tempo também, ou seja, gerar output para cada RDD. Podemos fazer isso em nível de RDD ou no nível do Dstream, aplicando operações a todo o streaming de dados.

5.10. Análise de Sentimentos do Twitter em Tempo Real com Spark Streaming e NLTK

Criando Aplicação no Twitter

1. Crie uma conta no twitter
2. Em seguida, acesse http://developer.twitter.com/
3. Peça a permissão de desenvolvedor
4. Após aprovação do pedido, crie o App
    1. Apps > Create App
    2. Dê um nome ao app
    3. Dê uma descrição ao app
    4. Coloque um site da sua empresa ou pessoal no campo URL
    5. Em Terms of Service coloque o link dos termos de uso do site
    6. Inclua também o link da política de privacidade
    7. Preencha o link do site da organização
    8. Descreva o funcionamento da aplicação
    9. Clique em Create
5. Após criação do App, clique em Keys and tokens
    1. Copie Consumer API Keys (2 chaves) e Access token & access token secret (2 chaves)
6. Utilize essas chaves no jupyter notebook.

Gerando um Streaming de Dados em Tempo Real e Coletando com Spark Streaming

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 10</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 ou 11 e Apache Spark 2.4.2
****** Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o
diretório metastore_db no mesmo diretório onde está este Jupyter notebook ******
Acesse http://localhost:4040 sempre que quiser acompanhar a execução dos jobs
## Spark Streaming
## Simulando um Streaming de Dados TCP/IP na porta 9898 do seu computador local
Windows: http://mobaxterm.mobatek.net/ . Clique em Start Local Terminal e depois: nc -lk 9898
Mac/Linux: nc -lk 9898
# ->
# Importando o StreamingContext
from pyspark.streaming import StreamingContext
# ->
# Criando um StreamingContext com intervalo batch de 1 segundo
ssc = StreamingContext(sc, 1)
# ->
# Criando um DStream que vai conectar na porta 9898 da sua máquina local
linhas = ssc.socketTextStream("localhost", 9898)
# ->
```

```
type(linhas) # pyspark.streaming.dstream.DStream
# ->
# Divide cada linha em palavras
palavras = linhas.flatMap(lambda line: line.split(" "))
# ->
type(palavras) # pyspark.streaming.dstream.TransformedDStream
# ->
# Conta cada palavra em cada batch
pares = palavras.map(lambda palavra: (palavra, 1)) # Big Data Big Data -- (Big, 1) (Big, 1)
contaPalavras = pares.reduceByKey(lambda x, y: x + y) # (Big, 2)
# ->
# Imprime os 10 primeiros elementos de cada RDD gerado no DStream
contaPalavras.pprint()
# Execute o comando nc -lk 9898 no terminal
# ->
ssc.start()             # Inicia a coleta e processamento do stream de dados
ssc.awaitTermination()  # Aguarda a computação ser finalizada
# Digite o que quiser no terminal (para que o codigo recupere e conte as palavras)
# Dê stop no jupyter notebook e depois execute o comando abaixo
# ->
ssc.stop()
# Digite ctrl+c no terminal
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Análise de Sentimentos de Streaming de Dados do Twitter em Tempo Real

Execute o código abaixo com pyspark.

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 10</font>
### *********** Atenção: ***********
Utilize Java JDK 1.8 ou 11 e Apache Spark 2.4.2
****** Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o
diretório metastore_db no mesmo diretório onde está este Jupyter notebook ******
Acesse http://localhost:4040 sempre que quiser acompanhar a execução dos jobs
# ->
O Twitter é uma das redes sociais mais dinâmicas disponíveis atualmente. É possível coletar em
tempo real informações preciosas
e o sentimento das pessoas sobre os mais variados temas. São milhões de tweets por minuto em todo
mundo e muita informação
preciosa está escondida em cada tweet.
O  Streaming  de  dados  gerado  pelo  Twitter  pode  alimentar  aplicações analíticas, permitindo
que  empresas  compreendam,
em  tempo  real,  o  que clientes, parceiros e fornecedores estão pensando (e escrevendo) sobre
você,sua marca, produto ou
serviço.
Neste  projeto,  coletamos  dados  de  Streaming  do  Twitter, aplicamostécnicas  de análise  em
tempo  real
(à  medida  que  os  dados  são  gerados) e obtemosinsights sobre um determinado assunto. O projeto
pode ser facilmente
reproduzido localmente em seu computador ou em um ambiente de clusterna nuvem.
Tecnologias utilizadas:
    Python/NLTK -construção do modelo de análise de sentimento
    Spark Streaming -coleta do streaming de dados
    API do Twitter -conexão a partir da nossa aplicação
## Spark Streaming - Twitter
# ->
# Pode ser necessário instalar esses pacotes
# !pip install requests_oauthlib
# !pip install twython
# !pip install nltk
# ->
# Módulos usados
from pyspark.streaming import StreamingContext
from pyspark import SparkContext
from requests_oauthlib import OAuth1Session
from operator import add
import requests_oauthlib
from time import gmtime, strftime
import requests
import time
```

```
import string
import ast
import json
# ->
# Pacote NLTK
import nltk
from nltk.classify import NaiveBayesClassifier
from nltk.sentiment import SentimentAnalyzer
from nltk.corpus import subjectivity
from nltk.corpus import stopwords
from nltk.sentiment.util import *
# ->
# Frequência de update
INTERVALO_BATCH = 5
# ->
# Criando o StreamingContext
ssc = StreamingContext(sc, INTERVALO_BATCH)
## Treinando o Classificador de Análise de Sentimento
Uma parte essencial da criação de um algoritmo de análise de sentimento (ou qualquer algoritmo de
mineração de dados) é ter um conjunto de dados abrangente ou "Corpus" para o aprendizado, bem como
um conjunto de dados de teste para garantir que a precisão do seu algoritmo atende aos padrões que
você espera. Isso também permitirá que você ajuste o seu algoritmo a fim de deduzir melhores (ou
mais precisas) características de linguagem natural que você poderia extrair do texto e que vão
contribuir para a classificação de sentimento, em vez de usar uma abordagem genérica. Tomaremos
como base o dataset de treino fornecido pela Universidade de Michigan, para competições do Kaggle
--> https://inclass.kaggle.com/c/si650winter11.
Esse dataset contém 1,578,627 tweets classificados e cada linha é marcada como:
### 1 para o sentimento positivo
### 0 para o sentimento negativo
# ->
# Lendo o arquivo texto e criando um RDD em memória com Spark
arquivo = sc.textFile("/Users/dmpm/Dropbox/DSA/BigDataAnalytics-Python-Spark2.0/Cap10/
dataset_analise_sentimento.csv")
# ->
# Removendo o cabeçalho
header = arquivo.take(1)[0]
dataset = arquivo.filter(lambda line: line != header)
# ->
type(dataset) # pyspark.rdd.PipelinedRDD
# ->
# Essa função separa as colunas em cada linha, cria uma tupla e remove a pontuação.
def get_row(line):
  row = line.split(',')
  sentimento = row[1]
  tweet = row[3].strip()
  translator = str.maketrans({key: None for key in string.punctuation})
  tweet = tweet.translate(translator)
  tweet = tweet.split(' ')
  tweet_lower = []
  for word in tweet:
    tweet_lower.append(word.lower())
  return (tweet_lower, sentimento)
# ->
# Aplica a função a cada linha do dataset
dataset_treino = dataset.map(lambda line: get_row(line))
# ->
# Cria um objeto SentimentAnalyzer
sentiment_analyzer = SentimentAnalyzer()
# ->
# Certifique-se de ter espaço em disco - Aproximadamente 5GB
# https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
# nltk.download()
nltk.download("stopwords")
# ->
from IPython.display import Image
Image(url = 'ntlkdata.png')
# ->
# Obtém a lista de stopwords em Inglês
stopwords_all = []
for word in stopwords.words('english'):
  stopwords_all.append(word)
  stopwords_all.append(word + '_NEG')
# ->
# Obtém 10.000 tweets do dataset de treino e retorna todas as palavras que não são stopwords
dataset_treino_amostra = dataset_treino.take(10000)
# ->
all_words_neg = sentiment_analyzer.all_words([mark_negation(doc) for doc in
dataset_treino_amostra])
all_words_neg_nostops = [x for x in all_words_neg if x not in stopwords_all]
```

```
# ->
# Cria um unigram (n-grama) e extrai as features
unigram_feats = sentiment_analyzer.unigram_word_feats(all_words_neg_nostops, top_n = 200)
sentiment_analyzer.add_feat_extractor(extract_unigram_feats, unigrams = unigram_feats)
training_set = sentiment_analyzer.apply_features(dataset_treino_amostra)
# ->
type(training_set) # nltk.collections.LazyMap
# ->
print(training_set)
# ->
# Treinar o modelo
trainer = NaiveBayesClassifier.train
classifier = sentiment_analyzer.train(trainer, training_set)
# ->
# Testa o classificador em algumas sentenças
test_sentence1 = [(['this', 'program', 'is', 'bad'], '')]
test_sentence2 = [(['tough', 'day', 'at', 'work', 'today'], '')]
test_sentence3 = [(['good', 'wonderful', 'amazing', 'awesome'], '')]
test_set = sentiment_analyzer.apply_features(test_sentence1)
test_set2 = sentiment_analyzer.apply_features(test_sentence2)
test_set3 = sentiment_analyzer.apply_features(test_sentence3)
# ->
# Autenticação do Twitter
consumer_key = "xxx"
consumer_secret = "xxx"
access_token = "xxx"
access_token_secret = "xxx"
# ->
# Especifica a URL termo de busca
search_term = 'Trump'
sample_url = 'https://stream.twitter.com/1.1/statuses/sample.json'
filter_url = 'https://stream.twitter.com/1.1/statuses/filter.json?track='+search_term
# ->
# Criando o objeto de atutenticação para o Twitter
auth = requests_oauthlib.OAuth1(consumer_key, consumer_secret, access_token, access_token_secret)
# ->
# Configurando o Stream
rdd = ssc.sparkContext.parallelize([0])
stream = ssc.queueStream([], default = rdd)
# ->
type(stream) # pyspark.streaming.dstream.DStream
# ->
# Total de tweets por update
NUM_TWEETS = 500
# ->
# Essa função conecta ao Twitter e retorna um número específico de Tweets (NUM_TWEETS)
def tfunc(t, rdd):
  return rdd.flatMap(lambda x: stream_twitter_data())
def stream_twitter_data():
  response = requests.get(filter_url, auth = auth, stream = True)
  print(filter_url, response)
  count = 0
  for line in response.iter_lines():
    try:
      if count > NUM_TWEETS:
        break
      post = json.loads(line.decode('utf-8'))
      contents = [post['text']]
      count += 1
      yield str(contents)
    except:
      result = False
# ->
stream = stream.transform(tfunc)
# ->
coord_stream = stream.map(lambda line: ast.literal_eval(line))
# ->
# Essa função classifica os tweets, aplicando as features do modelo criado anteriormente
def classifica_tweet(tweet):
  sentence = [(tweet, '')]
  test_set = sentiment_analyzer.apply_features(sentence)
  print(tweet, classifier.classify(test_set[0][0]))
  return(tweet, classifier.classify(test_set[0][0]))
# ->
# Essa função retorna o texto do Twitter
def get_tweet_text(rdd):
  for line in rdd:
    tweet = line.strip()
    translator = str.maketrans({key: None for key in string.punctuation})
```

```
    tweet = tweet.translate(translator)
    tweet = tweet.split(' ')
    tweet_lower = []
    for word in tweet:
      tweet_lower.append(word.lower())
    return(classifica_tweet(tweet_lower))
# ->
# Cria uma lista vazia para os resultados
resultados = []
# ->
# Essa função salva o resultado dos batches de Tweets junto com o timestamp
def output_rdd(rdd):
  global resultados
  pairs = rdd.map(lambda x: (get_tweet_text(x)[1],1))
  counts = pairs.reduceByKey(add)
  output = []
  for count in counts.collect():
    output.append(count)
  result = [time.strftime("%I:%M:%S"), output]
  resultados.append(result)
  print(result)
# ->
# A função foreachRDD() aplica uma função a cada RDD to streaming de dados
coord_stream.foreachRDD(lambda t, rdd: output_rdd(rdd))
# ->
# Start streaming
ssc.start()
# ssc.awaitTermination()
# ->
cont = True
while cont:
  if len(resultados) > 5:
    cont = False
# ->
# Grava os resultados
rdd_save = '/Users/dmpm/Dropbox/DSA/BigDataAnalytics-Python-Spark2.0/Cap10/r'+time.strftime("%I%M
%S")
resultados_rdd = sc.parallelize(resultados)
resultados_rdd.saveAsTextFile(rdd_save)
# ->
# Visualiza os resultados
resultados_rdd.collect()
# ->
# Finaliza o streaming
ssc.stop()
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Visualização de Dados e Apresentação do Resultado com D3.js

Execute o código abaixo com pyspark.

```
<!DOCTYPE html>
<html lang="en">
       <head>
              <meta charset="utf-8">
              <title>Visualizando Análise de Sentimentos em Tempo Real</title>
              <script type="text/javascript" src="./d3/d3.js"></script>
              <style type="text/css">
              </style>
       </head>
       <body>
              <style>

              </style>
              <p style="font-size:20px"><b>Análise de Sentimentos (Positivo e Negativo) de Tweets
sobre Donald 'Trump'</b></p>
              <script src="https://d3js.org/d3.v3.min.js" charset="utf-8"></script>
              <script type="text/javascript">

                     // Definindo largura e altura
                     var width = 1250;
                     var height = 650;
```

```
                        var pad = 50;

                        //add the svg
                        var svg = d3.select("body")
                                                .append("svg")
                                                .attr("width", width)
                                                .attr("height", height);

                        // Dados obtidos do Streaming
                        var trump = [
                                [0, 1, 14, 62],
                                [1, 1, 17, 67],
                                [2, 1, 5, 35],
                                [3, 1, 12, 73],
                                [4, 1, 22, 102],
                                [5, 1, 17, 72],
                                [6, 1, 13, 60],
                                [7, 1, 11, 50],
                                [0, 0, 48, 62],
                                [1, 0, 50, 67],
                                [2, 0, 30, 35],
                                [3, 0, 61, 73],
                                [4, 0, 80, 102],
                                [5, 0, 55, 72],
                                [6, 0, 47, 60],
                                [7, 0, 39, 50]
                        ];

                        // Escalas de x e y
                        var xScale = d3.scale.linear().domain([0, 10]).range([pad, width-2*pad]);
                        var yScale = d3.scale.linear().domain([100, 0]).range([pad ,height-pad]);

                        var xAxis = d3.svg.axis().scale(xScale).orient("bottom").tickFormat(function
(d) {
                                return xScale.tickFormat(4,d3.format(",d"))(d)}).ticks(3);
                        var yAxis = d3.svg.axis().scale(yScale).orient("left");

                        // Mapeamento de x e y para suas respectivas escalas
                        var xMap = function(d) { return xScale(d);}
                        var yMap = function(d) { return yScale(d);}

                        // Definindo o eixo x
                        svg.append("g")
                                .attr("class", "axis")
                                .attr("transform", "translate(0, " + (height-pad) + ")")
                                .call(xAxis);

                        // Definindo o eixo y
                        svg.append("g")
                                .attr("class", "axis")
                                .attr("transform", "translate(" + pad + ", 0)")
                                .call(yAxis);

                        var getPercentage = function(d) {

                                return d[2]/d[3]*100;
                        }

                        // Adiconando cada parte dos dados
                        svg.selectAll("circle")
                           .data(trump)
                           .enter()
                           .append("circle")
                           .attr("r", 7)

                           // Obtendo as coordenadas de x e y
                           .attr("cx", function(d) {return xMap(d[0]);})
                           .attr("cy", function(d) {return yMap(getPercentage(d));})

                           // Azul para os pontos reais e rosa para as previsões
                           .style("fill", function(d) {
                                        color = "";
                                        if (d[1] == 1)
                                        {
                                                color = "green";
                                        }
                                        else
                                        {
                                                color = "red";
```

```
                        }
                        return color;
                        })
            .attr("stroke", "black")

            // Alterando a opacidade com Mouseover
            .on("mouseover", function(d) {
          d3.select(this).style("opacity", .5);
                })
            .on("mouseout", function(d)
             {
                        d3.select(this).style("opacity", 1);
             })
            .append("title")
            .text(function(d) {
                        type = "Negative";
                        if (d[1] == 1)
                        {
                               type = "Positive";
                        }
                        return type + "(" + d[2] + " tweets)";})
                ;

       // Label de x
       svg.append("text")
               .attr("class", "x label")
               .attr("text-anchor", "start")
               .attr("x", pad)
               .attr("y", height - 6)
               .text("Intervalo de Tempo (5 segundos)");

       // Label de y
       svg.append("text")
               .attr("class", "y label")
               .attr("text-anchor", "start")
               .attr("x", pad-5)
               .attr("y", pad-10)
               .text("Percentual de Tweets (Positivo/Negativo) Contendo 'Trump'");



       var trumpPos = [
               [0, 1, 14, 62],
               [1, 1, 17, 67],
               [2, 1, 5, 35],
               [3, 1, 12, 73],
               [4, 1, 22, 102],
               [5, 1, 17, 72],
               [6, 1, 13, 60],
               [7, 1, 11, 50]
               ]
       var trumpNeg =        [
               [0, 0, 48, 62],
               [1, 0, 50, 67],
               [2, 0, 30, 35],
               [3, 0, 61, 73],
               [4, 0, 80, 102],
               [5, 0, 55, 72],
               [6, 0, 47, 60],
               [7, 0, 39, 50]
               ]

       var lineFunction = d3.svg.line()
.x(function(d) { return xMap(d[0]);})
.y(function(d) { return yMap(getPercentage(d));})
.interpolate("linear");

       svg.append("path")
               .attr("d", function(d) {
                               return lineFunction(trumpPos);

                        })
.attr("stroke", "green")
.attr("stroke-width", 2)
.attr("fill", "none");

       var lineFunction = d3.svg.line()
.x(function(d) { return xMap(d[0]);})
.y(function(d) { return yMap(getPercentage(d));})
```

```
                .interpolate("linear");

                      svg.append("path")
                              .attr("d", function(d) {
                                              return lineFunction(trumpNeg);

                                      })
                .attr("stroke", "red")
                .attr("stroke-width", 2)
                .attr("fill", "none");

              </script>
        </body>
</html>
```

