



LINGUAGENS DE PROGRAMAÇÃO PARA CIÊNCIA DE DADOS (PYTHON COM SPARK)

Danilo Rodrigues Pereira

**Linguagens de Programação para Ciência de Dados
(Python com Spark)**

1ª edição

Londrina
Editora e Distribuidora Educacional S.A.
2019

© 2019 por Editora e Distribuidora Educacional S.A.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

Presidente

Rodrigo Galindo

Vice-Presidente de Pós-Graduação e Educação Continuada

Paulo de Tarso Pires de Moraes

Conselho Acadêmico

Carlos Roberto Pagani Junior

Camila Braga de Oliveira Higa

Carolina Yaly

Giani Vendramel de Oliveira

Juliana Caramigo Gennarini

Nirse Ruscheinsky Breternitz

Priscila Pereira Silva

Tayra Carolina Nascimento Aleixo

Coordenador

Nirse Ruscheinsky Breternitz

Revisor

Adriano Thomaz

Editorial

Alessandra Cristina Fahl

Beatriz Meloni Montefusco

Daniella Fernandes Haruze Manta

Hâmila Samai Franco dos Santos

Mariana de Campos Barroso

Paola Andressa Machado Leal

Dados Internacionais de Catalogação na Publicação (CIP)

Pereira, Danilo Rodrigues

P436l Linguagens de programação para ciência de dados (Python com Spark)/
Danilo Rodrigues Pereira, Marcelo Tavares de Lima, Anderson Paulo Avila Santos –
Londrina: Editora e Distribuidora Educacional S.A. 2019.
160 p.

ISBN 978-85-522-1641-4

1. Manipulação de dados. 2. Real time analytics. I.
Pereira, Danilo Rodrigues. II. Lima, Marcelo Tavares
de. III. Santos, Anderson Paulo Avila. Título.

CDD 004

Thamiris Mantovani CRB: 8/9491

2019

Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: <http://www.kroton.com.br/>



SUMÁRIO

Apresentação da disciplina	5
Introdução: por que Python, Spark e Hadoop?	
Preparação do ambiente em Spark e Python	7
Manipulação de dados com Python	25
Organização e visualização de dados	43
Análise estatística dos dados	70
<i>Machine Learning</i> em Python	93
Processando <i>Big Data</i> com Apache Spark	117
<i>Real Time Analytics</i> com Python e Spark	135

► Apresentação da disciplina


Caro aluno,

Python é uma linguagem de programação que vem sendo bastante utilizada na área de ciência de dados. São muitas as características e recursos do Python que o tornam uma linguagem tão promissora e bastante utilizada na área de ciência de dados e processamento de imagens. Um *framework bastante utilizado* para processamento de grande volume de dados (*Big Data*) é o *Apache Spark*. Ele é gratuito e tem como principais características: velocidade no processamento, suporte para diversos formatos de dados, facilidade de uso e suporte para diversos tipos de linguagem de programação, como Python, Java, R e Scala.

Os objetivos desta disciplina são abordar a linguagem Python e suas bibliotecas para análise e processamento de *Big Data* e fornecer também uma visão geral do *Apache Spark* para processamento de dados em tempo real (*real-time streaming*) e em lote (*batch*).

O conteúdo está organizado de forma que facilite a compreensão do Python para manipulação de *Big Data*, em conjunto com *machine learning* e também integração com ferramentas como *Apache Spark* e *Hadoop*. No final de cada conteúdo são propostos exercícios para fixação.

Inicialmente, apresenta-se uma breve introdução da linguagem Python, Apache Spark e Hadoop, e detalhes sobre a instalação e configuração ambiente de programação. Algumas bibliotecas em Python para manipulação de dados são apresentadas e, na sequência, um conteúdo inteiro é dedicado à organização e visualização dos dados usando Python e algumas API, como Pandas e Matplotlib. A probabilidade e estatística são introduzidas e que podem ser aplicadas para análise e melhor compreensão das informações contidas nos dados.



Depois, são apresentados alguns algoritmos de *machine learning* em Python, que podem ser utilizados em *Big Data*, e é apresentada uma visão geral do *framework Apache Spark*, focando nas bibliotecas (API) que são utilizadas para processamento de *Big Data*. Para finalizar a disciplina, o último conteúdo foca no processamento de dados em tempo real (*real-time analytics*) usando a *API Apache Spark Streaming*. Também é apresentada a possibilidade de fazer integração do *framework Apache Spark* com outras ferramentas de análise de dados em *streaming*, como Apache Kafka e Elasticsearch.

Boa leitura!



Introdução: por que Python, Spark e Hadoop? Preparação do ambiente em Spark e Python

Autor: Anderson Paulo Ávila Santos

► Objetivos

- Apresentar o Python, suas vantagens e desvantagens.
- Mostrar o Spark, o que é, como funciona, motivação de uso.
- Apresentar o Hadoop seu funcionamento e utilização.
- Configurar ambiente para o desenvolvimento.