5.11. Apache Spark Machine Learning

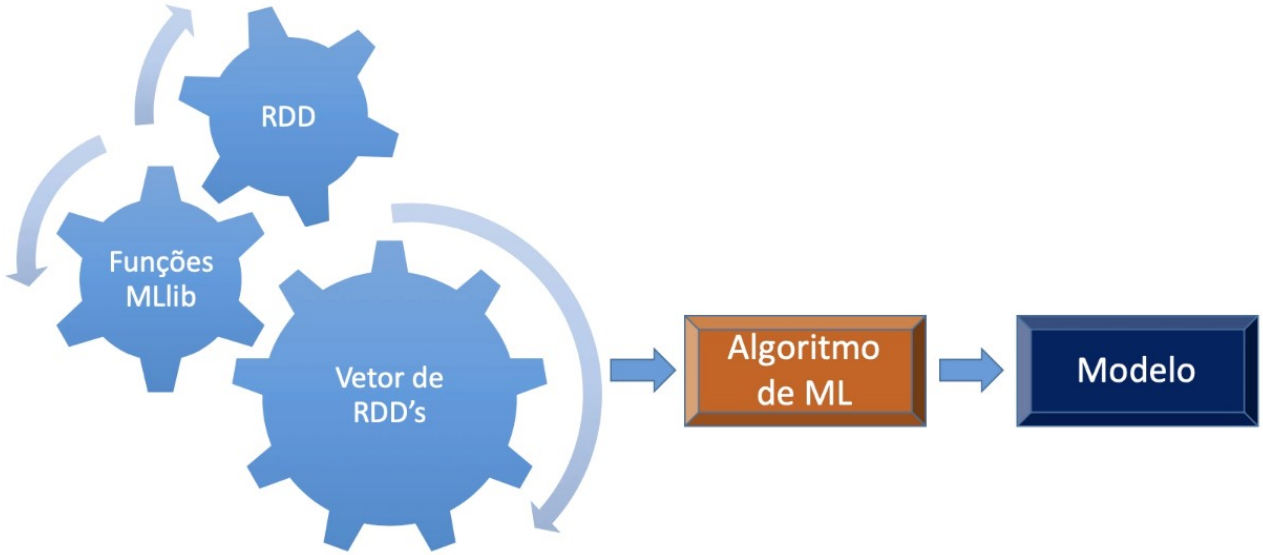
Introdução

O Apache Spark traz 2 bibliotecas para se trabalhar com Machine Learning:

- ML → Para dataframes
- Mllib → Para RDDs

Machine Learning com Apache Spark

Modelo de Machine Learning → RDD



Spark Mllib contém fuções que implementam ML e foi criado para ser executado em paralelo, através de um cluster.

Scikit Learn x Spark Mllib

- Scikit Learn tem foco em execução em uma única máquina
- Spark MLlib tem foco em execução em ambientes distribuídos

Função parallelize() → Define em quais nodes (ou máquinas de um cluster) se realizará a execução do modelo de ML.

NumPy → O MLlib precisa do NumPy para ser executado. Não é necessário uma chamada explícita ao NumPy, mas ele precisa estar instalado na máquina onde se executa o MLlib.

Analytics e Dataficação

Dataficação (datafication) é o processo pelo qual estamos passando, onde temos dados sobre absolutamente tudo.

Analytics é o processo de coletar dados e gerar insights para tomadas de decisões baseadas em fatos.

- Big Data + Analytics = Big Data Analytics
  Análise em Tempo Real + Analytics = Real-Time Analytics

Podemos hoje analisar grandes conjuntos de dados ou dados gerados em tempo real e coletar insights que não podiam ser coletados há pouco tempo atrás.

Tipos de Analytics

Principais Tipos de Analytics (e seus objetivos):

- Descritiva: Compreender o que aconteceu
- Exploratória: Descobrir porque alguma coisa aconteceu
- Inferencial: Compreender uma população a partir de uma amostra
- Preditiva: Prever o que vai acontecer
- Causal: O que ocorre com uma variável quando outra é alterada
- Deep: Técnicas avançadas para compreender grandes conjuntos de dados de diversas fontes

Análise Exploratória e Análise Preditiva: ambos são a essência do que é feito em ML

Análise Exploratória de Dados

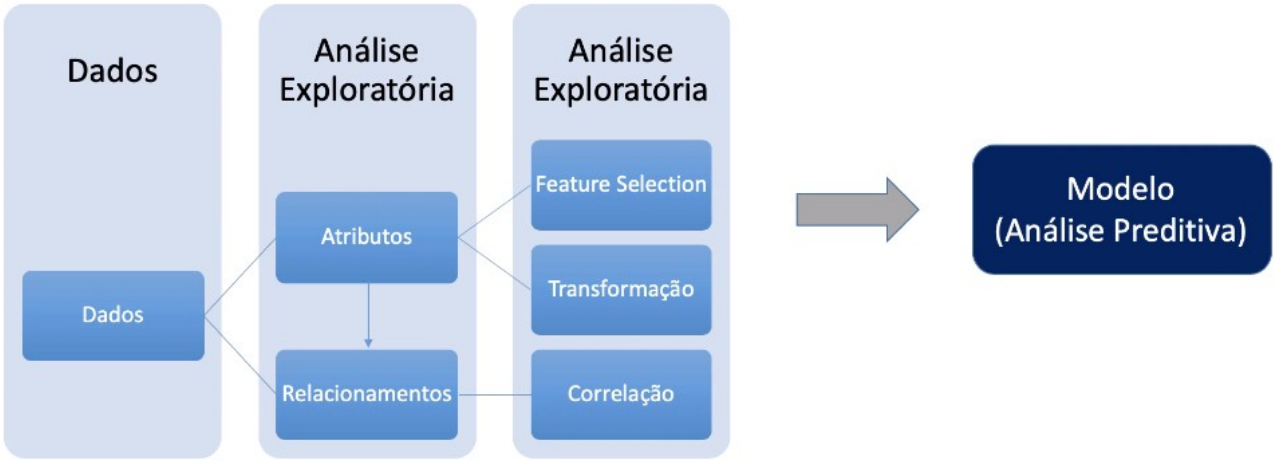
Atividades Realizadas na Análise Exploratória de Dados:

- Compreender variáveis preditoras e target no dataset
- Validar o processo de coleta dos dados
- Descobrir padrões e tendências
- Encontrar variáveis chaves
- Detectar outliers
- Testar hipóteses
- Descobrir a correlação entre as variáveis
- Eliminar variáveis irrelevantes

Ferramentas usadas na Análise Exploratória de Dados:

- Matriz de Correlação
- Histogramas
- Scatterplots
- Boxplots
- Principal Component Analysis

Análise Preditiva



As 2 principais subáreas de ML: Aprendizagem Supervisionada x Aprendizagem Não Supervisionada.

Aprendizagem Supervisionada:

- Tenta fazer previsões a partir do treinamento com dados de entrada e dados de saída.
- Os modelos são construídos em datasets de treino.
- Os modelos são usados para prever o futuro.
- Pode ser: Regressão (dados numéricos e contínuos) ou Classificação (classes)
- Dados históricos contém variáveis preditoras e a variável alvo (target).
- O conjunto de dados é separado em dados de treino e dados de teste.
- Dados de treino são usados para treinar o modelo.
- Dados de teste são usados para testar e validar o modelo.
- Utilizamos uma métrica (como acurácia) para avaliar o modelo.
- Split 70/30 (treino/teste).
- Seleção aleatória dos dados em ambos os datasets.

Aprendizagem Não Supervisionada:

- Busca estrutura ou similaridade oculta nos dados.
- Grupos observados baseados em similaridade entre as entidades.
- Similaridade entre as entidades pode ser: distância entre os valores, presença/ausência de atributos.
- Pode ser: Clustering, Regras de Associação ou Filtros Colaborativos (Sistemas de Recomendação)

Trade-off Entre Viés e Variância

Bias e Variance Trade-off (Viés e Variância)

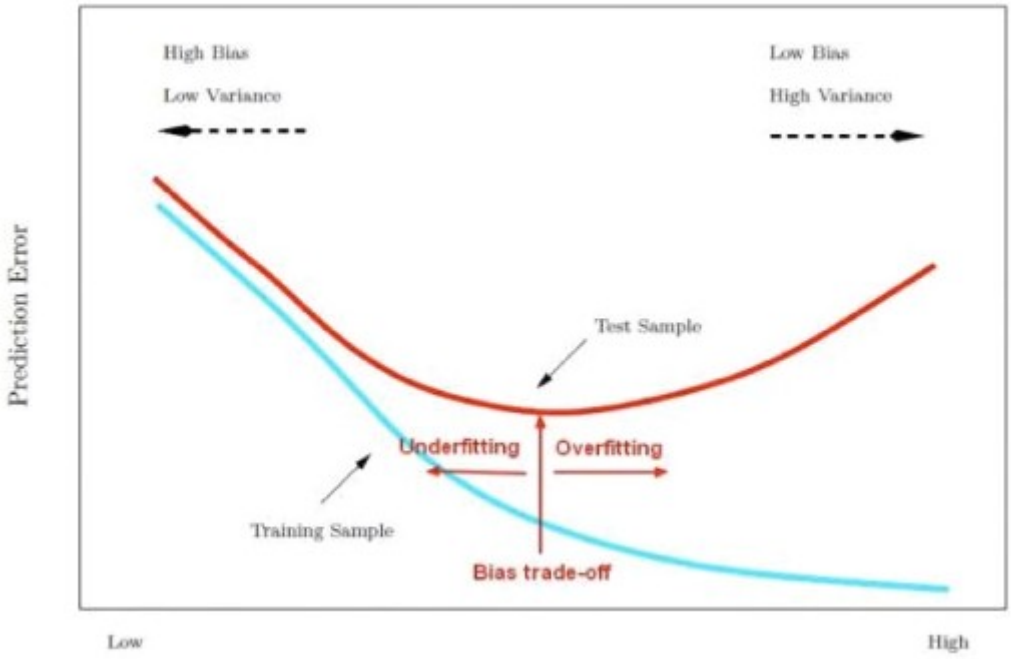
É comum, ao construirmos e escolhermos parâmetros para um modelo, nos depararmos com a seguinte questão: como reduzir o erro do modelo? Para respondermos essa pergunta de maneira correta, em primeiro lugar, devemos entender os 2 principais componentes do erro em nossas predições: bias e variance.

Bias (Viés): É a diferença entre o valor esperado da predição do nosso modelo (média das predições) e o valor real que queremos predizer (dados históricos).

Variância: É a variabilidade das predições.

De forma resumida: o bias está relacionado à habilidade do modelo em se ajustar aos dados, ou seja, se o seu problema é um underfitting, o seu modelo tem um alto bias. Já a variância está relacionada à habilidade do modelo se ajustar a novos dados, ou seja, se o seu problema é um overfitting, o seu modelo tem uma alta variância.

O nosso objetivo é reduzir o bias e a variância o máximo que pudermos, entretanto, nos deparamos com um trade-off entre underfitting e overfitting.



Spark Machine Learning Library

APIs de Machine Learning do Apache Spark

Documentação: https://spark.apache.org/

Apache Spark MLLib (Machine Learning Library):

- spark.mllib → API original construída para trabalhar com RDD's.
- spark.ml → Nova API construída para funcionar também com Dataframes e SparkSQL.

Tipos de Dados:

- org.apache.spark.mllib (Java/Scala)
- pyspark.mllib (Python)

Tipo de Dado	**Pacote**
Vetor	mllib.linalg.Vectors
LabeledPoint	mllib.regression
Rating	mllib.recommendation

Tipos de Vetores:

- Vetor Denso

  (2.0, 4.0, 8.5)

- Vetor Esparso

  Original (1.0, 0.0, 0.0, 2.0, 0.0)

  Representação (5, (0,3), (1.0, 2.0))

Pipelines: Pipeline consiste de uma série de transformações e ações que precisam ser realizadas para criar um modelo.

Mllib – Outras Funcionalidades:

Funcionalidade (Feature Extraction)	**Funções (importadas a partir do pacote mllib.feature)**
TF-IDF (Term Frequency – Inverse Document Frequency)	HashingTF() e IDF ()
Escala	StandardScaler()
Normalização	Normalizer()
Word2Vec	Word2Vec()
Estatística	colStats(), corr(), chiSqTest(), mean(), stdev(), sample()

Spark MLLib – Regressão Linear – Compreendendo o Problema de Negócio e Carregando os Dados /
Spark MLLib – Regressão Linear – Limpeza do Dataset e Remoção de Valores Missing /
Spark MLLib – Regressão Linear – Análise Exploratória e Análise de Correlação /
Spark MLLib – Regressão Linear – Pré-processamento de Dados, Vetores Densos e Esparsos /
Spark MLLib – Regressão Linear – Criação, Treinamento e Avaliação do Modelo de Machine Learning

Execute o código abaixo via pyspark (comando dado no terminal, dentro do diretório de trabalho).

```
 # <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 11</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 ou 11 e Apache Spark 2.4.2
****** Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook ******
## <font color='blue'>Spark MLLib - Regressão Linear</font>
<strong> Descrição </strong>
<ul style="list-style-type:square">
  <li>Método para avaliar o relacionamento entre variáveis.</li>
  <li>Estima o valor de uma variável dependente a partir dos valores das variáveis independentes.</li>
  <li>Usado quando as variáveis dependente e independente são contínuas e possuem alguma correlação.</li>
  <li>O R-Square mede quão perto os dados estão da linha de regressão. O valor do R-Squared será entre 0 e 1, sendo que quanto maior o valor, melhor.</li>
  <li>Os dados de entrada e de saída são usados na construção do modelo. A equação linear retorna os valores dos coeficientes.</li>
  <li>A equação linear representa o modelo.</li>
</ul>
<dl>
  <dt>Vantagens</dt>
  <dd>- Baixo custo</dd>
  <dd>- Veloz</dd>
  <dd>- Excelente para relações lineares</dd>
  <br />
  <dt>Desvantagens</dt>
  <dd>- Somente variáveis numéricas</dd>
  <dd>- Sensível a outliers</dd>
  <br />
  <dt>Aplicação</dt>
  <dd>- Um dos modelos mais antigos e pode ser usado para resolver diversos problemas</dd>
</dl>
## Usaremos Regressão Linear para prever os valores de MPG (Miles Per Gallon)
MPG será a variável target e as demais variáveis serão as features (variáveis preditoras).
# ->
# Imports
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
# ->
# Spark Session - usada quando se trabalha com Dataframes no Spark
spSession = SparkSession.builder.master("local").appName("DSA-SparkMLLib").getOrCreate()
# ->
# Carregando os dados e gerando um RDD
carrosRDD = sc.textFile("data/carros.csv")
# ->
# Colocando o RDD em cache. Esse processo otimiza a performance.
carrosRDD.cache()
# ->
carrosRDD.count() # conta as linhas ou registros
# ->
carrosRDD.take(5) # traz as 5 primeiras linhas
# ->
# Removendo a primeira linha do arquivo (cabeçalho)
carrosRDD2 = carrosRDD.filter(lambda x: "DISPLACEMENT" not in x) # transformação
carrosRDD2.count() # ação
## Limpeza dos Dados
```

```
# ->
carrosRDD2.take(398) # agora sem o cabeçalho
# ->
# Usando um valor padrão para average HP (que será usado para preencher os valores missing)
mediaHP = sc.broadcast(75.0)
# ->
# Função para limpeza dos dados
def limpaDados(inputStr) :
    global mediaHP
    attList = inputStr.split(",")
    # Substitui o caracter ? (missing do dataset) por um valor
    hpValue = attList[3]
    if hpValue == "?":
        hpValue = mediaHP.value
    # Cria uma linha usando a função Row, limpando e convertendo os dados de string para float
    linhas = Row(MPG = float(attList[0]), CYLINDERS = float(attList[1]), DISPLACEMENT = float(attList[2]),
                 HORSEPOWER = float(hpValue), WEIGHT = float(attList[4]), ACCELERATION = float(attList[5]),
                 MODELYEAR = float(attList[6]), NAME = attList[7])
    return linhas
# ->
# Executa a função no RDD
carrosRDD3 = carrosRDD2.map(limpaDados)
carrosRDD3.cache()
carrosRDD3.take(5)
## Análise Exploratória de Dados
# ->
# Cria um Dataframe
carrosDF = spSession.createDataFrame(carrosRDD3)
# ->
# Estatísticas descritivas (select possível graças a transformação do RDD em dataframe)
carrosDF.select("MPG","CYLINDERS").describe().show()
# ->
# Encontrando a correlação entre a variável target com as variáveis preditoras
for i in carrosDF.columns:
    if not(isinstance(carrosDF.select(i).take(1)[0][0], str)) :
        print("Correlação da variável MPG com", i, carrosDF.stat.corr('MPG', i))
## Pré-Processamento dos Dados
# ->
from IPython.display import Image
Image("imagens/vetores.png")
Vetores esparsos são vetores que tem muitos valores como zero. Enquanto um vetor denso é quando a maioria dos valores no vetor são diferentes de zero.
Conceitualmente são o mesmo objeto. Apenas um vetor. Normalmente, o vetor esparso é representado por uma tupla (id, valor).
Por exemplo, um vetor denso (1, 2, 0, 0, 5, 0, 9, 0, 0) seria representado como vetor esparso assim: {(0,1,4,6), (1, 2, 5, 9)}
# ->
# Convertendo para um LabeledPoint (target, Vector[features])
# Remove colunas não relevantes para o modelo ou com baixa correlação
def transformaVar(row) :
    obj = (row["MPG"], Vectors.dense([row["ACCELERATION"], row["DISPLACEMENT"], row["WEIGHT"]]))
    return obj
# ->
# Utiliza o RDD, aplica a função, converte para Dataframe e aplica a função select()
carrosRDD4 = carrosRDD3.map(transformaVar)
carrosDF = spSession.createDataFrame(carrosRDD4,["label", "features"])
carrosDF.select("label","features").show(10)
# ->
carrosRDD4.take(5)
## Machine Learning
# ->
# Dados de Treino e de Teste
(dados_treino, dados_teste) = carrosDF.randomSplit([0.7, 0.3])
# ->
dados_treino.count()
# ->
dados_teste.count()
# ->
# Treinamento e criação do modelo
linearReg = LinearRegression(maxIter = 10)
modelo = linearReg.fit(dados_treino)
# ->
print(modelo)
# ->
# Imprimindo as métricas
print("Coeficientes: " + str(modelo.coefficients))
print("Intercepto: " + str(modelo.intercept))
# ->
# Previsões com dados de teste
predictions = modelo.transform(dados_teste)
```

```
predictions.select("features", "prediction").show()
# ->
# Coeficiente de determinação R2
avaliador = RegressionEvaluator(predictionCol = "prediction", labelCol = "label", metricName = "r2")
avaliador.evaluate(predictions)
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Spark MLLib – Classificação – Decision Tree – Problema de Negócio, Limpeza e Análise de Correlação / Spark MLLib – Classificação – Decision Tree – Pré-processamento, Criação, Treino e Avaliação do Modelo

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 11</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 ou 11 e Apache Spark 2.4.2
****** Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook ******
## <font color='blue'>Spark MLLib - Classificação - Decision Tree</font>
<strong> Descrição </strong>
<ul style="list-style-type:square">
  <li>Fácil de compreender e fácil de explicar.</li>
  <li>Variáveis preditoras são usadas para construir uma árvore que progressivamente prevê valores target.</li>
  <li>Dados de treino são usados para construir a árvore de decisão e prever o valor target.</li>
  <li>A árovre de decisão se torna um modelo que é usado para fazer previsões com novos dados.</li>
</ul>
<dl>
  <dt>Vantagens</dt>
  <dd>- Fácil de interpretar e explicar</dd>
  <dd>- Funciona com valores missing</dd>
  <dd>- Veloz</dd>
  <br />
  <dt>Desvantagens</dt>
  <dd>- Acurácia limitada</dd>
  <dd>- Bias podem ocorrer com frequência</dd>
  <dd>- Não funciona bem com muitas variáveis preditoras</dd>
  <br />
  <dt>Aplicação</dt>
  <dd>- Aprovação de crédito</dd>
  <dd>- Categorização preliminar</dd>
</dl>
## Classificar as espécies de flores, listadas no dataset iris
# ->
# Imports
from pyspark.sql import Row
from pyspark.ml.feature import StringIndexer
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# ->
# Spark Session - usada quando se trabalha com Dataframes no Spark
spSession = SparkSession.builder.master("local").appName("DSA-SparkMLLib").getOrCreate()
# ->
# Carregando os dados e gerando um RDD
irisRDD = sc.textFile("data/iris.csv")
# ->
# Colocando o RDD em cache. Esse processo otimiza a performance.
irisRDD.cache()
# ->
irisRDD.count()
# ->
irisRDD.take(5)
# ->
# Removendo a primeira linha do arquivo (cabeçalho)
irisRDD2 = irisRDD.filter(lambda x: "Sepal" not in x)
irisRDD2.count()
## Limpeza dos Dados
# ->
# Separando as colunas
irisRDD3 = irisRDD2.map(lambda l: l.split(","))
# ->
```

```
# Mapeando as colunas
irisRDD4 = irisRDD3.map(lambda p: Row(SEPAL_LENGTH = float(p[0]), SEPAL_WIDTH = float(p[1]),
                                      PETAL_LENGTH = float(p[2]), PETAL_WIDTH = float(p[3]),
                                      SPECIES = p[4] ))
# ->
# Criando um Dataframe
irisDF = spSession.createDataFrame(irisRDD4)
irisDF.cache()
# ->
irisDF.cache()
# ->
irisDF.take(5)
# ->
# Criando um índice numérico para a coluna de label target
stringIndexer = StringIndexer(inputCol = "SPECIES", outputCol = "IDX_SPECIES") # converte categoria de string p/ um numero
si_model = stringIndexer.fit(irisDF)
irisNormDF = si_model.transform(irisDF)
# ->
irisNormDF.select("SPECIES","IDX_SPECIES").distinct().collect()
## Análise Exploratória de Dados
# ->
# Estatística descritiva
irisNormDF.describe().show()
# ->
# Correlação entre as variáveis
for i in irisNormDF.columns:
    if not(isinstance(irisNormDF.select(i).take(1)[0][0], str)) :
        print("Correlação da variável IDX_SPECIES com", i, irisNormDF.stat.corr('IDX_SPECIES', i))
## Pré-Processamento dos Dados
# ->
# Criando um LabeledPoint (target, Vector[features])
# Remove colunas não relevantes para o modelo ou com baixa correlação
def transformaVar(row) :
    obj = (row["SPECIES"], row["IDX_SPECIES"], Vectors.dense([row["SEPAL_LENGTH"], row["SEPAL_WIDTH"],
                                                              row["PETAL_LENGTH"], row["PETAL_WIDTH"]]))
    return obj
# ->
irisRDD5 = irisNormDF.rdd.map(transformaVar)
# ->
irisRDD5.take(5)
# ->
irisDF = spSession.createDataFrame(irisRDD5,["species", "label", "features"])
irisDF.select("species","label","features").show(10)
irisDF.cache()
## Machine Learning
# ->
# Dados de Treino e de Teste
(dados_treino, dados_teste) = irisDF.randomSplit([0.7, 0.3])
# ->
dados_treino.count()
# ->
dados_teste.count()
# ->
# Construindo o modelo com os dados de treino
dtClassifer = DecisionTreeClassifier(maxDepth = 2, labelCol = "label", featuresCol = "features")
modelo = dtClassifer.fit(dados_treino)
# ->
modelo.numNodes
modelo.depth
# ->
# Previsões com dados de teste
previsoes = modelo.transform(dados_teste)
previsoes.select("prediction","species","label").collect()
# ->
# Avaliando a acurácia
avaliador = MulticlassClassificationEvaluator(predictionCol = "prediction", labelCol = "label", metricName = "accuracy")
avaliador.evaluate(previsoes)
# ->
# Resumindo as previsões - Confusion Matrix
previsoes.groupBy("label","prediction").count().show()
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Spark MLLib – Classificação – Random Forest – Problema de Negócio, Limpeza e Análise de Correlação /
Spark MLLib – Classificação – Random Forest – Pré-processamento, Redução de Dimensionalidade e String Indexer /
Spark MLLib – Classificação – Random Forest – Construção, Treino e Teste do Classificador

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 11</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 ou 11 e Apache Spark 2.4.2
****** Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook ******
## <font color='blue'>Spark MLLib - Classificação - Random Forest</font>
<strong> Descrição </strong>
<ul style="list-style-type:square">
  <li>Um dos algoritmos mais populares.</li>
  <li>É um algoritmo de Método Ensemble.</li>
  <li>Um modelo de Random Forest constrói diversos modelos e cada modelo é usado para prever resultados de forma individual. Uma votação é feita pelo Random Forest paar escolher o melhor modelo.</li>
</ul>
<dl>
  <dt>Vantagens</dt>
  <dd>- Normalmente oferece boa acurácia</dd>
  <dd>- Eficiente com muitas variáveis preditoras</dd>
  <dd>- Funciona muito bem de forma paralelizada</dd>
  <dd>- Excelente com valores missing</dd>
  <br />
  <dt>Desvantagens</dt>
  <dd>- Mais lento</dd>
  <dd>- Bias podem ocorrer com frequência</dd>
  <br />
  <dt>Aplicação</dt>
  <dd>- Pesquisa científica</dd>
  <dd>- Diagnóstico médico</dd>
</dl>
## Classificar clientes de acordo com a possibilidade de pagar ou não o crédito
# ->
# Imports
import math
from pyspark.ml.linalg import Vectors
from pyspark.sql import Row
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import PCA
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# ->
# Spark Session - usada quando se trabalha com Dataframes no Spark
spSession = SparkSession.builder.master("local").appName("DSA-SparkMLLib").getOrCreate()
# ->
# Carregando os dados e gerando um RDD
bankRDD = sc.textFile("data/bank.csv")
# ->
bankRDD.cache()
# ->
bankRDD.count()
# ->
bankRDD.take(5)
# ->
# Removendo a primeira linha do arquivo (cabeçalho)
firstLine = bankRDD.first()
bankRDD2 = bankRDD.filter(lambda x: x != firstLine)
bankRDD2.count()
## Limpeza dos Dados
# ->
# Transformando os dados para valores numéricos
def transformToNumeric( inputStr) :
    attList = inputStr.replace("\"","").split(";")
    age = float(attList[0])
    outcome = 0.0 if attList[16] == "no" else 1.0
    single = 1.0 if attList[2] == "single" else 0.0
    married = 1.0 if attList[2] == "married" else 0.0
```

```
    divorced = 1.0 if attList[2] == "divorced" else 0.0
    primary = 1.0 if attList[3] == "primary" else 0.0
    secondary = 1.0 if attList[3] == "secondary" else 0.0
    tertiary = 1.0 if attList[3] == "tertiary" else 0.0
    default = 0.0 if attList[4] == "no" else 1.0
    balance = float(attList[5])
    loan = 0.0 if attList[7] == "no" else 1.0
    # Cria as linhas com os objetos transformados
    linhas = Row(OUTCOME = outcome, AGE = age, SINGLE = single, MARRIED = married, DIVORCED = divorced,
                 PRIMARY = primary, SECONDARY = secondary, TERTIARY = tertiary, DEFAULT = default, BALANCE = balance,
                 LOAN = loan)
    return linhas
# ->
# Aplicando a função de limpeza ao conjunto de dados
bankRDD3 = bankRDD2.map(transformToNumeric)
bankRDD3.collect()[:15]
## Análise Exploratória de Dados
# ->
# Transforma para Dataframe
bankDF = spSession.createDataFrame(bankRDD3)
# ->
# Estatística descritiva
bankDF.describe().show()
# ->
# Correlação entre as variáveis
for i in bankDF.columns:
    if not( isinstance(bankDF.select(i).take(1)[0][0], str)) :
        print( "Correlação da variável OUTCOME com", i, bankDF.stat.corr('OUTCOME',i))
## Pré-Processamento dos Dados
# ->
# Criando um LabeledPoint (target, Vector[features])
def transformaVar(row) :
    obj = (row["OUTCOME"], Vectors.dense([row["AGE"], row["BALANCE"], row["DEFAULT"], row["DIVORCED"], row["LOAN"],
                                          row["MARRIED"], row["PRIMARY"], row["SECONDARY"], row["SINGLE"],
                                          row["TERTIARY"]]))
    return obj
# ->
bankRDD4 = bankDF.rdd.map(transformaVar)
# ->
bankRDD4.collect()
# ->
bankDF = spSession.createDataFrame(bankRDD4,["label", "features"])
bankDF.select("features", "label").show(10)
# ->
# Aplicando Redução de Dimensionalidade com PCA
bankPCA = PCA(k = 3, inputCol = "features", outputCol = "pcaFeatures")
pcaModel = bankPCA.fit(bankDF)
pcaResult = pcaModel.transform(bankDF).select("label","pcaFeatures")
pcaResult.show(truncate = False)
# ->
# Indexação é pré-requisito para Decision Trees
stringIndexer = StringIndexer(inputCol = "label", outputCol = "indexed")
si_model = stringIndexer.fit(pcaResult)
obj_final = si_model.transform(pcaResult)
obj_final.collect()
## Machine Learning
# ->
# Dados de Treino e de Teste
(dados_treino, dados_teste) = obj_final.randomSplit([0.7, 0.3])
# ->
dados_treino.count()
# ->
dados_teste.count()
# ->
# Criando o modelo
rfClassifer = RandomForestClassifier(labelCol = "indexed", featuresCol = "pcaFeatures")
modelo = rfClassifer.fit(dados_treino)
# ->
# Previsões com dados de teste
predictions = modelo.transform(dados_teste)
predictions.select("prediction", "indexed", "label", "pcaFeatures").collect()
# ->
# Avaliando a acurácia
evaluator = MulticlassClassificationEvaluator(predictionCol = "prediction", labelCol = "indexed", metricName = "accuracy")
evaluator.evaluate(predictions)
# ->
# Confusion Matrix
predictions.groupBy("indexed", "prediction").count().show()
```

```
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

?

Spark MLLib – Classificação – Naive Bayes – Definição do Problema de Classificação de Spam /
Spark MLLib – Classificação – Naive Bayes – Processamento de Linguagem Natural /
Spark MLLib – Classificação – Naive Bayes – Pipeline, Criação, Treinamento e Avaliação do Modelo

```
 # <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 11</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 ou 11 e Apache Spark 2.4.2
****** Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook ******
## <font color='blue'>Spark MLLib - Classificação - Naive Bayes</font>
<strong> Descrição </strong>
<ul style="list-style-type:square">
  <li>Baseado no Teorema de Bayes.</li>
  <li>Probabilidade de um Evento A = P(A) é entre 0 e 1.</li>
  <li>A probabilidade P(A/B) = P(A intersect B ) * P (A) /P(B).</li>
  <li>A variável target se torna o evento A.</li>
  <li>O modelo armazena a probabilidade condicional da variável target para cada possível valor das variáveis preditoras.</li>
</ul>
<dl>
  <dt>Vantagens</dt>
  <dd>- Rápido e simples</dd>
  <dd>- Funciona bem mesmo com valores missing</dd>
  <dd>- Provê probabilidades de um resultado</dd>
  <dd>- Excelente com variáveis categóricas</dd>
  <br />
  <dt>Desvantagens</dt>
  <dd>- Não funciona bem com muitas variáveis numéricas</dd>
  <dd>- Espera que as variáveis preditoras sejam independentes</dd>
  <br />
  <dt>Aplicação</dt>
  <dd>- Filtro de Spam</dd>
  <dd>- Diagnóstico médico</dd>
  <dd>- Classificação de documentos</dd>
</dl>
## Classificação de Spam
https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection
# ->
# Imports
from pyspark.ml import Pipeline
from pyspark.ml.feature import IDF
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.ml.classification import NaiveBayes, NaiveBayesModel
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# ->
# Spark Session - usada quando se trabalha com Dataframes no Spark
spSession = SparkSession.builder.master("local").appName("DSA-SparkMLLib").getOrCreate()
# ->
# Carregando os dados e gerando um RDD
spamRDD = sc.textFile("data/SMSSpamCollection.csv", 2)
# ->
spamRDD.cache()
# ->
spamRDD.collect()
## Pré-Processamento dos Dados
# ->
def TransformToVector(inputStr):
    attList = inputStr.split(",")
    smsType = 0.0 if attList[0] == "ham" else 1.0
    return [smsType, attList[1]]
# ->
spamRDD2 = spamRDD.map(TransformToVector)
spamDF = spSession.createDataFrame(spamRDD2, ["label", "message"])
spamDF.cache()
```