► 1. Introdução

O Apache Spark é atualmente uma das ferramentas mais utilizadas quando se fala em trabalhar com *Big Data*, e o Python, por sua vez, é uma das linguagens de programação mais utilizadas para a Análise de Dados, Aprendizado de Máquina, entre outras tarefas. Então, por que não unir essas duas coisas? É aqui que o Spark com Python se destaca. Devido ao seu rico conjunto de bibliotecas, o Python é usado pela maioria dos especialistas em Ciência de Dados e analista de dados da atualidade. A integração do Python ao Spark foi uma grande contribuição para a comunidade.

O Spark foi desenvolvido na linguagem Scala, que é muito semelhante ao Java. Ele compila o código do programa no bytecode para o processamento de *big data* do JVM para Spark. Para dar suporte ao Spark com Python, a comunidade Apache Spark lançou o PySpark. Neste contexto, serão apresentadas as motivações para seu uso e as vantagens dessas tecnologias que são tão utilizadas, e que vêm cada dia mais sendo aplicadas em tarefas para se trabalhar com *Big Data*.

► 2. Python: O que é? Por que usar?

2.1 O que é o Python?

A linguagem de programação Python foi criada por Guido van Rossum, em 1991. Tinha por objetivo ser uma linguagem com alta produtividade e de fácil legibilidade. Assim, Python é uma linguagem criada com intuito de possibilitar a produção de código bom de fácil manutenção de maneira rápida. Existem algumas características que podem ser ressaltadas na linguagem de programação, são elas:


- A pouca utilização de caracteres especiais, fazendo com que a linguagem fique mais próxima do pseudocódigo.

- A obrigatoriedade da indentação para determinação dos blocos.
- O uso de poucas palavras-chave voltadas para a compilação.
- A existência de um coletor de lixo realizar o gerenciamento automático do uso da memória etc.

Além disso, a linguagem possibilita o uso de múltiplos paradigmas de programação. O paradigma de programação procedimental pode ser utilizado em programas simples e rápidos. Quando houver a necessidade da utilização de estruturas de dados complexas, tais como tuplas, listas e dicionários, estas estarão disponibilizadas para facilitar o desenvolvimento. Os projetos de grande porte podem ser desenvolvidos utilizando a orientação a objetos, que é amplamente suportada pelo Python (sobrecarga de operadores, herança múltipla etc.). Existe a possibilidade de uma programação utilizando também paradigma funcional, mesmo que de maneira mais modesta, fazendo do Python uma linguagem de programação com uma expressividade muito alta: é possível fazer muita coisa utilizando poucas linhas de comando. Também possui a capacidade de utilizar a meta-programação: técnicas simples que permitem alterar o comportamento da linguagem, possibilitando que sejam realizadas a criação de linguagens de domínio específico (DALCÍN; PAZ; STORTI, 2005, p. 1108-1115).

O Python disponibiliza uma grande biblioteca padrão, contendo classes, métodos e funções para realizar diversas tarefas, que vão do acesso a bancos de dados até interfaces gráficas com o usuário. Um dos objetivos dessa disciplina é mostrar que existe uma gama de ferramentas para trabalhar com dados científicos e grande volume de informações (DA SILVA e SILVA, 2019, p. 55-71).

A linguagem de programação Python é livre e multiplataforma. Com isso um programa escrito em uma determinada plataforma poderá ser executado em outra sem a necessidade de alteração no




código-fonte. Se a plataforma desejada não tiver a versão do Python, desenvolvedores podem estudar e modificar a linguagem para fazer com que ela rode onde quer que seja. Atualmente, o Python possui duas versões: a 2.x e a 3.x, elas possuem algumas diferenças de implementação, por isso recomenda-se a utilização da versão 3.x, pois a versão 2.x só receberá atualizações de segurança até 2020.

2.2 Por que Python?

A linguagem de programação Python traz muitas facilidades para sua utilização. Então, a pergunta coerente seria: por que a linguagem é ideal para *big data* e *machine learning*? Essa questão abre um leque com diversas respostas, mas é possível sintetizar algumas. A primeira se deve à grande expressividade da linguagem, facilitando a tradução do raciocínio lógico em algoritmo. Em aplicações de *big data* e *machine learning* o raciocínio é complexo. Seria um problema a mais para os desenvolvedores terem de se preocupar com sintaxe, gerenciamento de memória, recursos e outras complicações, além da resolução dos problemas que são de grande complexidade. Assim, Python faz todas essas tarefas de maneira automática e com grande eficiência, o que faz com que os desenvolvedores possam focar apenas na resolução do problema (OLIPHANT, 2007, p. 10-20).

O Python facilita a legibilidade. Com isso torna fácil a leitura e a compreensão de programas escritos há muito tempo. Comumente novos programas são baseados em outros mais antigos, fazendo com que sejam uma evolução. Para tal, é de grande importância a capacidade de entendimento dos códigos criados anteriormente. Isso se deve ao fato de que as palavras-chave da linguagem têm como objetivo a estrutura dos programas e não têm como finalidade a indicação de como o código deve ser compilado ou interpretado, fazendo com que não existam códigos que atrapalhem a interpretação do raciocínio lógico (OLIPHANT, 2007, p. 10-20).