```
spamDF.select("label", "message").show()
## Machine Learning
# ->
# Dados de Treino e de Teste
(dados_treino, dados_teste) = spamDF.randomSplit([0.7, 0.3])
# ->
dados_treino.count()
# ->
dados_teste.count()
# ->
# Divisão em palavras e aplicação do TF-IDF
tokenizer = Tokenizer(inputCol = "message", outputCol = "words")
hashingTF = HashingTF(inputCol = tokenizer.getOutputCol(), outputCol = "tempfeatures")
idf = IDF(inputCol = hashingTF.getOutputCol(), outputCol = "features")
nbClassifier = NaiveBayes()
# ->
# Criação do Pipeline
pipeline = Pipeline(stages = [tokenizer, hashingTF, idf, nbClassifier])
# ->
# Criação do modelo com o Pipeline
modelo = pipeline.fit(dados_treino)
# ->
# Previsões nos dados de teste
previsoes = modelo.transform(dados_teste)
previsoes.select("prediction", "label").collect()
# ->
# Avaliando a acurácia
avaliador = MulticlassClassificationEvaluator(predictionCol = "prediction", labelCol = "label", metricName = "accuracy")
avaliador.evaluate(previsoes)
# ->
# Resumindo as previsões - Confusion Matrix
previsoes.groupBy("label","prediction").count().show()
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Spark MLLib – Clustering – K-Means – Aprendizagem Não Supervisionada / Spark MLLib – Clustering – K-Means – Criação, Treinamento e Avaliação do Modelo

```
 # <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 11</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 ou 11 e Apache Spark 2.4.2
****** Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook ******
## <font color='blue'>Spark MLLib - Clustering - K-Means</font>
<strong> Descrição </strong>
<ul style="list-style-type:square">
  <li>Algoritmo Não Supervisionado.</li>
  <li>Agrupamento de Dados por Similaridade.</li>
  <li>Particiona os dados em um número "k" de clusters, sendo que cada observação pertende a apenas um cluster.</li>
  <li>A clusterização é feita medindo a distância entre os pontos de dados e agrupando-os.</li>
  <li>Múltiplas medidas de distância podem ser usadas, como distância Euclidiana e distância Manhattan.</li>
</ul>
<dl>
  <dt>Vantagens</dt>
  <dd>- Veloz</dd>
  <dd>- Eficiente quando se tem muitas variáveis</dd>
  <br />
  <dt>Desvantagens</dt>
  <dd>- O valor de K precisa ser conhecido</dd>
  <dd>- O valor inicial de k tem influência nos clusters criados</dd>
  <br />
  <dt>Aplicação</dt>
  <dd>- Agrupamento preliminar antes de se aplicar técnicas de classificação</dd>
  <dd>- Clusterização geográfica</dd>
</dl>
## Agrupando automóveis
# ->
# Imports
import math
```

```
import pandas as pd
from pyspark.sql import Row
from pyspark.ml.linalg import Vectors
from pyspark.ml.clustering import KMeans
import matplotlib.pylab as plt
%matplotlib inline
# ->
# Spark Session - usada quando se trabalha com Dataframes no Spark
spSession = SparkSession.builder.master("local").appName("DSA-SparkMLLib").getOrCreate()
# ->
# Carregando os dados e gerando um RDD
carrosRDD = sc.textFile("data/carros2.csv")
carrosRDD.cache()
# ->
# Removendo a primeira linha do arquivo (cabeçalho)
primeiraLinha = carrosRDD.first()
carrosRDD2 = carrosRDD.filter(lambda x: x != primeiraLinha)
carrosRDD2.count()
# ->
carrosRDD2.take(5)
# ->
# Convertendo e limpando os dados
def transformToNumeric( inputStr) :
    attList = inputStr.split(",")
    doors = 1.0 if attList[3] == "two" else 2.0
    body = 1.0 if attList[4] == "sedan" else 2.0
    linhas = Row(DOORS = doors, BODY = float(body), HP = float(attList[7]), RPM = float(attList[8]),
                 MPG = float(attList[9]))
    return linhas
# ->
# Aplicando a função
carrosRDD3 = carrosRDD2.map(transformToNumeric)
carrosRDD3.persist()
carrosRDD3.take(5)
# ->
# Criando um Dataframe
carrosDF = spSession.createDataFrame(carrosRDD3)
carrosDF.show()
# ->
# Sumarizando os dados e extraindo a média e o desvio padrão
estats = carrosDF.describe().toPandas()
medias = estats.iloc[1,1:5].values.tolist()
desvios = estats.iloc[2,1:5].values.tolist()
# ->
# Colocando a média e o desvio padrão e variáves do tipo Broadcast
bc_media = sc.broadcast(medias)
bc_desvio = sc.broadcast(desvios)
# ->
# Função para centralizar e aplicar escala aos dados. Cada valor será subtraído da média então dividido pelo desvio padrão
def centerAndScale(inRow) :
    global bc_media
    global bc_desvio
    meanArray = bc_media.value
    stdArray = bc_desvio.value
    retArray = []
    for i in range(len(meanArray)):
        retArray.append( (float(inRow[i]) - float(meanArray[i])) / float(stdArray[i]) )
    return Vectors.dense(retArray)
# ->
carrosRDD4 = carrosDF.rdd.map(centerAndScale)
carrosRDD4.collect()
# ->
# Criando um Dataframe
carrosRDD5 = carrosRDD4.map( lambda f:Row(features = f))
carrosDF = spSession.createDataFrame(carrosRDD5)
carrosDF.select("features").show(10)
# ->
# Criando o modelo
kmeans = KMeans(k = 3, seed = 1)
modelo = kmeans.fit(carrosDF)
# ->
# Previsões
previsoes = modelo.transform(carrosDF)
previsoes.show()
# ->
def unstripData(instr) :
    return (instr["prediction"], instr["features"][0], instr["features"][1], instr["features"][2], instr["features"][3])
# ->
```

```
carrosRDD6 = previsoes.rdd.map(unstripData)
predList = carrosRDD6.collect()
predPd = pd.DataFrame(predList)
# ->
# Gráfico com o resultados dos clusters criados
plt.cla()
plt.scatter(predPd[3], predPd[4], c = predPd[0])
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Mini-Projeto 3 – Sistema de Recomendação

```
 # <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 11</font>
### ********** Atenção: **********
Utilize Java JDK 1.8 ou 11 e Apache Spark 2.4.2
****** Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook ******
## <font color='blue'>Spark MLLib - Sistema de Recomendação</font>
<strong> Descrição </strong>
<ul style="list-style-type:square">
  <li>Também chamado de filtros colaborativos.</li>
  <li>Analisa dados passados para compreender comportamentos de pessoas/entidades.</li>
  <li>A recomendação é feita por similaridade de comportamento.</li>
  <li>Recomendação baseada em usuários ou items.</li>
  <li>Algoritmos de Recomendação esperam receber os dados em um formato específico: [user_ID, item_ID, score].</li>
  <li>Score, também chamado rating, indica a preferência de um usuário sobre um item. Podem ser valores booleanos, ratings ou mesmo volume de vendas.</li>
</ul>
# ->
# Imports
from pyspark.ml.recommendation import ALS
# ->
# Spark Session - usada quando se trabalha com Dataframes no Spark
spSession = SparkSession.builder.master("local").appName("DSA-SparkMLLib").getOrCreate()
# ->
# Carrega os dados no formato ALS (user, item, rating)
ratingsRDD = sc.textFile("data/user-item.txt")
ratingsRDD.collect()
# ->
# Convertendo as strings
ratingsRDD2 = ratingsRDD.map(lambda l: l.split(',')).map(lambda l:(int(l[0]), int(l[1]), float(l[2])))
# ->
# Criando um Dataframe
ratingsDF = spSession.createDataFrame(ratingsRDD2, ["user", "item", "rating"])
# ->
ratingsDF.show()
# ->
# Construindo o modelo
# ALS = Alternating Least Squares --> Algoritmo para sistema de recomendação, que otimiza a loss function (função que calcula
# a taxa de erros do modelo) e funciona muito bem em ambientes paralelizados
als = ALS(rank = 10, maxIter = 5)
modelo = als.fit(ratingsDF)
# ->
# Visualizando o Affinity Score
modelo.userFactors.orderBy("id").collect()
# ->
# Criando um dataset de teste com usuários e items para rating
testeDF = spSession.createDataFrame([(1001, 9003),(1001,9004),(1001,9005)], ["user", "item"])
# ->
# Previsões
# Quanto maior o Affinity Score, maior a probabilidade do usuário aceitar uma recomendação
previsoes = (modelo.transform(testeDF).collect())
previsoes
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Deploy dos Modelos de Machine Learning

Utilize o Mleap.

Documentação em: https://mleap-docs.combust.ml/

Quiz

Qual o tipo do objeto autoData?
autoData = sc.textFile("carros2.csv")
R: Um RDD!

Qual dos comandos abaixo importa o algoritmo K-Means no Spark MLLib?
from pyspark.ml.clustering import KMeans

K-Means é um tipo de algoritmo não supervisionado que resolve problemas de agrupamento. O seu procedimento segue uma maneira simples e fácil para classificar um dado conjunto através de um certo número de grupos de dados (clusters). Os pontos de dados dentro de um cluster são homogêneos, mas os clusters são heterogêneos.

Em um modelo de Machine Learning com viés mais alto e variância mais baixa podemos ter problema de Underfitting.

O nosso objetivo é reduzir o viés e a variância o máximo que pudermos, entretanto, nos deparamos com um trade-off entre underfitting e overfitting.

Códigos Fonte:

Cap02:

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 2</font>
Para gerar um arquivo pdf a partir do Jupyter Notebook, você precisar ter instalado em seu
computador:
    MiKTeX e Pandoc (Windows)
    LaTeX e Pandoc (Mac)
    TeX Live e Pandoc (linux)
Neste link você encontra todas as informações: http://pandoc.org/installing.html
## Computação com Numpy
# ->
# Importando o módulo Numpy
import numpy as np
<h2>Criando Estruturas de Dados NumPy</h2>
# ->
# Criando array unidimensional
array1 = np.array([10, 20, 30, 40])
# ->
# Criando array bi-dimensional
array2 = np.array([[100, 83, 15],[42, 78, 0]])
# ->
array1
# ->
array2
# ->
# Verificando o formato (shape) do array
array1.shape
# ->
# Verificando o formato (shape) do array
array2.shape
# ->
# Verificando o número de dimensões
array2.ndim
<h2>Em Python TUDO é objeto, com métodos e atributos</h2>
### Método - realiza ação no objeto
### Atributo - característica do objeto
# ->
# Método
array2.max()
# ->
# Atributo
array2.ndim
<h2>Criando estruturas de dados com arange</h2>
# ->
array3 = np.arange(15)
# ->
array3
# ->
# Usando start/end (exclusive)
array4 = np.arange(0, 15, 5)
# ->
array4
<h2>Criando estruturas de dados com linspace</h2>
# ->
# Argumentos: (start, end, number of elements)
array5 = np.linspace(0, 3, 4); array5 # ponto e vírgula separa comandos numa única linha
<h2>Criando estruturas de dados com outras funções</h2>
<h3>numpy.zeros</h3>
# ->
# Array 10x8 de zeros
array6 = np.zeros((10, 8)); array6
<h3>numpy.ones</h3>
# ->
# Array 2x3x2 de 1's
array7 = np.ones((2, 3, 2)); array7
<h3>numpy.eye</h3>
# ->
# Produz uma Identitiy Matrix (matriz de identidade)
array8 = np.eye(3); array8
<h3>numpy.diag</h3>
# ->
# Matriz diagonal
```

```
array9 = np.diag((2,1,4,6)); array9
<h3>numpy.random.rand</h3>
A função rand está no pacote random, que está no pacote numpy
# ->
# A função rand(n) produz uma sequência de números uniformemente distribuídos com range de 0 a n
np.random.seed(100) # Set seed
array10 = np.random.rand(5); array10
# ->
# A função randn(n) produz uma sequência de números com distribuição normal (Gaussian)
array11 = np.random.randn(5); array11
<h3>numpy.empty</h3>
# ->
array12 = np.empty((3,2)); array12
<h3>numpy.tile</h3>
# ->
np.array([[9, 4], [3, 7]])
# ->
np.tile(np.array([[9, 4], [3, 7]]), 4) # multiplica array
# ->
np.tile(np.array([[9, 4], [3, 7]]), (2,2))
<h2>Tipos de Dados NumPy</h2>
# ->
array13 = np.array([8, -3, 5, 9], dtype = 'float') # define o tipo de dado do array
# ->
array13
# ->
array13.dtype # verifica o tipo de dado do array
# ->
array14 = np.array([2, 4, 6, 8])
# ->
array14
# ->
array14.dtype
# ->
array15 = np.array([2.0, 4, 6, 8])
# ->
array15.dtype
# ->
array16 = np.array(['Análise','de','Dados','com Python'])
# ->
array16.dtype
# ->
array17 = np.array([True, False, True])
# ->
array17.dtype
# ->
array18 = np.array([7, -3, 5.24])
# ->
array18.dtype
# ->
array18.astype(int) # converte um tipo de dado para outro
<h2>Operações com Arrays</h2>
# ->
# Array começando por 0, com 30 elementos e multiplicado cada um por 5
array19 = np.arange(0, 30) * 5
# ->
array19
# ->
array19 = np.arange(0, 30)
# ->
array19
# ->
# Elevando um array a potência
array20 = np.arange(5) ** 4
# ->
array20
# ->
# Somando um número a cada elemento do array
array21 = np.arange(0, 30) + 1
# ->
array21
# ->
array22 = np.arange(0, 30,3) + 3
array23 = np.arange(1,11)
# ->
array22
# ->
array23
# ->
```

```
# Subtração com numpy
array22 - array23
# ->
# Soma com numpy
array22 + array23
# ->
# Divisão com numpy
array22 / array23
# ->
# Multiplicação com numpy
array22 * array23
# ->
# Podemos ainda comparar arrays com numpy
array22 > array23
# ->
arr1 = np.array([True,False,True,False])
arr2 = np.array([False,False,True, False])
np.logical_and(arr1, arr2)
### Podemos criar arrays com Python, sem usar o módulo NumPy. Porém o NumPy é muito mais rápido
# ->
array_numpy = np.arange(1000)
%timeit array_numpy ** 4 # %timeit é uma função do jupyter notebook que calcula o tempo de execução
# ->
array_python = range(1000)
%timeit [array_python[i] ** 4 for i in array_python]
A métrica µs significa microsegundos.
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 2</font>
## Indexação e Slicing com Numpy
# ->
# Importando o módulo Numpy
import numpy as np
# ->
array1 = np.arange(5)
# ->
array1
# ->
# Primeiro elemento do array (em Python a indexação começa por 0)
array1[0]
# ->
# Segundo elemento do array (em Python a indexação começa por 0)
array1[1]
# ->
# Último elemento do array (em Python a indexação começa por 0)
array1[-1]
# ->
# Criando um array
array2 = np.arange(5)
# ->
array2
# ->
# Invertendo o array ou lista
array2[::-1]
## Trabalhando com Array multi-dimensional
# ->
from IPython.display import Image
Image('Numpy.png')
# ->
array3 = np.array([[190, 290, 490], [129, 458, 567], [761, 902, 346]])
# ->
array3
# ->
# Slicing da Matriz
array3[2, 1]
# ->
# Atribuindo um valor a um elemento da Matriz
array3[1, 1] = 10
# ->
array3
# ->
array3[2]
# ->
# Retornando a terceira linha da Matriz
array3[2,]
# ->
```

```
# Retornando todas as linhas da segunda coluna da Matriz
array3[:,1]
# ->
array4 = np.array([100, 200, 300])
# ->
array4[5] # erro: indice que nao existe
<h2>Slicing de Arrays</h2>
# ->
array5 = np.arange(50) * 2
# ->
array5
# ->
array5[5:10:2] # elementos que começam no índice 5 até o índice 10, saltando de 2 em 2
# ->
# Retorna todos os valores até o elemento de index 4 (exclusive)
array5[:4]
# ->
# A partir do elemento de index 4, retorna todos os valores
array5[4:]
# ->
array5[::3]
# ->
# O slicing não altera o objeto original
array5
# ->
# Podemos usar as técnicas de slicing para alterar elementos ou conjuntos de elementos dentro do
array
array5[:3] = 1; array5
# ->
# Atribuindo um array a uma parte de outro array
array5[:4] = np.ones(4); array5
<h2>Criando um subset de um array</h2>
# ->
array7 = np.arange(12)
# ->
array7
# ->
array8 = array7[::2]
# ->
array8
## Copiando um array com np.copy
# ->
array9 = np.arange(8)
# ->
array9
# ->
array10 = array9.copy() # método copy copia um array
# ->
array10
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 2</font>
# Transposição e Reshaping de Arrays
# ->
# Importando o módulo Numpy
import numpy as np
# ->
array1 = np.array([[65, 23, 19], [41, 87, 10]])
# ->
array1
# ->
array1.T # pivo dos dados ou matriz transposta
# ->
np.transpose(array1)
## Broadcasting
# ->
array2 = np.array([[33, 34, 35]])
# ->
array2.T
# ->
array2.T + array2
## Transformando Matrizes em Vetores
# ->
array3 = np.array([np.arange(1,6), np.arange(10,15)])
# ->
array3
```

```
# ->
array3.ravel() # ravel transforma uma matriz em um vetor
# ->
array3.T.ravel() # transforma em transposta e depois num vetor
## Operadores Lógicos
# ->
np.random.seed(100)
array4 = np.random.randint(1, 10, size = (4,4))
# ->
array4
# ->
np.any((array4 % 7) == 0) # any perguta se existe algum elemento (true ou false)
# ->
np.all(array4 < 11) # all pergunta se todos os elementos são
## Reshaping
# ->
array5 = np.arange(1,16)
# ->
array5
# ->
array5.reshape(3, 5) # reshape transforma um vetor em uma matriz
# Ordenando o Array
# ->
array7 = np.array([[3,2],[10,-1]])
# ->
array7
# ->
# Ordenando por linhas (axis=1)
array7.sort(axis = 1)
# ->
array7
# ->
array8 = np.array([[3,2],[10,-1]])
# ->
array8
# ->
# Ordenando por colunas (axis=0)
array8.sort(axis = 0)
# ->
array8
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 2</font>
# Estruturas de Dados em Pandas
pandas.Series()          - 1 Dimensão  - 1D
pandas.DataFrame()  - 2 Dimensões - 2D
pandas.Panel()           - 3 Dimensões - 3D
numpy.ndarray()     - x Dimensões
# DataFrames
# ->
import pandas as pd
import numpy as np
# ->
# dict ou dicionario é um conjunto de pares, chave e valor
stock = {'AMZN': pd.Series([346.15,0.59,459,0.52,589.8,158.88],
                           index = ['Closing price','EPS','Shares Outstanding(M)',
                                    'Beta', 'P/E','Market Cap(B)']),
         'GOOG': pd.Series([1133.43,36.05,335.83,0.87,31.44,380.64],
                           index = ['Closing price','EPS','Shares Outstanding(M)',
                                    'Beta','P/E','Market Cap(B)']),
         'FB': pd.Series([61.48,0.59,2450,104.93,150.92],
                           index = ['Closing price','EPS','Shares Outstanding(M)',
                                    'P/E', 'Market Cap(B)']),
         'YHOO': pd.Series([34.90,1.27,1010,27.48,0.66,35.36],
                           index=['Closing price','EPS','Shares Outstanding(M)',
                                    'P/E','Beta', 'Market Cap(B)']),
         'TWTR':pd.Series([65.25,-0.3,555.2,36.23],
                           index=['Closing price','EPS','Shares Outstanding(M)',
                                    'Market Cap(B)']),
         'AAPL':pd.Series([501.53,40.32,892.45,12.44,447.59,0.84],
                           index=['Closing price','EPS','Shares Outstanding(M)','P/E',
                                    'Market Cap(B)','Beta'])}
# ->
type(stock)
# ->
stock_df = pd.DataFrame(stock)
```

```
# ->
stock_df
# ->
stock_df = pd.DataFrame(stock,
                        index = ['Closing price',
                                 'EPS',
                                 'Shares Outstanding(M)',
                                 'P/E',
                                 'Market Cap(B)',
                                 'Beta'])
# ->
stock_df
# ->
stock_df = pd.DataFrame(stock,
                        index = ['Closing price',
                                 'EPS',
                                 'Shares Outstanding(M)',
                                 'P/E',
                                 'Market Cap(B)',
                                 'Beta'],
                        columns = ['FB','TWTR','SCNW']) # seleção de colunas (se nao existir,
recebe NaN)
# ->
stock_df
# ->
stock_df.index # lista os índices do data frame
# ->
stock_df.columns # lista as colunas do data frame
## Operações com DataFrames
# ->
Quadro_Medalhas_Olimpiada = {
    'USA' : {'Ouro':46, 'Prata':37, 'Bronze':38},
    'China':{'Ouro':26, 'Prata':18, 'Bronze':26},
    'Britain':{'Ouro':27, 'Prata':23, 'Bronze':17},
    'Russe':{'Ouro':19, 'Prata':18, 'Bronze':19},
    'Germany':{'Ouro':17, 'Prata':10, 'Bronze':15}}
# ->
olimpiada = pd.DataFrame.from_dict(Quadro_Medalhas_Olimpiada) # from_dict é necessário pelo objeto
ser um dicionário aninhado
# ->
olimpiada
# ->
type(olimpiada)
# ->
Medalhas_China = olimpiada['China'] # criando subset do data frame da coluna China
# ->
Medalhas_China
# ->
olimpiada['China']
# ->
olimpiada.Russe # chamando um atributo do data frame
# ->
olimpiada[['Russe','Germany']] # colchetes externos representam slicing; os internos, uma lista
# ->
olimpiada.get('Germany')
# ->
olimpiada
# ->
type(olimpiada)
# ->
# iloc - busca no data frame utilizando índices
olimpiada.iloc[1]
## Slicing
# ->
olimpiada
# ->
olimpiada[:2] # traz todas as linhas até a do índice 2 exclusive
# ->
olimpiada[2:] # traz todas as linhas a partir da do índice 2 inclusive
# ->
olimpiada[::2] # traz todas as linhas, saltando de 2 em 2
# ->
olimpiada[::-1] # traz todas as linhas, invertendo a ordem das linhas
# ->
olimpiada # slicing não altera o data frame original
## Slicing por indexação
# ->
olimpiada.loc['Ouro'] # utilizando o método loc é possível utilizar índices nomeados do data frame
# ->
```

```
olimpiada.loc[:,'USA']
# ->
olimpiada.loc['Prata','China']
# ->
olimpiada.loc['Prata']['China'] # outra anotação, para o mesmo resultado da anotação acima
# ->
olimpiada.loc['Prata']
# ->
olimpiada.loc['Ouro'] > 20
# ->
olimpiada.loc[:, olimpiada.loc['Ouro'] > 20]
# ->
olimpiada.loc['Prata'] > 20
# ->
olimpiada.iloc[:2] # iloc é mais rápido que o loc; iloc é usado para apontar índices de forma
numérica
# ->
olimpiada.iloc[2,0:2]
# ->
olimpiada.iloc[2:3, :]
# ->
olimpiada.iloc[1,:]
# ->
olimpiada.iloc[2, 0]
## Removendo um membro do dataframe
# ->
del olimpiada['USA'] # del deleta, de acordo com a condição
# ->
olimpiada
## Inserindo um membro no dataframe
# ->
olimpiada.insert(0, 'Brasil', (7, 6, 6)) # insert (índice, nome da coluna, valores das linhas)
# ->
olimpiada
## Resumo do dataframe
# ->
olimpiada.describe() # resumo estatísco completo do conjunto de dados
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 2</font>
# Estruturas de Dados em Pandas
pandas.Series()          - 1 Dimensão  - 1D
pandas.DataFrame()  - 2 Dimensões - 2D
pandas.Panel()           - 3 Dimensões - 3D
numpy.ndarray()     - x Dimensões
# Panel
# ->
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
# ->
data = np.random.randint(1, 10, (5, 3, 2)) # random.randint organiza dados de maneira aleatória
# ->
painel = pd.Panel(
    data,
    items = ['item {}'.format(i) for i in range(1, 6)],
    major_axis = [2015, 2016, 2017],
    minor_axis = ['US', 'UK'])
# ->
print(painel)
# ->
data = data.reshape(5, 6).T
# ->
# reshape com o pacote MultiIndex, dentro de DataFrame (mais moderno)
df = pd.DataFrame(
    data = data,
    index = pd.MultiIndex.from_product([[2015, 2016, 2017], ['US', 'UK']]),
    columns = ['item {}'.format(i) for i in range(1, 6)])
# ->
df
## Criando objetos multidimensionais com Panel
# ->
USData = pd.DataFrame(np.array([[249.62 , 8900],
                                [282.16,12680],
                                [309.35,14940]]),
```

```
                      columns = ['População(M)','PIB($B)'],
                      index = [1995,2005,2015])
# ->
USData
# ->
ChinaData = pd.DataFrame(np.array([[1133.68, 390.28],
                                   [1266.83,1198.48],
                                   [1339.72, 9923.47]]),
                         columns=['População(M)','PIB($B)'],
                         index=[1995,2005,2015])
# ->
ChinaData
# ->
US_ChinaData = {'US' : USData,
                'China': ChinaData}
# ->
US_ChinaData
# ->
pd.Panel(US_ChinaData)
# ->
type(pd.Panel(US_ChinaData))
## Convertendo DataFrame para Panel
# ->
mIdx = pd.MultiIndex(levels = [['US', 'China'], [1995,2005, 2015]],
                     labels = [[1,1,1,0,0,0], [0,1,2,0,1,2]])
# ->
mIdx
# ->
ChinaUSDF = pd.DataFrame({'População(M)' : [1133.68, 1266.83, 1339.72, 249.62, 282.16,309.35],
                          'PIB($B)': [390.28, 1198.48, 6988.47, 8900,12680,14940]},
                         index = mIdx)
# ->
ChinaUSDF
# ->
# ChinaUSDF.to_panel()
!pip install xarray # comando que começa com ! será executado no sistema operacional, a partir do
jupyter
# ->
ChinaUSDF.to_xarray()
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 2</font>
# <font color='blue'>Exercício</font>
Dados obtidos do Portal Brasileiro de Dados Abertos:
http://dados.gov.br/dataset/cidades-digitais
O projeto Cidades Digitais foi planejado para modernizar a gestão e ampliar o acesso ao serviço
público, bem como promover o desenvolvimento dos municípios brasileiros através da tecnologia.
# Group By e Reshape
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html
https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html
# ->
import numpy as np
import pandas as pd
# ->
cidadesDigitais = pd.read_csv('cidades_digitais.csv', sep = ';', encoding = 'latin-1')
# ->
cidadesDigitais.head()
# ->
cidadesDigitais.count()
# ->
del cidadesDigitais['IBGE']
# ->
# Aplicando Group By
cidadesUF_1 = cidadesDigitais.groupby('UF')
# ->
type(cidadesUF_1)
# ->
cidadesUF_1.groups
# ->
# Número de grupos
len(cidadesUF_1.groups)
# ->
ordenaUF = cidadesUF_1.size()
# ->
# Ordena os grupos por ordem decrescente
# ->
```