Python é uma linguagem de programação livre e tem uma comunidade muita ativa e que contribui frequentemente com melhorias em bibliotecas de maneira voluntária. A documentação é ampla e consistente e há módulos para execução de qualquer tarefa necessária. Isso é de grande valia, pois não se tem tempo para se concentrar em coisas triviais, e poder contar com esses módulos já desenvolvidos é muito vantajoso. Um cuidado que se deve ter é com as versões disponíveis. Temos as versões 2 e 3 do Python, códigos feitos para serem executados na versão 2 não funcionarão na versão 3.

A linguagem que tem propósito de uso geral, diversas vezes precisa lidar com tarefas literais, tais como: utilização de banco de dados, leitura de páginas web, exibição gráfica de dados, criação de planilhas etc. Python tem grande facilidade nesse tipo de atividade, contendo módulos para praticamente todas as atividades. Esses são alguns dos motivos que torna Python uma linguagem que vem conquistando uma grande popularidade entre a comunidade desenvolvedora de plataformas para manipulação de grandes massas de dados.

2.3 O que é Spark?

Spark é um *framework* com objetivo de ter bom desempenho, uso fácil e análise de dados de maneira sofisticada para *Big Data*. É desenvolvido desde 2009 pela Universidade da Califórnia, em Berkeley, e no ano de 2010 teve o código aberto como projeto da Apache Foundation. O Spark possui vantagens em comparação com outras tecnologias para *Big Data* e tecnologias do paradigma MapReduce, tais como, Hadoop e Storm (SPARK, Apache. Spark. 2010).

O Spark proporciona um *framework* único e simples, com sua compreensão facilitada para gerenciar e processar *Big Data* com grande diversificação da natureza dos dados, por exemplo, texto, planilhas, grafos, entre outros, e de diversas origens, *batch* ou *streaming* em tempo real (SPARK, Apache. Spark. 2010).




Utilizando o Spark, as aplicações têm a possibilidade de serem executadas em clusters Hadoop aumentando o desempenho de até 100 vezes mais veloz em memória e 10 vezes mais rápido em disco, tudo isso podendo ser desenvolvido em Python, Scala ou Java. Disponibiliza mais de 80 possibilidades de operadores de alto nível, permitindo a utilização de forma interativa de consultas pelo console. Além de operações de Map/Reduce existe também suporte a consultas SQL, *streaming* de dados, processamento de grafos e aprendizado de máquina. Esses recursos podem ser utilizados isoladamente ou de forma combinada (SPARK, Apache. Spark. 2010).

2.4 O Hadoop

Hadoop existe desde 2008 e tem se mostrado uma ótima solução de processamento de grandes volumes de dados. O MapReduce é uma excelente solução que permite a execução de processamento de grandes massas de dados de maneira paralela e distribuída, na maioria das vezes no formato de cluster de computadores (O'MALLEY, Owen. 2010, p. 26-27).

Sua grande utilização se deve pela sua simplicidade, cada etapa do fluxo de processamento tem somente uma fase de Map e uma fase de Reduce, sendo assim é necessário converter qualquer processamento no padrão MapReduce para se chegar à solução. Os dados obtidos da saída do processamento nas etapas precisam ser armazenados em um sistema de arquivos distribuídos antes da execução do passo seguinte. Isso faz com que essa abordagem seja lenta por conta da necessidade do uso contínuo do armazenamento em disco (TURNER, 2011).

Além desses problemas, o Hadoop possui clusters que apresentam dificuldades para sua configuração e gerenciamento, além na necessidade de integrar com diversas ferramentas para diferentes casos de *Big Data*, tais como, Mahout para Aprendizado de Máquina e o Sorm para *streaming* de processamento.




Em cenários de alta complexidade, existiria a necessidade de encadear vários jobs de MapReduce para serem executados sequencialmente. Em cada um haveria alta latência e não poderia ser iniciado enquanto o anterior não tivesse sido concluído.

Com o Spark é possível que desenvolvedores criem pipelines com várias etapas complexas fazendo o uso de grafos direcionais acíclicos. Possibilita ainda o compartilhamento de dados em memória através desses grafos, sendo possível que diferentes tarefas trabalhem com os mesmos dados. No Spark a infraestrutura utilizada é o *Hadoop Distributed File System* (HDFS), com ferramentas adicionais que melhoram a implementação padrão do HDFS. O Spark pode ser considerado uma alternativa para o MapReduce do Hadoop, ao invés de um substituto. Considerando o Spark, então, uma solução com grande abrangência e unificada para diferentes casos de *Big Data* (ZAHARIA, 2016, p. 56-65).