```
cidadesUF_2 = cidadesDigitais.groupby(['UF','STATUS'])
# ->
# Total agrupado
# ->
cidadesDigitais.head()
# ->
cidadesUF_3 = cidadesDigitais.set_index('STATUS')
# ->
# Aplicando Group By
# ->
cidadesUF_3.head()
# ->
# ->
cidadesUF_4
# ->
cidadesDigitais.head(3)
# ->
# Fim
### Obrigado - Data Science Academy - <a
href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Cap03:

```
 # <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
Dados obtidos do Portal Brasileiro de Dados Abertos:
http://dados.gov.br/dataset/cidades-digitais
O projeto Cidades Digitais foi planejado para modernizar a gestão e ampliar o acesso ao serviço público, bem como promover o desenvolvimento dos municípios brasileiros através da tecnologia.
# Group By e Reshape
# ->
import numpy as np
import pandas as pd
# ->
cidadesDigitais = pd.read_csv('data/cidades_digitais.csv', sep = ';', encoding = 'latin-1')
# ->
cidadesDigitais.head()
# ->
cidadesDigitais.count()
# ->
del cidadesDigitais['IBGE']
# ->
# Aplicando Group By
cidadesUF_1 = cidadesDigitais.groupby('UF')
# ->
type(cidadesUF_1)
# ->
cidadesUF_1.groups
# ->
# Número de grupos
len(cidadesUF_1.groups)
# ->
ordenaUF = cidadesUF_1.size()
# ->
# Ordena os grupos por ordem decrescente
ordenaUF.sort_values(ascending = False)
# ->
cidadesUF_2 = cidadesDigitais.groupby(['UF','STATUS'])
# ->
total = cidadesUF_2.size()
total.sort_values(ascending = False)
total
# ->
cidadesDigitais.head()
# ->
cidadesUF_3 = cidadesDigitais.set_index('STATUS')
# ->
# Aplicando Group By
cidadesUF_3 = cidadesUF_3.groupby(level = 'STATUS')
# ->
cidadesUF_3.head()
# ->
cidadesUF_4 = cidadesDigitais.set_index(['STATUS', 'UF'])
# ->
cidadesUF_4
# ->
cidadesUF_4 = cidadesDigitais.set_index(['UF', 'STATUS'])
# ->
cidadesUF_4 = cidadesUF_4.groupby(level = ['UF', 'STATUS'])
# ->
cidadesUF_4.sum()
# ->
cidadesUF_4.aggregate(np.sum)
# ->
cidadesUF_4.aggregate([np.sum, np.mean, np.size])
# ->
cidadesUF_5 = cidadesDigitais.set_index([ 'STATUS', 'CIDADE', 'VALOR_TOTAL_PREVISTO', 'VALOR_INVESTIDO', 'POPULAÇÃO', 'PONTOS_ATENDIDOS'])
# ->
cidadesUF_5 = cidadesUF_5.groupby(level = ['STATUS', 'VALOR_TOTAL_PREVISTO', 'VALOR_INVESTIDO'])
# ->
cidadesDigitais.head(3)
# ->
cidadesUF_5.sum()
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
```
Map
A função map() faz basicamente o mapeamento de um valor para outro, quase como um objeto do tipo dicionário em Python. Este é um conceito de programação funcional, mas que pode ser útil em certas circunstâncias dentro do processo de análise de dados. Este é conceito fundamental de programação MapReduce, popular em tecnologias de processamento de Big Data, como Hadoop e Spark.
```
# ->
# Definindo um array com a função bulti-in range()
array1 = range(0, 15)
# ->
print(array1)
# ->
type(array1)
# ->
# Criando um função para calcular o cubo de um número
def calc_cubo(num):
    return num ** 3
# ->
for elemento in array1:
    print(calc_cubo(elemento))
# ->
# Com a função map() aplicamos uma função a um conjunto de elementos
lista_map_func = map(calc_cubo, array1)
# ->
# A função map() retorna um objeto iterable
print(lista_map_func)
# ->
# Precisamos converter a saída da função map() para uma lista, a fim de visualizar os resultados
print(list(lista_map_func))
# ->
# Com a função map() aplicamos uma função a um conjunto de elementos
lista_map_func_2 = map(float, range(20))
# ->
lista_map_func_2
# ->
print(list(lista_map_func_2))
# ->
from math import sqrt
# ->
array2 = list(array1)
# ->
array2
# ->
print(list(map(sqrt, array2[0:5])))
```
Filter
A função filter() permite filtrar elementos de uma lista. Aplicamos uma função que retorna valores booleanos (True ou False). Retornamos os valores onde a expressão é True e filtramos os valores onde a expressão retorna False.
```
# ->
# Definindo um array com a função bulti-in range()
array1 = range(0, 15)
# ->
print(array1)
# ->
type(array1)
# ->
# Criando um função para verificar se um número é par
def verificaPar(num):
    return num % 2 == 0
# ->
verificaPar(190)
# ->
verificaPar(191)
# ->
# Com a função map() aplicamos uma função a um conjunto de elementos
list(map(verificaPar, array1))
# ->
# A função filter() retorna apenas os valores onde a expressão é igual a True,
# filtrando os valores onde a expressão retorna False
list(filter(verificaPar, array1))
# ->
# Lista de frases
lista_frases = ["Big Data",
                "Machine Learning em Big Data e Algoritmos em Big Data",
                "Data Science e Big Data",
                "Big Data e Internet das Coisas",
                "Big Data está revolucionando o mundo"]
# ->
```

```
# Criando uma função
def testa_BigData(frase):
    return frase.count("Big Data") >= 2
# ->
# Aplicando a função filter()
list(filter(testa_BigData, lista_frases))
# ->
# Aplicando a função map()
list(map(testa_BigData, lista_frases))
## Reduce
A função reduce() aplica uma função a uma sequência de elementos, até reduzir a sequência a um único elemento
# ->
from functools import reduce
# ->
soma = reduce((lambda x, y: x + y), [1, 2, 3, 4])
# ->
soma
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## List Comprehension
A utilização de list comprehension em Python é muito comum. List comprehension é a forma que Python utiliza para realizar filtros e mapeamentos, semelhante ao que vimos com as funções filter() e map().
# ->
# Definindo um array com a função bulti-in range()
array1 = range(0, 15)
# ->
print(array1)
# ->
type(array1)
# ->
# Aplicando list comprehension
[item ** 3 for item in array1]
# ->
[item for item in array1 if item % 2 == 0]
# ->
[item ** 2 for item in array1 if item % 2 == 0]
# ->
# Lista de frases
lista_frases = ["Big Data",
                "Machine Learning em Big Data e Algoritmos em Big Data",
                "Data Science e Big Data",
                "Big Data e Internet das Coisas",
                "Big Data está revolucionando o mundo, pois Big Data é um grande volume de dados"]
# ->
[item for item in lista_frases if item.count("Big Data") >= 2]
# ->
[float(item) for item in array1]
## Funções Lambda
Funções lambda são funções anônimas que não precisam ser definidas antes de serem usadas. São também chamadas funções inline e ajudam a manter seu código mais organizado.
# ->
# Definindo um array com a função bulti-in range()
array1 = range(0, 15)
# ->
print(array1)
# ->
type(array1)
# ->
# Aplicando list comprehension
[item ** 2 for item in array1]
# ->
# Definindo um função
def square(num):
    return num ** 2
# ->
# Aplicando a função map()
list(map(square, array1))
# ->
# Função lambda
list(map(lambda num: num**2, array1))
# ->
list(map(lambda a: a**2, filter(lambda a: a % 2 == 0, range(1,20))))
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Pandas - DataFrames
# ->
import sys
import string
import pandas as pd
import numpy as np
print(sys.version)
pd.__version__
# Se precisar instalar uma versão específica o pandas, use:
# !pip uninstall pandas
# !pip install pandas == 0.23.4
## Operações com DataFrames
# ->
dict1 = {'Coluna1': [190, 231, 784, 127],
        'Coluna2': [545, 278, 104, 347],
        'Coluna3': [665, 224, 901, 503],
        'Coluna4': [123, 456, 789, 763]}
# ->
dict1
# ->
df = pd.DataFrame(dict1)
# ->
df
# ->
df.dtypes # dtypes mostra os tipos de dados do dataframe
# ->
df['Coluna2']
# ->
df['Coluna2'][0]
# ->
# Utilizar o nome da coluna ou .loc ou .iloc ou .ix
df.iloc[0]
# ->
df[:2]
# ->
df.index
# ->
df.insert(0, 'ID', [1, 2, 3, 4]) # inserindo coluna no dataframe (0: primeiro índice de colunas)
# ->
df
# ->
df2 = pd.DataFrame(df.set_index('ID')) # removendo índice padrão, selecionando outra coluna como índice
# ->
df2
## Manipulando e Organizando DataFrames
# ->
colunas = list(range(26))
# ->
ucase = [x for x in string.ascii_uppercase]
lcase = [x for x in string.ascii_lowercase]
# ->
alfabeto = pd.DataFrame([lcase, ucase, colunas])
# ->
alfabeto
# ->
# Transposta da Matriz (DataFrame)
alfabeto = alfabeto.T
# ->
alfabeto.head()
# ->
alfabeto.index
# ->
alfabeto.columns
# ->
alfabeto.columns = ['minusculo', 'maiusculo', 'numero'] # altera título das colunas
# ->
alfabeto.head()
# ->
alfabeto.minusculo.head()
# ->
alfabeto['minusculo'].head()
# ->
alfabeto.index = pd.date_range('7/1/2019', periods = 26) # criando índice com datas ou período
# ->
alfabeto[:10]
```

```
# ->
alfabeto['2019-07-12':'2019-07-19']
# ->
letras = pd.DataFrame({'minusculas':lcase, 'maiusculas':ucase})
letras.head()
# ->
np.random.seed(25)
letras['Numeros'] = np.random.randint(1, 50, 26) # criando coluna com valores randomicos
# ->
letras.head()
# ->
letras.dtypes
# ->
letras.index = lcase
# ->
letras[:10]
# ->
letras.sort_values('Numeros').head()
# ->
letras.sort_index().head()
# ->
letras[['minusculas','maiusculas']].head()
# ->
letras.iloc[5:10]
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## NumPy
NumPy oferece sequências de elementos que são mais eficientes que sequências padrões em Python.
# ->
import sys
import numpy as np
# descobrindo a versão:
print(sys.version)
np.__version__
# ->
# Criando um array NumPy
array1 = np.arange(15)
# ->
array1
# ->
# Help
# ?array1
# ->
array1.mean()
# ->
array1.sum()
# ->
array1.min()
# ->
array1.max()
# ->
array1.std()
## Utilizando List Compreheension em Arrays NumPy
# ->
# Utilizando list compreheension em arrays NumPy
[x * x for x in array1]
# ->
# Utilizando list compreheension em arrays NumPy
[x * x for x in array1 if x % 2 == 0]
## Utilizando Função Lambda em Arrays NumPy
# ->
# Utilizando expressão lambda
list(filter(lambda x: x % 2 == 0, array1))
# ->
# Atenção com a notação
array1 % 2 == 0
# ->
# # Atenção com a notação
array1[array1 % 2 == 0]
## Avaliando a Performance
# ->
# Performance usando list compreheension
%timeit [x for x in array1 if x % 2 == 0]
# ->
```

```
# Performance usando NumPy
%timeit array1[array1 % 2 == 0]
## Usando Operadores Lógicos
# ->
array1 > 8
# ->
array1[array1 > 8]
# ->
# Operador lógico and
(array1 > 9) & (array1 < 12)
# ->
# Operador lógico or
(array1 > 13) | (array1 < 12)
# ->
# Operador lógico or
array1[(array1 > 13) | (array1 < 12)]
# ->
# Criando um array NumPy com list comprehension
array2 = np.array([x ** 3 for x in range(15)])
# ->
array2
# ->
# Criando um array NumPy com list comprehension
array2 = np.array([True for x in range(15)])
# ->
array2
# ->
# Usando um array como índice para outro array
array1[array2]
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## NumPy
NumPy oferece sequências de elementos que são mais eficientes que sequências padrões em Python.
# ->
import sys
import numpy as np
print(sys.version)
np.__version__
## Concatenando Arrays
# ->
array1 = np.ones(4)
# ->
array2 = np.arange(15)
# ->
array_conc = np.concatenate((array1, array2))
# ->
array_conc
## Joining Arrays
# ->
a = np.ones((3,3))
b = np.zeros((3,3))
# ->
print(a)
# ->
print(b)
# ->
# ?np.vstack
# ->
# ?np.hstack
# ->
# Primeiro preenche as linhas
np.vstack((a,b))
# ->
# Primeiro preenche as colunas
np.hstack((a,b))
# ->
# Joining arrays
a = np.array([0, 1, 2])
b = np.array([3, 4, 5])
c = np.array([6, 7, 8])
# ->
# ?np.column_stack
# ->
# ?np.row_stack
```

```
# ->
np.column_stack((a, b, c))
# ->
np.row_stack((a, b, c))
## Split Arrays
# ->
array3 = np.arange(16).reshape((4,4)); array3
# ->
[array3_p1, array3_p2] = np.hsplit(array3, 2)
# ->
array3_p1
# ->
array3_p2
# ->
[array3_p1, array3_p2] = np.vsplit(array3, 2)
# ->
array3_p1
# ->
array3_p2
## Gravando e Carragendo Dados com NumPy
# ->
dados = array3
# ->
np.save('dados_saved_v1', dados)
# ->
loaded_data = np.load('dados_saved_v1.npy')
# ->
loaded_data
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## NumPy
### Arrays NumPy - objetos multidimensionais com N dimensões.
### Matrizes NumPy - objetos com 2 dimensões.
A principal vantagem de utilizar matrizes com NumPy, é que este tipo de objeto possui convenientes notações para multiplicação de matrizes.
Arrays e Matrizes NumPy possuem o atributo .T para retornar a transposta da matriz, enquanto objetos do tipo matriz possuem adicionalmente os atributos .I (Inversa) e .H (Transposta Conjugada).
Existem diferenças em operações de álgebra linear entre arrays e matrizes.
Muitas funções NumPy retornam arrays e não matrizes como objeto resultante.
# ->
import sys
import numpy as np
print(sys.version)
np.__version__
## Criando Matrizes
# ->
mat1 = np.matrix("1,2,3; 4,5,6")
print(mat1)
# ->
type(mat1)
# ->
mat2 = np.matrix( [ [1,2,3], [4,5,6] ] )
print(mat2)
# ->
mat3 = np.matrix([ [0,10,0,0,0], [0,0,20,0,0], [0,0,0,30,0], [0,0,0,0,40], [0,0,0,0,0] ])
print(mat3)
# ->
type(mat3)
# ->
mat3[2, 3]
## Operações Matemáticas com Arrays e Matrizes
# ->
a = np.array([[1,2],[3,4]])
a
# ->
type(a)
# ->
a * a
# ->
A = np.mat(a)
A
# ->
type(A)
# ->
A * A
```

```
## $$  \boxed{ \begin{align} \begin{pmatrix} 1 & 2    \\ 3 &  4   \end{pmatrix} & \begin{pmatrix} 1 & 2    \\ 3 &  4   \end{pmatrix}  = \
begin{pmatrix} 7 & 10    \\ 15 &  22   \end{pmatrix} \end{align} }$$
# ->
# from IPython.display import Image
# Image('imagens/Matriz.png') # importa imagem para jupyter notebook
# ->
np.dot(a, a) #dot product (multiplicacao de matrizes)
# ->
# Convertendo um Array para Matriz
mat5 = np.asmatrix(a)
# ->
mat5
# ->
mat5 * mat5
# ->
# Convertendo uma Matriz para Array
array2 = np.asarray(mat5)
# ->
array2
# ->
array2 * array2
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Vetorização
Embora possamos usar list comprehension e função map em arrays numpy, este pode não ser o melhor método e a maneira mais eficiente de se obter o mesmo resultado seria através de vetorização.
Vetorização nos permite aplicar uma função a um array inteiro, ao invés de aplicar a função elemento a elemento (similar ao que fazemos com as funções map() e filter()).
Ao trabalhar com objetos NumPy e Pandas, existem maneiras mais eficientes de se aplicar uma função a um conjunto de elementos, que serão mais velozes que a aplicação de loops for.
# ->
import numpy as np
# ->
array1 = np.random.randint(0, 50, 20)
# ->
array1
# ->
# Criando um função
def calc_func(num):
    if num < 10:
        return num ** 3
    else:
        return num ** 2
# ->
# Para que a função funcione no objeto array do NumPy, ela precisa ser vetorizada
calc_func(array1)
# ->
#?np.vectorize
# ->
# Vetorizando a função
v_calc_func = np.vectorize(calc_func)
# ->
type (v_calc_func)
# ->
# Aplicando a função vetorizada ao array3 NumPy
v_calc_func(array1)
# ->
# Aplicando a função map() sem vetorizar a função
list(map(calc_func, array1))
# ->
# Podemos usar list comprehension para obter o mesmo resutado, sem vetorizar a função
[calc_func(x) for x in array1]
No Python 3, a list comprehension recebeu atualizações e ficou muito mais rápida e eficiente, uma vez que ela é amplamente utilizada em programação Python. Lembre-se sempre de checar a documentação antes de decidir como você irá manipular suas estruturas de dados.
# ->
# Função vetorizada
%timeit v_calc_func(array1)
# List comprehension
%timeit [calc_func(x) for x in array1]
# Função map()
%timeit list(map(calc_func, array1))
# ->
# Criando um array com valores maiores
array2 = np.random.randint(0, 100, 20 * 10000)
```

```
# ->
# Função vetorizada
%timeit v_calc_func(array2)
# List comprehension
%timeit [calc_func(x) for x in array2]
# Função map()
%timeit list(map(calc_func, array2))
Utilizar as versões mais recentes de um software pode trazer problemas de compatibilidade com aplicações existentes, mas é grande a possibilidade trazerem melhorias em performance e novas funcionalidades.
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Pandas - Series
# ->
import sys
import pandas as pd
import numpy as np
print(sys.version)
pd.__version__
# Se precisar instalar uma versão específica o pandas, use:
# !pip uninstall pandas
# !pip install pandas == 0.23.4
## Operações com Series
# ->
serie1 = pd.Series(np.arange(26))
# ->
serie1
# ->
serie1.index
# ->
import string
# ->
lcase = string.ascii_lowercase
ucase = string.ascii_uppercase
print(lcase, ucase)
# ->
lcase = list(lcase)
ucase = list(ucase)
print(lcase)
print(ucase)
# ->
serie1.index = lcase
# ->
serie1
# ->
serie1['f':'r']
# ->
serie1['f']
## Aplicando Funções em Series
# ->
import matplotlib.pyplot as plt
%matplotlib inline
# ->
np.random.seed(784)
# ->
array1 = np.random.randint(1, 30, 40)
# ->
array1
# ->
dados = pd.Series(array1)
# ->
dados
# ->
dados = pd.Series(array1, dtype = np.float16)
# ->
dados
# ->
dados.mean() # média
# ->
dados.median() # mediana
# ->
dados.mode() # moda
# ->
dados.unique() # retorna dados que são únicos
# ->
```

```
dados.value_counts() # conta informacoes
# ->
dados.describe() # resumo estatístico
# ->
# dados.hist() # historgrama
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Pandas - Series
## Valores NaN (Not a Number)
# ->
import sys
import pandas as pd
import numpy as np
print(sys.version)
pd.__version__
# Se precisar instalar uma versão específica o pandas, use:
# !pip uninstall pandas
# !pip install pandas == 0.23.4
# ->
# Usando NumPy
array1 = np.array([1, 2, 3, np.nan])
# ->
array1
# ->
array1.mean() # numpy mostra que existe valores NaN
# ->
# Usando Pandas
serie2 = pd.Series([1, 2, 3, np.nan])
# ->
serie2
# ->
serie2.mean() # pandas desconsidera valores NaN
## Concatenando as Séries
# ->
np.random.seed(567)
# ->
serie3 = pd.Series(np.random.rand(5))
# ->
serie3
# ->
serie4 = pd.Series(np.random.rand(5))
# ->
serie4
# ->
combo = pd.concat([serie3, serie4])
# ->
combo
# ->
combo[0]
# ->
combo.index = range(combo.count()) # refaz índices a partir de uma lista de elementos
# ->
combo
# ->
combo.reindex([0, 2, 16, 21]) # refaz índices
# ->
combo.reindex([0, 2, 16, 21], fill_value = 0)
# ->
new_combo = combo.reindex([0, 2, 16, 21])
# ->
new_combo
# ->
new_combo.ffill() # ffill preenche para frente
# ->
new_combo.bfill() # bfill preenche para trás
# ->
new_combo.fillna(12) # fillna preenche valores na
## Operações com Series - Índices Diferentes
# ->
s1 = pd.Series(np.random.randn(5)); s1
# ->
s2 = pd.Series(np.random.randn(5)); s2
# ->
s1 + s2
```

```
# ->
s2.index = list(range(3,8)); s2
# ->
s1 + s2
# ->
# soma de series, refazendo índices e preenchendo valores NA com zero
s1.reindex(range(10), fill_value = 0) + s2.reindex(range(10), fill_value = 0)
# ->
s1 = pd.Series(range(1, 4), index = ['a','a','c']); s1
# ->
s2 = pd.Series(range(1,4), index = ['a','a','b']); s2
# ->
s1 * s2
# ->
s1 + s2
## Copiando uma Serie
# ->
s1_copy = s1.copy()
# ->
s1_copy['a'] = 3
# ->
s1_copy
# ->
s1
## Função map() do Pandas
# ->
# Função map() - versão do Pandas
s1.map(lambda x: x ** 2)
# ->
s1.map({1:2,2:3,3:12})
# ->
s1.map({2:3,3:12})
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## SQL Join e Junção de Tabelas com Pandas
# ->
# Imports
import pandas as pd
# ->
# from IPython.display import Image
# Image(url = 'imagens/sql-join.png')
# ->
# Dados originais
dados1 = {
      'disciplina_id': ['1', '2', '3', '4', '5'],
      'nome': ['Bernardo', 'Alan', 'Mateus', 'Ivo', 'Gerson'],
      'sobrenome': ['Anderson', 'Teixeira', 'Amoedo', 'Trindade', 'Vargas']}
# Criação do dataframe
df_a = pd.DataFrame(dados1, columns = ['disciplina_id', 'nome', 'sobrenome'])
df_a
# ->
# Dados originais
dados2 = {
      'disciplina_id': ['4', '5', '6', '7', '8'],
      'nome': ['Roberto', 'Mariana', 'Ana', 'Marcos', 'Maria'],
      'sobrenome': ['Sampaio', 'Fernandes', 'Arantes', 'Menezes', 'Martins']}
# Criação do dataframe
df_b = pd.DataFrame(dados2, columns = ['disciplina_id', 'nome', 'sobrenome'])
df_b
# ->
# Dados originais
dados3 = {
      'disciplina_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
      'teste_id': [81, 75, 75, 71, 76, 84, 95, 61, 57, 90]}
# Criação do dataframe
df_n = pd.DataFrame(dados3, columns = ['disciplina_id','teste_id'])
df_n
# ->
# Join dos dataframes pelas linhas
df_new = pd.concat([df_a, df_b])
df_new
# ->
# Join dos dataframes pelas colunas
pd.concat([df_a, df_b], axis=1)
```

```
# ->
# Join de dois dataframes pela coluna disciplina_id
pd.merge(df_new, df_n, on = 'disciplina_id')
# ->
# Join de dois dataframes pela coluna disciplina_id (igual ao item anterior)
pd.merge(df_new, df_n, left_on = 'disciplina_id', right_on = 'disciplina_id')
# ->
# Merge outer join
# “A união externa completa produz o conjunto de todos os registros na Tabela A e na Tabela B,
# com registros correspondentes de ambos os lados, quando disponíveis.
# Se não houver correspondência, o lado ausente conterá null.”
pd.merge(df_a, df_b, on = 'disciplina_id', how = 'outer')
# ->
# Merge inner join
# “A junção interna produz apenas o conjunto de registros que correspondem na Tabela A e na Tabela B.”
pd.merge(df_a, df_b, on = 'disciplina_id', how = 'inner')
# ->
# Merge left join
# "Junção externa esquerda produz um conjunto completo de registros da Tabela A,
# com os registros correspondentes (quando disponíveis) na Tabela B.
# Se não houver correspondência, o lado direito conterá nulo."
pd.merge(df_a, df_b, on = 'disciplina_id', how = 'left')
# ->
# Merge right join (contrário do item anterior)
pd.merge(df_a, df_b, on = 'disciplina_id', how = 'right')
# ->
# Adicionando um sufixo para identificar os nomes das colunas
pd.merge(df_a, df_b, on = 'disciplina_id', how = 'left', suffixes = ('_left', '_right'))
# ->
# Join baseado em índices
pd.merge(df_a, df_b, right_index = True, left_index = True)
## FIM
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Estudo de Caso - Análise Exploratória de Dados - Parte 1
Analisando dados de aluguel de bikes como táxis na cidade de New York.
# ->
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# permite abrir os gráficos nas células do jupyter notebook:
# %matplotlib inline
# ->
# Importando o arquivo csv
df = pd.read_csv('data/taxis_bikes_nycity.csv')
# ->
print(type(df))
# ->
df.head(10)
# ->
df.dtypes
# ->
df.columns
# ->
df.index
# ->
df['Data'].head()
# ->
# recarregando o dataframe, mas alterando o tipo de dado da coluna Data para data com parse_dates
df = pd.read_csv('data/taxis_bikes_nycity.csv', parse_dates = ['Data'])
# ->
df['Data'].head()
# ->
df.set_index('Data', inplace = True) # transformando a coluna Data em índice
# ->
df.head(10)
# ->
# df.plot()
# ->
# ?df.plot
# ->
# df.plot(kind = 'bar')
# plt.show()
# ->
```

```
# df.plot(kind = 'area')
# plt.ylabel("Count")
# ->
df.describe()
# ->
df['2015-11']
# ->
len(df['2015-11'])
# ->
df.to_csv('data/dataframe_saved_v1.csv') # salvando dataframe como arquivo csv
## FIM
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Estudo de Caso - Análise Exploratória de Dados - Parte 2
# ->
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
# ->
pd.__version__
# ->
# Carregando os dados salvos em disco
df = pd.read_csv('data/dataframe_saved_v1.csv')
# ->
df.head()
# ->
df.dtypes
# ->
df = pd.read_csv('data/dataframe_saved_v1.csv', parse_dates = ['Data'])
# ->
df.head()
# ->
df.dtypes
# ->
cols = ['Data', 'Distancia', 'Tempo']
df.columns = cols
df.head()
# ->
# Transformando a coluna Data em índice
df.set_index('Data', inplace = True)
# ->
df.head()
# ->
# df.plot()
# ->
# Função para converter a coluna de duração no tempo em segundos
def calcula_total_segundos(time):
    if time is np.nan:
        return np.nan
    hrs, mins, seconds = str(time).split(':')
    seconds = int(seconds) + 60 * int(mins) + 60 * 60 * int(hrs)
    return seconds
# ->
df['Segundos'] = df.Tempo.map(calcula_total_segundos)
# ->
df.head(10)
# ->
df.describe()
# ->
df.fillna(0).describe()
# ->
df['Minutos'] = df['Segundos'].map(lambda x: x / 60)
# ->
df.fillna(0).describe()
# ->
# df.plot(x = 'Distancia', y = 'Minutos', kind = 'scatter')
# ->
df.corr()
# ->
df.corr(method = 'spearman')
# ->
df.corr(method = 'kendall')
# ->
# df.boxplot('Minutos', return_type = 'axes')
```

```
# ->
df['Min_Por_Km'] = df['Minutos'] / df['Distancia']
# ->
df.fillna(0).describe()
# ->
# df.hist('Min_Por_Km')
# ->
# df.hist('Min_Por_Km', bins = 20)
# ->
# df.hist('Min_Por_Km', bins = 20, figsize = (10, 8))
# plt.xlim((5, 11))
# plt.ylim((0, 12))
# plt.title("Histograma Minutos Por Km")
# plt.grid(False)
# plt.savefig('imagens/hist_minutos_por_km.png')
# ->
# df['Distancia'].plot()
# ->
df.head(15)
# ->
# Calculando a média de distância em uma janela (window) de 2 horas
# df['Distancia'].plot()
# pd.Series(df['Distancia']).rolling(window = 2).mean().plot()
# ->
# Calculando a soma de distância em uma janela (window) de 2 horas
# df['Distancia'].plot()
# pd.Series(df['Distancia']).rolling(window = 2).sum().plot()
# ->
df.index
# ->
df['2015-11':'2015-12']
# ->
# df['2015-11':'2016-1-1']['Distancia'].plot()
# ->
df.loc['2015-8-12']
# ->
df.to_csv('data/dataframe_saved_v2.csv')
# ->
df.reset_index()
## FIM
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Configuração e Customização Avançada do Matplotlib
# ->
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Image
%matplotlib inline
mpl.__version__
# ->
print(plt.style.available)
# ->
def cria_plot ():
    x = np.random.randn(5000, 6)
    (figure, axes) = plt.subplots(figsize = (16,10))
    (n, bins, patches) = axes.hist(x, 12,
                                   density = 1,
                                   histtype = 'bar',
                                   label = ['Color 1', 'Color 2', 'Color 3', 'Color 4', 'Color 5', 'Color 6'])
    axes.set_title("Histograma\nPara\nDistribuição Normal", fontsize = 25)
    axes.set_xlabel("Dados", fontsize = 16)
    axes.set_ylabel("Frequência", fontsize = 16)
    axes.legend()
    plt.show()
# ->
cria_plot()
# ->
# Usuários Windows utilizem:
#!dir estilos
!ls -l estilos
# ->
# Usuários Windows utilizem:
# !type estilos/personalestilo-1.mplstyle
```

```
!cat estilos/personalestilo-1.mplstyle
# ->
plt.style.use("estilos/personalestilo-1.mplstyle")
# ->
cria_plot()
### Subplots
Utilizaremos o dataset sobre automóveis do repositório de Machine Learning da UCI: [UCI Machine Learning
Repository](http://archive.ics.uci.edu/ml/index.html)
[Automobile Data Set](https://archive.ics.uci.edu/ml/datasets/Automobile)
## Usando o Pandas para Carregar os Dados
# ->
import sys
sys.path.append("lib")
import geradados, geraplot, radar
# ->
dados = geradados.get_raw_data()
dados.head()
# ->
dados_subset = geradados.get_limited_data()
dados_subset.head()
# ->
geradados.get_all_auto_makes()
# ->
(fabricantes, total) = geradados.get_make_counts(dados_subset)
total
# ->
dados = geradados.get_limited_data(lower_bound = 6)
dados.head()
# ->
len(dados.index)
## Normalizando os Dados
# ->
dados_normalizados = dados.copy()
dados_normalizados.rename(columns = {"horsepower": "power"}, inplace = True)
# ->
dados_normalizados.head()
# ->
# Valores mais altos para estas variáveis
geradados.norm_columns(["city mpg", "highway mpg", "power"], dados_normalizados)
dados_normalizados.head()
# ->
# Valores mais baixos para estas variáveis
geradados.invert_norm_columns(["price", "weight", "riskiness", "losses"], dados_normalizados)
dados_normalizados.head()
## Plots
# ->
figure = plt.figure(figsize = (15, 5))
prices_gs = mpl.gridspec.GridSpec(1, 1)
prices_axes = geraplot.make_autos_price_plot(figure, prices_gs, dados)
plt.show()
# ->
figure = plt.figure(figsize = (15, 5))
mpg_gs = mpl.gridspec.GridSpec(1, 1)
mpg_axes = geraplot.make_autos_mpg_plot(figure, mpg_gs, dados)
plt.show()
# ->
figure = plt.figure(figsize = (15, 5))
risk_gs = mpl.gridspec.GridSpec(1, 1)
risk_axes = geraplot.make_autos_riskiness_plot(figure, risk_gs, dados_normalizados)
plt.show()
# ->
figure = plt.figure(figsize=(15, 5))
loss_gs = mpl.gridspec.GridSpec(1, 1)
loss_axes = geraplot.make_autos_losses_plot(figure, loss_gs, dados_normalizados)
plt.show()
# ->
figure = plt.figure(figsize = (15, 5))
risk_loss_gs = mpl.gridspec.GridSpec(1, 1)
risk_loss_axes = geraplot.make_autos_loss_and_risk_plot(figure, risk_loss_gs, dados_normalizados)
plt.show()
# ->
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import Image
import warnings
#warnings.filterwarnings('ignore')
```

```
import matplotlib
%matplotlib inline
import sys
sys.path.append("lib")
import geradados, geraplot, radar
#plt.style.use("estilos/personalestilo-1.mplstyle")
dados = geradados.get_raw_data()
dados.head()
dados_subset = geradados.get_limited_data()
dados_subset.head()
dados = geradados.get_limited_data(lower_bound = 6)
dados.head()
dados_normalizados = dados.copy()
dados_normalizados.rename(columns = {"horsepower": "power"}, inplace = True)
figure = plt.figure(figsize = (15, 5))
radar_gs = mpl.gridspec.GridSpec(3, 7,
                                 height_ratios = [1, 10, 10],
                                 wspace = 0.50,
                                 hspace = 0.60,
                                 top = 0.95,
                                 bottom = 0.25)
radar_axes = geraplot.make_autos_radar_plot(figure, gs=radar_gs, pddata=dados_normalizados)
plt.show()
## Plots Combinados
wireframe
-------------------------------------------------
|              overall title                    |
-------------------------------------------------
|              price ranges                     |
-------------------------------------------------
| combined loss/risk |                          |
|                    |        radar             |
----------------------         plots             |
|  risk   |   loss   |                          |
-------------------------------------------------
|                  mpg                          |
-------------------------------------------------
# ->
# Construindo as camadas (sem os dados)
figure = plt.figure(figsize=(10, 8))
gs_master = mpl.gridspec.GridSpec(4, 2, height_ratios=[1, 2, 8, 2])
# Camada 1 - Title
gs_1 = mpl.gridspec.GridSpecFromSubplotSpec(1, 1, subplot_spec=gs_master[0, :])
title_axes = figure.add_subplot(gs_1[0])
# Camada 2 - Price
gs_2 = mpl.gridspec.GridSpecFromSubplotSpec(1, 1, subplot_spec=gs_master[1, :])
price_axes = figure.add_subplot(gs_2[0])
# Camada 3 - Risks & Radar
gs_31 = mpl.gridspec.GridSpecFromSubplotSpec(2, 2, height_ratios=[2, 1], subplot_spec=gs_master[2, :1])
risk_and_loss_axes = figure.add_subplot(gs_31[0, :])
risk_axes = figure.add_subplot(gs_31[1, :1])
loss_axes = figure.add_subplot(gs_31[1:, 1])
gs_32 = mpl.gridspec.GridSpecFromSubplotSpec(1, 1, subplot_spec=gs_master[2, 1])
radar_axes = figure.add_subplot(gs_32[0])
# Camada 4 - MPG
gs_4 = mpl.gridspec.GridSpecFromSubplotSpec(1, 1, subplot_spec=gs_master[3, :])
mpg_axes = figure.add_subplot(gs_4[0])
# Une as camas, ainda sem dados
gs_master.tight_layout(figure)
plt.show()
# ->
# Construindo as camadas (com os dados)
figure = plt.figure(figsize = (15, 15))
gs_master = mpl.gridspec.GridSpec(4, 2,
                                  height_ratios = [1, 24, 128, 32],
                                  hspace = 0,
                                  wspace = 0)
# Camada 1 - Title
gs_1 = mpl.gridspec.GridSpecFromSubplotSpec(1, 1, subplot_spec = gs_master[0, :])
title_axes = figure.add_subplot(gs_1[0])
title_axes.set_title("Plots", fontsize = 30, color = "#cdced1")
geraplot.hide_axes(title_axes)
# Camada 2 - Price
gs_2 = mpl.gridspec.GridSpecFromSubplotSpec(1, 1, subplot_spec = gs_master[1, :])
price_axes = figure.add_subplot(gs_2[0])
geraplot.make_autos_price_plot(figure,
                               pddata = dados,
                               axes = price_axes)
```

```
# Camada 3, Part I - Risks
gs_31 = mpl.gridspec.GridSpecFromSubplotSpec(2, 2,
                                             height_ratios = [2, 1],
                                             hspace = 0.4,
                                             subplot_spec = gs_master[2, :1])
risk_and_loss_axes = figure.add_subplot(gs_31[0, :])
geraplot.make_autos_loss_and_risk_plot(figure,
                                       pddata = dados_normalizados,
                                       axes = risk_and_loss_axes,
                                       x_label = False,
                                       rotate_ticks = True)
risk_axes = figure.add_subplot(gs_31[1, :1])
geraplot.make_autos_riskiness_plot(figure,
                                   pddata = dados_normalizados,
                                   axes = risk_axes,
                                   legend = False,
                                   labels = False)
loss_axes = figure.add_subplot(gs_31[1:, 1])
geraplot.make_autos_losses_plot(figure,
                                pddata = dados_normalizados,
                                axes = loss_axes,
                                legend = False,
                                labels = False)
# Camada 3, Part II - Radar
gs_32 = mpl.gridspec.GridSpecFromSubplotSpec(5, 3,
                                             height_ratios = [1, 20, 20, 20, 20],
                                             hspace = 0.6,
                                             wspace = 0,
                                             subplot_spec = gs_master[2, 1])
(rows, cols) = geometry = gs_32.get_geometry()
title_axes = figure.add_subplot(gs_32[0, :])
inner_axes = []
projection = radar.RadarAxes(spoke_count = len(dados_normalizados.groupby("make").mean().columns))
[inner_axes.append(figure.add_subplot(m, projection = projection)) for m in [n for n in gs_32][cols:]]
geraplot.make_autos_radar_plot(figure,
                               pddata = dados_normalizados,
                               title_axes = title_axes,
                               inner_axes = inner_axes,
                               legend_axes = False,
                               geometry = geometry)
# Camada 4 - MPG
gs_4 = mpl.gridspec.GridSpecFromSubplotSpec(1, 1, subplot_spec = gs_master[3, :])
mpg_axes = figure.add_subplot(gs_4[0])
geraplot.make_autos_mpg_plot(figure,
                             pddata = dados,
                             axes = mpg_axes)
# Unindo as camadas
gs_master.tight_layout(figure)
plt.show()
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Gráficos Estatísticos com Seaborn
https://seaborn.pydata.org/
# ->
!pip install seaborn
# ->
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import numpy as np
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
# ->
sns.__version__
# ->
# Datasets importados junto com o Seaborn
sns.get_dataset_names()
# ->
# Carregando um dataset
iris = sns.load_dataset("iris")
# ->
type(iris)
# ->
```

```
iris.head()
# ->
iris.describe()
# ->
iris.columns
## Construindo Gráficos Estatísticos
# ->
# Histograma com estimativa de densidade de kernel - Distribuição univariada
sns.distplot(iris.sepal_length, rug = True, fit = stats.gausshyper);
# ->
# Scatterplot - Distribuição bivariada
sns.jointplot(x = "sepal_length", y = "petal_length", data = iris);
# ->
# Gráfico útil quando estiver trabalhando com grandes conjuntos de dados - Distribuição bivariada
with sns.axes_style("white"):
    sns.jointplot(x = "sepal_length", y = "petal_length", data = iris, kind = "hex", color  ="k");
# ->
# Distribuição bivariada
sns.jointplot(x = "sepal_length", y = "petal_length", data = iris, kind = "kde");
# ->
# Distribuição bivariada
g = sns.jointplot(x = "sepal_length", y = "petal_length", data = iris, kind = "kde", color = "m")
g.plot_joint(plt.scatter, c = "w", s = 30, linewidth = 1, marker = "+")
g.ax_joint.collections[0].set_alpha(0);
# ->
# Plot para distirbuições bi-variadas
sns.pairplot(iris);
## Visualização de Relacionamento Linear
# ->
# Carregando o dataset tips
tips = sns.load_dataset("tips")
# ->
type(tips)
# ->
tips.head()
# ->
tips.describe()
# ->
# Scatterplot com linha de regressão - Distribuição bivariada
sns.jointplot(x = "total_bill", y = "tip", data = tips, kind = "reg");
# ->
# Regressão Linear (utiliza 95% de intervalo de confiança por padrão)
# tip - variável dependente
# total_bill - variável independente
sns.lmplot(x = "total_bill", y = "tip", data = tips);
# ->
# Alterando a variável independente
sns.lmplot(x = "size", y = "tip", data = tips, x_jitter = .05);
# ->
# Visualizando o efeito de diferentes tamanho de tips
sns.lmplot(x = "size", y = "tip", data = tips, x_estimator = np.mean);
# ->
# Carregando o dataset anscombe
anscombe = sns.load_dataset("anscombe")
# ->
# Relacionamento não-linear
sns.lmplot(x = "x", y = "y", data = anscombe.query("dataset == 'II'"), ci = None, scatter_kws = {"s": 80});
# ->
# Podemos ajustar os parâmetros para se adequarem a curva
sns.lmplot(x = "x", y = "y", data = anscombe.query("dataset == 'II'"), order = 2, ci = None, scatter_kws = {"s": 80});
# ->
## Visualizando outliers
sns.lmplot(x = "x", y = "y", data = anscombe.query("dataset == 'III'"), ci = None, scatter_kws = {"s": 80});
# ->
## Usando o lowess smoother para variáveis com relacionamento não linear.
sns.lmplot(x = "total_bill", y = "tip", data = tips, lowess = True);
# ->
# Usando mais de 2 variáveis
sns.lmplot(x = "total_bill", y = "tip", hue = "smoker", data = tips);
# ->
# Alterando a configuração do gráfico
sns.lmplot(x = "total_bill", y = "tip", hue = "smoker", data = tips, markers = ["o", "x"], palette = "Set1");
# ->
# Dividindo a área de desenho
sns.lmplot(x = "total_bill", y = "tip", hue = "smoker", col = "time", data = tips);
# ->
# Dividindo a área de desenho
sns.lmplot(x = "total_bill", y = "tip", hue = "smoker", col = "time", row = "sex", data = tips);
```

```
# ->
# Dividindo a área de desenho
sns.lmplot(x = "total_bill", y = "tip", col = "day", data = tips, col_wrap = 2, size = 3);
# ->
# Dividindo a área de desenho
sns.lmplot(x = "total_bill", y = "tip", col = "day", data = tips, aspect = .5);
## Gráficos para Variáveis Categóricas
# ->
# stripplot
sns.stripplot(x = "day", y = "total_bill", data = tips);
# ->
# stripplot
sns.stripplot(x = "day", y = "total_bill", data = tips, jitter = True);
# ->
# swarmplot - Evitando overlap dos pontos
sns.swarmplot(x = "day", y = "total_bill", data = tips);
# ->
# boxplot
sns.boxplot(x = "day", y = "total_bill", hue = "time", data = tips);
# ->
# boxplot
sns.boxplot(data = iris, orient = "h");
# ->
# violinplot
sns.violinplot(x = "total_bill", y = "day", hue = "time", data = tips);
# ->
# violinplot
sns.violinplot(x = "total_bill", y = "day", hue = "time", data = tips, bw = .1, scale = "count", scale_hue = False);
# ->
 # violinplot
sns.violinplot(x = "day", y = "total_bill", hue = "sex", data = tips, split = True);
# ->
# barplot
sns.barplot(x = "day", y = "total_bill", hue = "sex", data = tips);
# ->
# countplot
sns.countplot(x = "day", data = tips, palette = "Greens_d");
# ->
# countplot
sns.countplot(y = "day", hue = "sex", data = tips, palette = "Greens_d");
# ->
# countplot
f, ax = plt.subplots(figsize=(7, 3))
sns.countplot(y = "day", data = tips, color = "c");
# ->
# pointplot
sns.pointplot(x = "sex", y = "total_bill", hue = "smoker", data = tips);
# ->
# factorplot
sns.factorplot(x = "day", y = "total_bill", hue = "smoker", data = tips);
## Visualizando DataFrames Pandas com Seaborn
# ->
import random
import pandas as pd
# ->
df = pd.DataFrame()
# ->
df['x'] = random.sample(range(1, 100), 25)
df['y'] = random.sample(range(1, 100), 25)
# ->
df.head()
# ->
# Scatterplot
sns.lmplot('x', 'y', data = df, fit_reg = False)
# ->
# Density Plot
sns.kdeplot(df.y)
# ->
# Distplot
sns.distplot(df.x)
# ->
# Histograma
plt.hist(df.x, alpha = .3)
sns.rugplot(df.x);
# ->
# Boxplot
sns.boxplot([df.y, df.x])
# ->
```

```
# Heatmap
sns.heatmap([df.y, df.x], annot = True, fmt = "d")
# ->
# Clustermap
sns.clustermap(df)
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Gráficos em Python com ggplot
# ->
!pip install ggplot
# ->
import warnings
warnings.filterwarnings("ignore")
from ggplot import *
# Caso receba mensagem de erro nesta célula, faça o seguinte:
# Para resolver o problema é preciso editar o arquivo:
# anaconda3/lib/python3.7/site-packages/ggplot/stats/smoothers.py
# Ao abrir o arquivo é preciso mudar o comando:
# from pandas.lib import Timestamp
# para:
# from pandas import Timestamp
# E Substitua pd.tslib.Timestamp por pd.Timestamp
# Para resolver o problema é preciso também editar o arquivo:
# /anaconda3/lib/python3.7/site-packages/ggplot/utils.py
# Substitua pd.tslib.Timestamp por pd.Timestamp
# ->
print (ggplot(mtcars, aes('mpg', 'qsec')) + \
      geom_point(colour = 'steelblue') + \
      scale_x_continuous(breaks = [10, 20, 30],  \
                         labels = ["Ruim", "Pk", "Bom"]))
# ->
print (ggplot(meat, aes('date','beef')) + \
      geom_line(color = 'black') + \
      scale_x_date(breaks = date_breaks('7 years'), labels = '%b %Y') + \
      scale_y_continuous(labels = 'comma'))
# ->
import matplotlib.pyplot as plt
from matplotlib.dates import YearLocator, DateFormatter
from ggplot import meat
tick_every_n = YearLocator(7)
date_formatter = DateFormatter('%b %Y')
x = meat.date
y = meat.beef
fig, ax = plt.subplots()
ax.plot(x, y, 'black')
ax.xaxis.set_major_locator(tick_every_n)
ax.xaxis.set_major_formatter(date_formatter)
fig.autofmt_xdate()
plt.show()
# ->
print (ggplot(meat, aes('date','beef * 2000')) + \
      geom_line(color = 'coral') + \
      scale_x_date(breaks = date_breaks('36 months'), labels = '%Y') + \
      scale_y_continuous(labels = 'millions'))
# Fim
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

Cap04:

```
 # <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 3</font>
## Solução - Exercícios
Nesta lista de exercícios você vai continuar o trabalho que fizemos no Estudo de Caso neste capítulo. O código para carregar o dataset já está disponível para você. Leia cada exercício atentamente e coloque em prática suas habilidades analíticas. Todas as questões podem ser resolvidas com tudo já estudado até aqui no curso e consulte a documentação Python se necessário!
# ->
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#plt.style.use('seaborn-poster')
%matplotlib inline
# ->
plt.style.use("classic")
# ->
print(plt.style.available)
# ->
# parse_dates converte para formato de data
df = pd.read_csv('data/dataframe_saved_v2.csv', parse_dates = ['Data'], usecols = list(range(0,6)))
# ->
df.dtypes
# ->
df.sort_index(inplace = True)
df.head()
# ->
# Exercício 1 - Qual o valor máximo da coluna Minutos?
df.Minutos.max()
# ->
# Exercício 2 - Qual o valor mínimo de distância acima de 2.0?
df.Distancia[df.Distancia > 2.0].min() # valor mínimo com filtro de pesquisa
# ->
# Exercício 3 - Crie um plot com a frequência acumulada da coluna Distancia.
df['Distancia'].hist(bins = 30, cumulative = True)
# ->
# ou:
df.Distancia.cumsum().plot()
plt.xlabel("Dia")
plt.ylabel("Distancia")
# ->
# Exercício 4 - Qual o dia da semana no índice de posição zero?
df.Data[0].strftime("%A") # strftime é uma função que extrai data
# ->
# Exercício 5 - Qual o dia da semana nos índices nas 5 primeiras posições?
df.Data.map(lambda x: x.strftime("%A")).head()
# ->
# Exercício 6 - Extraia todos os dias da semana (em formato texto) e insira em uma nova coluna no dataframe df.
df['Dia_Semana'] = df.Data.map(lambda x: x.strftime("%A"))
# ->
df.head(10)
# ->
# Exercício 7 - Crie um gráfico de barras com o total da distância percorrida em cada dia da semana.
df[['Distancia', 'Dia_Semana']].groupby('Dia_Semana').sum().plot(kind = 'bar')
# ->
# Exercício 8 - Delete a coluna Tempo do dataframe df.
del(df['Tempo'])
# ->
df.head()
# ->
# Exercício 9 - Qual o total de corridas de taxi por dia da semana?
df['Distancia'] = df.Distancia[df.Distancia > 0]
dias = df.groupby('Dia_Semana')
dias.size()
# ->
# Exercício 10 - Qual a média para cada uma das colunas por dia da semana?
dias.mean()
## FIM
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 4</font>
## Distribuição de Frequência
Um professor coletou as notas de todos os alunos na avaliação final de um curso de Matemática. O professor gostaria de tabular os dados a fim de extrair insights e compreender como os dados estão organizados. Crie uma distribuição de frequência para ajudar o professor.
```