Realizando uma extensão do MapReduce, o Spark evita que os dados sejam movidos durante seu processamento, isso é realizado com recursos como, por exemplo, o armazenamento de dados em memória com o processamento quase em tempo real desses dados. Com isso o desempenho é realmente muito bom, sendo melhor que muitas soluções apresentadas para resolução de problemas de *Big Data*. O Spark também disponibiliza suporte para validar consultas sob demanda para *Big Data*, ajudando na otimização do processamento, fornecendo uma API (Application Programming Interface) de alto nível para aprimoramento da produtividade no desenvolvimento e um modelo muito consistente para o desenvolvedor de soluções para *Big Data* (ZAHARIA, 2016, p. 56-65).

No Spark resultados intermediários são obtidos ainda em memória antes de escrever-se em disco. Isso é de grande valia quando existe a necessidade de processar o mesmo conjunto de dados sucessivas vezes. Foi projetado para que sua execução seja possível tanto na memória como em disco, isso faz com que o Spark execute operações em disco



em caso da falta de espaço de armazenamento em memória. Desta forma, é possível utilizar para o processamento de dados maiores que a memória agregada em um cluster (ZAHARIA, 2016, p. 56-65).

O Spark armazenará todos os dados que forem possíveis em memória para, na sequência, enviá-los para o disco. É tarefa do arquiteto olhar para os dados do problema que vai resolver e avaliar quais os requisitos de memória. Assim, o Spark traz diversas vantagens e ganhos de desempenho (ZAHARIA *et al.*, 2016, p. 56-65).

Ainda existem outras características que podemos observar no Spark, são elas:

- Suporte para funções que vão além do *Map* e *Reduce*.
- Otimização de operações para grafos.
- A avaliação de consultas para *Big Data* sob demanda, contribuindo para a otimização do fluxo do processamento de dados.
- APIs consistentes em Python, Java e Scala.
- Disponibiliza um Shell interativo para as linguagens Python e Scala.

O Spark é feito em linguagem Scala, que por sua vez é executado na máquina virtual do Java. Isso faz com que se tenha suporte para diversas linguagens de programação, atualmente trabalhando com as seguintes linguagens: Java, Scala, Python, Clojure e R (ZAHARIA, 2016, p. 56-65).

O Spark, além de uma API, disponibiliza várias bibliotecas adicionais, formando assim um ecossistema com grande capacidade para a realização das análises de *Big Data* e Aprendizado de Máquina. As bibliotecas incluem:

- O Spark *Streaming*: utilizado para processamento de dados em tempo real, ele tem como base a computação de *microbatch*.

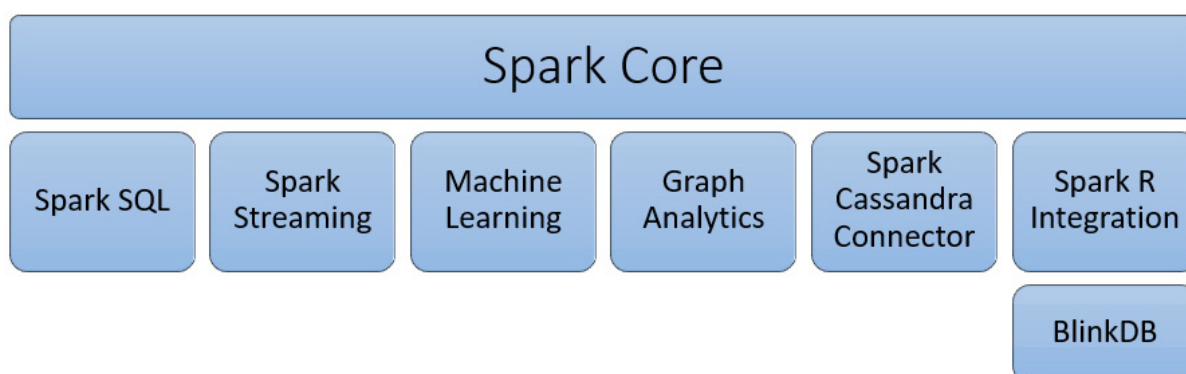
- O Spark SQL: permite a utilização do Spark através de API do JDBC, permitindo a execução de consultas nos mesmos moldes SQL sobre os dados, fazendo uso de ferramentas tradicionais de BI.
- O Spark MLlib: biblioteca que disponibiliza algoritmos de aprendizado de máquina, classificação, redução de dimensionalidade, regressão, clustering e filtragem colaborativa.
- O Spark GraphX: API para grafos e para computação paralela.

Além destas, existem outras bibliotecas e outros componentes dentro desse ecossistema, como por exemplo, BlinkDB e o Tachyon (ARMBRUST, 2015. p. 1383-1394).

O BlinkDB permite a realização de consultas interativas realizadas por amostragem, o que permite a calibragem das consultas em grandes volumes de dados. O Tachyon, que é um sistema de arquivos distribuídos em memória, facilita o compartilhamento de arquivos de forma rápida através em cluster, que armazena em cache arquivos que estão sendo utilizados (ARMBRUST, 2015. p. 1383-1394).

Por fim, existem alguns adaptadores para integração com outros tipos de produtos, como o Cassandra e R, como aparece esquematizado na Figura 1.

Figura 1 – Bibliotecas do *framework* spark

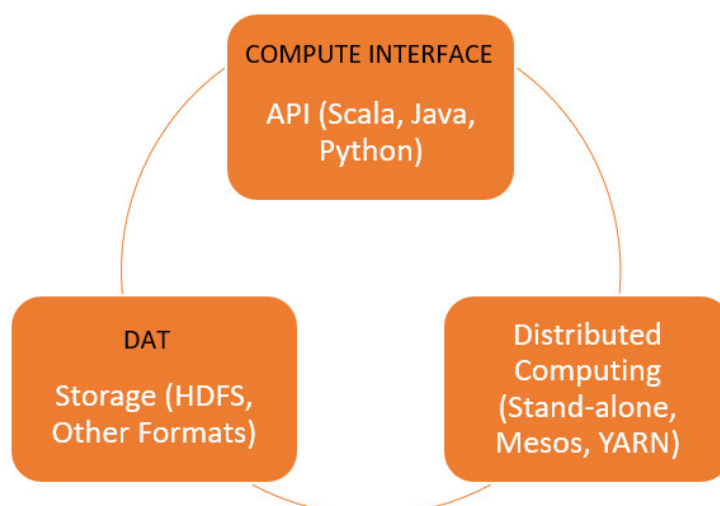


Fonte: elaborada pelo autor.

Com relação à arquitetura do Spark, temos os componentes: Armazenamento dos dados; API; *Framework* de Gerenciamento.

O armazenamento dos dados faz uso do sistema de arquivos HDFS. Este tipo de armazenamento possibilita a utilização de qualquer fonte de dados compatível com o Hadoop, que inclui: o próprio HDFS, HBase, Cassandra. A API possibilita aos desenvolvedores a utilização e criação de suas aplicações fazendo uso de interfaces de API padrão para Python, Java e Scala. Finalmente, na gestão de recursos com o Spark é possível implementar na forma de servidor autônomo, mas também no formato de computação distribuída. A figura 2 apresenta a Arquitetura do Spark (OUNACER, 2017, p. 44-51).

Figura 2 – Arquitetura Spark



Fonte: elaborada pelo autor.

O Spark apresenta também o conjunto de dados resilientes e distribuídos ou RDD (Resilient Distributed Datasets). Esse conjunto foi apresentado no trabalho de Zaharia *et al.* (2012) e é o conceito principal do *framework* Spark. Podemos imaginar o RDD como uma tabela do banco de dados que tem a capacidade de armazenar qualquer tipo de dado. No Spark os dados do RDD são armazenados em diferentes partições. Com isso, tem-se um benefício na reorganização e na otimização no processamento dos dados. Esses conceitos e funcionamentos serão vistos com mais detalhes.

► 3. Configurando o ambiente de desenvolvimento

Um dos pré-requisitos para a instalação do Spark é a instalação da JVM (Java Virtual Machine), o Java é utilizado por muitos outros softwares. Portanto, é bem possível que uma versão necessária (no nosso caso, versão 7 ou posterior) já esteja instalada no seu computador. Para verificar se o Java está disponível e encontrar sua versão, abra um prompt de comando e digite o seguinte comando:

```
java -version
```

Se o Java já estiver instalado e configurado para funcionar em um prompt de comando, a execução do comando acima deverá imprimir as informações sobre a versão do Java no console, como no exemplo da Figura 3.

Figura 3 – Verificação de versão do Java

```
C:\Users\anderson5687>java --version
java 10 2018-03-20
Java(TM) SE Runtime Environment 18.3 (build 10+46)
Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10+46, mixed mode)
```

Fonte: elaborada pelo autor.

Se não estiver instalado, você poderá receber uma mensagem como a seguinte:

```
'java' is not recognized as an internal or external command,  
operable program or batch file.
```

Isso significa que você precisa instalar o Java. Faça o download e instale o Java JDK. A única coisa relevante é definir a variável de ambiente `JAVA_HOME`, sendo recomendável usar a rota curta; de fato, se possível, instalar uma rota que não possui espaços em branco. Também é aconselhável adicionar `%JAVA_HOME% bin` ao `PATH`.

Na sequência, instale o Anaconda (para Python). Para verificar se o Python está disponível, abra um prompt de comando e digite o seguinte comando.

```
python --version
```

Se o Python estiver instalado e configurado para funcionar em um prompt de comando, a execução do comando acima deverá imprimir as informações sobre a versão do Python no console. Em vez disso, se você receber uma mensagem como:

```
'python' is not recognized as an internal or external  
command, operable program or batch file.
```

significa que você precisa instalar o Python. Instale o Anaconda com o qual todos os pacotes necessários serão instalados. O download pode ser feito realizando uma busca sobre `anaconda download`.

Após a conclusão da instalação, feche o prompt de comando; se já estiver aberto, abra-o e verifique se é possível executar com êxito o comando `python --version`.



PARA SABER MAIS

A instalação do Anaconda não é obrigatória. Ele é uma IDE para desenvolvimento em Python, podendo ser substituída por outra de sua preferência. As IDE's são aplicações e têm todas as funções necessárias para o desenvolvimento.