```
# ->
# Imports
import pandas as pd
# ->
# Lista de notas de alunos na avaliação final do curso de Matemática
notas_alunos = [92, 81, 100, 92, 71, 73, 100, 92, 84, 100, 73, 92, 73, 84, 92, 92, 92, 84, 73, 100]
# ->
len(notas_alunos)
## Solução com Series em Pandas
# ->
# Série com notas e frequências
# value_counts() conta quantas vezes cada serie aparece no df
# reset_index() coloca os indices na sequencia correta
df1 = pd.Series(notas_alunos).value_counts().reset_index().sort_values('index').reset_index(drop=True)
df1.columns = ['Nota', 'Frequencia'] # renomeando as colunas do df
print(df1)
# ->
# Série com notas e frequências, alterando a quantidade de classes
# value_counts(bins=3) agrupa por 3 faixas de notas
df1 = pd.Series(notas_alunos).value_counts(bins=3).reset_index().sort_values('index').reset_index(drop=True)
df1.columns = ['Nota', 'Frequencia']
print(df1)
## Solução com Dataframes em Pandas
### Distribuição de Frequência Simples ou Absoluta - fi
São os valores que representam o número de dados de cada classe. A soma das frequências simples é igual ao número total dos dados.
# ->
df2 = pd.value_counts(notas_alunos).to_frame(name='fi').rename_axis('Nota').sort_index()
print(df2)
### Distribuição de Frequência Relativa Simples -  fri
Permite visualizar os valores das razões entre as Frequências Simples e a Frequência Total.
# ->
df2['fri'] = pd.value_counts(notas_alunos) / len(notas_alunos)
print(df2)
### Distribuição de Frequência Acumulada -  Fi
Permite visualizar o total das Frequências de todos os valores inferiores ao limite superior do intervalo de uma dada classe.
# ->
df2['Fi'] = df2.fi.cumsum()
print(df2)
### Distribuição de Frequência Relativa Acumulada -  Fri
Permite visualizar a frequência acumulada da classe, dividida pela frequência total da distribuição.
# ->
df2['Fri'] = df2.fi.cumsum() / len(notas_alunos)
print(df2)
## FIM
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```

```
# <font color='blue'>Data Science Academy</font>
# <font color='blue'>Big Data Real-Time Analytics com Python e Spark</font>
# <font color='blue'>Capítulo 4</font>
## Estatística Descritiva
# ->
# Imports
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
# ->
# Dataset de notas de alunos
notas = {'idade': [22, 23, 26, 24, 23, 25, 28, 23, 18, 21, 22, 25, 27, 24, 22, 25, 27, 23, 19, 22],
        'teste': [41, 50, 61, 70, 80, 79, 82, 87, 90, 93, 42, 52, 64, 71, 81, 78, 83, 84, 91, 94],
        'prova_final': [45, 57, 68, 81, 81, 82, 85, 92, 93, 98, 46, 58, 69, 80, 83, 84, 87, 94, 95, 97]}
df = pd.DataFrame(notas, columns = ['idade', 'teste', 'prova_final'])
df
# ->
# Descrição dos dados
df['prova_final'].describe() # resumo estatístico
# ->
# Média
df['prova_final'].mean()
# ->
# Mediana
df['prova_final'].median()
# ->
# Moda
from statistics import mode
# https://docs.python.org/3.7/library/statistics.html#module-statistics
mode(df['prova_final'])
# ->
# Contagem
```

```
df['prova_final'].count()
# ->
# Valor mínimo
df['prova_final'].min()
# ->
# Valor máximo
df['prova_final'].max()
# ->
# Variância
df['prova_final'].var()
# ->
# Desvio padrão
df['prova_final'].std()
# ->
# Skewness - coeficiente de assimetria
df['prova_final'].skew()
```
Geralmente, os dados de um determinado conjunto de dados não são distribuídos uniformemente em torno da média de dados em uma curva de distribuição normal. Um conjunto de dados negativamente assimétrico tem sua cauda estendida para a esquerda. É uma indicação de que a média é menor que a moda do conjunto de dados. Em suma, é a medida do grau de assimetria dos dados em torno de sua média.
Um conjunto de dados negativamente assimétrico não tem uma curva de sino. Mais dados são concentrados no lado direito do eixo.
```
# ->
# Histograma da variável prova_final
dados = df['prova_final']
num_bins = 5 # numero de barras
plt.hist(dados, num_bins, facecolor = 'blue', alpha = 0.5)
plt.show()
# ->
# Kurtosis
df['prova_final'].kurt()
```
Uma curtose negativa significa que sua distribuição é mais plana que uma curva normal com a mesma média e desvio padrão.
```
# ->
# Correlação
df.corr()
# ->
# Gráfico de Dispersão
x = df['teste']
y = df['prova_final']
# Plot
plt.scatter(x, y, alpha=0.5)
plt.title('Teste x Prova Final')
plt.xlabel('Teste')
plt.ylabel('Prova Final')
plt.show()
## FIM
### Obrigado - Data Science Academy - <a href="http://facebook.com/dsacademybr">facebook.com/dsacademybr</a>
```