Na sequência deve ser instalado o Spark. Para tanto, siga os seguintes passos:

1. Pesquise por Spark Download e proceda como segue: escolha uma versão do Spark, selecione a versão estável mais recente do Spark.
2. Escolha um tipo de pacote, selecione uma versão pré-criada para a versão mais recente do Hadoop, como Pré-criada para Hadoop 2.7 e posterior.
3. Clique no link ao lado de Download Spark para baixar o spark-2.X.X-bin-hadoop2.7.tgz.
4. Para instalar o Apache Spark, não há necessidade de executar nenhum instalador. Você pode extrair os arquivos do arquivo zip baixado usando o winzip (clique com o botão direito no arquivo extraído e clique em extrair aqui).
5. Verifique se o caminho da pasta e o nome da pasta que contém os arquivos Spark não contêm espaços.

Agora, crie uma pasta chamada "spark" na área de trabalho e descompacte o arquivo que você baixou como uma pasta chamada spark-2.X.X-bin-hadoop2.7. Portanto, todos os arquivos Spark estarão em uma pasta chamada C: \ Usuários \ <nome do usuário> \ Desktop \ Spark \ spark-2.X.X-bin-hadoop2.7. A partir de agora, nos referiremos a esta pasta como SPARK_HOME neste documento.

Para testar se sua instalação foi bem-sucedida, abra o Anaconda Prompt, vá para o diretório SPARK_HOME e digite bin \ pyspark. Isso deve iniciar o shell PySpark, que pode ser usado para trabalhar interativamente com o Spark. A mensagem inicial do Spark será similar a apresentada na Figura 4.

Figura 4 – Mensagem Spark

A screenshot of a terminal window showing the PySpark shell's welcome message. The text is as follows:

```
Welcome to
 _ _ _ _ _
/ _ _ _ _ \   version 1.6.2
 \ V V V V /
  _/ _ _ \_
  |  _ _ |
  | | _ |
  |_|_|_|
>>>

Using Python version 2.7.10 (default, May 23 2015 09:44:00)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

Fonte: elaborada pelo autor.

Digite o comando version no shell. Ele deve imprimir a versão do Spark. Você pode sair do shell PySpark da mesma maneira que sai de qualquer shell Python, digitando exit ().

Por fim, será necessário baixar o winutils.exe e configurar a instalação do Spark para encontrar o winutils.exe.

1. Crie uma pasta hadoop \ bin dentro da pasta SPARK_HOME que já foi criada na etapa anterior.
2. Baixe o executável da versão do hadoop equivalente à sua instalação do Spark. Faça o download do winutils.exe para o hadoop de sua versão, faça uma pesquisa por winutils, localize o github de steveloughran, copie para a pasta hadoop \ bin na pasta SPARK_HOME.

Nota: os passos 3 e 4 requerem acesso de administrador.

3. Crie uma variável de ambiente do sistema no Windows chamada SPARK_HOME que aponte para o caminho da pasta SPARK_HOME. Isso precisa de acesso de administrador.
4. Crie outra variável de ambiente do sistema no Windows chamada HADOOP_HOME que aponte para a pasta hadoop dentro da pasta SPARK_HOME.

Como a pasta hadoop está dentro da pasta SPARK_HOME, é melhor criar a variável de ambiente HADOOP_HOME usando um valor de % SPARK_HOME% \ hadoop. Dessa forma, você não precisará alterar HADOOP_HOME se SPARK_HOME for atualizado.

ASSIMILE



O Spark necessita da instalação da JVM, pois foi desenvolvido em Scala, que compila Bytecode e é executado pela JVM, por isso sendo é uma multiplataforma.

TEORIA EM PRÁTICA



A geração de muitos dados e a não utilização destes é algo recorrente no dia a dia das empresas. A maior parte gera uma quantidade enorme de dados e não os analisa de forma a utilizá-los para melhorias, aumento das vendas, entre outras vantagens que podem ser obtidas. Neste contexto, uma dessas empresas que realiza vendas em varejo e armazena todos os dados das compras de seus clientes, decidiu que utilizará alguma tecnologia de *Big Data* para analisar o perfil de seus clientes, entre outras possibilidades que serão levantadas após a análise.

Para isto, contratou a sua consultoria para realizar um estudo para decidir qual das ferramentas existentes no mercado será a melhor opção. Portanto, elabore uma explicação com apresentação de exemplo de como a empresa poderá analisar a tecnologia para analisar o perfil dos clientes.



VERIFICAÇÃO DE LEITURA

1. Levando em consideração o contexto apresentado, qual das alternativas apresenta uma das principais vantagens da utilização da linguagem Python?
 - a. Linguagem que desobriga a indentação.
 - b. Grande verbosidade.
 - c. Documentação ampla.
 - d. Suporta apenas orientação a objetos.
 - e. Bibliotecas pouco atualizadas.
2. Com relação ao Spark, o que não caracteriza o *framework*?
 - a. Suporte apenas para função de *Map* e *Reduce*.
 - b. Operação em grafos otimizada.
 - c. Avaliação de consultas de *Big Data* sob demanda.

- d. APIs consistentes para Python, Java e Scala.
 - e. Shell interativo.
3. Com relação às características do Spark, seu ganho de desempenho em memória pode ser de quantas vezes em relação às tecnologias tradicionais?
- a. 2 vezes.
 - b. 4 vezes.
 - c. 200 vezes.
 - d. 100 vezes.
 - e. 30 vezes.

Referências bibliográficas

- ARMBRUST, M. *et al.* Spark sql: Relational data processing in spark. *In: Proceedings of the 2015 ACM SIGMOD international conference on management of data.* ACM, 2015. p. 1383-1394.
- DA SILVA, R. O.; SILVA, Igor Rodrigues Sousa. Linguagem de Programação Python. **Tecnologias em Projeção**, v. 10, n. 1, p. 55-71, 2019.
- DALCÍN, L.; PAZ, R.; STORTI, M.. MPI for Python. **Journal of Parallel and Distributed Computing**, v. 65, n. 9, p. 1108-1115, 2005.
- O'MALLEY, O. Integrating kerberos into apache hadoop. *In: Kerberos Conference.* 2010. p. 26-27.
- OLIPHANT, T. E. Python for scientific computing. **Computing in Science & Engineering**, v. 9, n. 3, p. 10-20, 2007.
- OUNACER, S. *et al.* A New Architecture for Real Time Data Stream Processing. **International Journal of Advanced Computer Science and Applications**, v. 8, n. 11, p. 44-51, 2017.
- SPARK, Apache. Spark. The Apache Software Foundation, [Online]. Available: <http://spark.apache.org/>. Acesso 06 de set. de 2019], 2010.

TURNER, J. **Hadoop**: What it is, how it works, and what it can do. O'Reilly, 2011.
Disponível em: <https://www.oreilly.com/ideas/what-is-hadoop>. Acesso em: 9 dez. 2019.

ZAHARIA, M. *et al.* Apache spark: a unified engine for big data processing. **Communications of the ACM**, v. 59, n. 11, p. 56-65, 2016.

ZAHARIA, M. *et al.* Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012. p. 2-2.

Gabarito

Questão 1 – Resposta: C

Resolução: A documentação do Python é ampla e disponibiliza diversos módulos para auxílio no desenvolvimento de softwares.

Questão 2 – Resposta: A

Resolução: Ele suporta funções que vão além do *Map* e *Reduce*.

Questão 3 – Resposta: D

Resolução: O ganho de desempenho devido ao paralelismo e outras técnicas utilizadas pelo Spark pode ser até 100 vezes mais rápido que outras técnicas.



Manipulação de dados com Python

Autor: Anderson Paulo Ávila Santos

► Objetivos

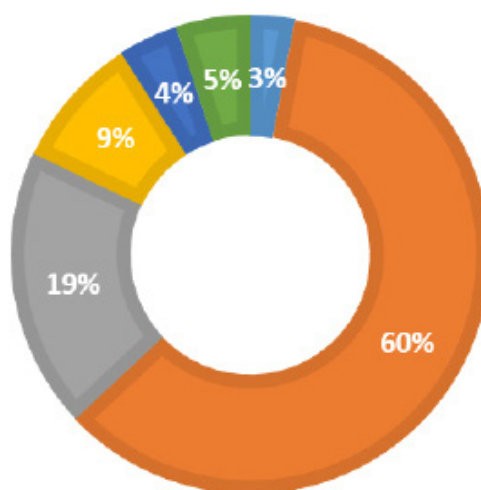
- Apresentar as formas de manipulação de dados em Python.
- Preparar bibliotecas de limpeza de dados.
- Apresentar biblioteca Pandas para manipulação de dados.

► 1. Motivação

A manipulação de dados é de extrema importância para a ciência de dados. Em uma pesquisa com cientistas de dados publicada pela Forbes, percebeu-se que eles passam a maior parte do tempo preparando os dados em vez de minerá-los ou modelá-los. Ainda assim, a maioria está feliz em ter o emprego mais cobiçado do século XXI. A pesquisa com cerca de 80 cientistas de dados foi realizada pelo segundo ano consecutivo pela *CrowdFlower*, fornecedora de uma plataforma de “enriquecimento de dados” para cientistas de dados. As tarefas realizadas por um cientista de dados, a preparação de dados, representa cerca de 80% do trabalho.

Figura 1 – O que um cientista de dados gasta mais tempo fazendo

■ Criando conjunto de testes ■ Limpando e organizando dados ■ Coletando conjunto de dados
■ Minerando dados para padrões ■ Refinando algoritmos ■ Outros



Fonte: Press (2016).

Como podemos ver na Figura 1, os cientistas de dados gastam 60% de seu tempo limpando e organizando dados. A coleta de conjuntos de dados fica em segundo lugar, em 19% do tempo, o que significa que os cientistas gastam cerca de 80% do tempo na preparação e gerenciamento de dados para análise.



PARA SABER MAIS

Destaca-se na Figura 1 que 57% dos cientistas de dados consideram a limpeza e a organização dos dados a parte menos agradável de seu trabalho e 19% dizem isso sobre a coleta de conjuntos de dados, ou seja, a captação das informações.

Para fazer as manipulações de dados, o Python disponibiliza uma série de bibliotecas que visam facilitar e proporcionar maior agilidade nessas tarefas que são consideradas tão árduas.

► 2. Principais bibliotecas

Dentre uma série de bibliotecas que o Python disponibiliza para a manipulação de dados, estão o Numpy, o Scipy e o Pandas.

O Numpy é uma biblioteca Python que dá suporte de operações numéricas, trabalha com arranjos, vetores e matrizes de N dimensões, possui diversas funções e operações sofisticadas; fornece estruturas para array; operações matemáticas e estatísticas; execução vetorizada das funções. O Scipy é voltado para a programação científica e reúne alguns algoritmos para álgebra linear, equações diferenciais, otimização, estatística, entre outras. Por fim, o Pandas disponibiliza uma estrutura para manipulação de dados em formato tabular; implementa a estrutura de data.frames do R; fornece operações para leitura e escrita, ordenação e arranjo de disposição dos dados, seleção, filtragem e fatiamento dos dados, transformações, agregações e junções dos dados (WILLEMS, 2017).

Uma das principais estruturas que o Pandas disponibiliza para a utilização são os *DataFrames*. Os *DataFrames* têm estruturas que

trabalham de maneira tabular. Sendo assim, ele é organizado em linhas e colunas, onde cada linha é um registro e cada coluna um campo. Desta forma as colunas possuem dados de um mesmo tipo, podemos imaginar a tabulação como em uma tabela do Excel, com a limitação de um tipo de dado por coluna (WILLEMS, 2017).

Pandas é uma biblioteca do Python e para ser utilizada é necessário que sua instalação seja feita previamente; para instalar via pip (gerenciador de pacotes do Python) o comando é:

```
python -m pip install --upgrade pandas
```

ASSIMILE



O pip é o gerenciador de pacotes padrão do Python. Ele permite que você instale e gerencie pacotes adicionais que não fazem parte da biblioteca padrão do Python. Traz diversas facilidades para gerenciamento das versões desses pacotes.

Como já mencionado, o Pandas utiliza o *DataFrame* para manipulação dos dados, que pode ser criado através do dicionário do Python. Para dar início à utilização do Pandas, primeiro deve ser feita a importação da biblioteca, como no exemplo a seguir, no qual é criado um *DataFrame* através de um dicionário de dados, apresenta um data frame que objetiva contabilizar a população do Brasil, Chile e Argentina nos anos de 2005 a 2008 (WILLEMS, 2017).

No entanto, essa forma de criação de *DataFrame* não é a mais utilizada, então tem-se a possibilidade de importação de arquivos e a transformação dos mesmos em *DataFrame* ou a utilização de banco de dados para tal.

Quem está familiarizado com o R já conhece o *DataFrame* como uma maneira de armazenar dados em formato de tabela de dados que podem ser facilmente visualizadas. Cada linha dessas tabelas corresponde a uma instância do dado, enquanto cada coluna é um vetor que contém dados para uma variável específica. Sendo assim, os *DataFrames* no Python são muito semelhantes: eles vêm com a biblioteca Pandas e são definidos como estruturas de dados rotuladas bidimensionais com colunas de tipos potencialmente diferentes. Portanto, pode-se definir que os *DataFrames* do Pandas consistem em três componentes principais: os dados, o índice e as colunas.

```
import pandas as pd

dt_data = {'pais': ['Brasil', 'Argentina', 'Argentina',
                  'Brasil', 'Chile', 'Chile'],
          'ano': [2005, 2006, 2005, 2006, 2007, 2008],
          'população': [170.1, 30.5, 32.2, 172.6, 40.8, 42.0]}

df = pd.DataFrame(dt_data)

print(df)
```

	país	ano	população
0	Brasil	2005	170.1
1	Argentina	2006	30.5
2	Argentina	2005	32.2
3	Brasil	2006	172.6
4	Chile	2007	40.8
5	Chile	2008	42.0

No código anterior foi demonstrado como é realizada a criação de uma estrutura de um *DataFrame*. Esse tipo de estrutura disponibiliza diversas funções que podem ser executadas, como exemplo, para descobrir a quantidade de elementos, o valor médio de alguma coluna etc. Portanto,

na sequência, serão demonstradas algumas das principais funções disponíveis no pandas para a utilização.

Primeiramente, antes de começar a excluir, adicionar, renomear os componentes em um *DataFrame* é preciso saber como os elementos podem ser selecionados dentro dessa estrutura (DA SILVA, 2019). A seleção dos dados é um tanto quanto simples, por exemplo, considerando o *DataFrame* do Quadro 1:

Quadro 1 – Exemplo de *DataFrame*

	A	B	C
0	1	2	3
1	4	5	6
2	7	8	9

Supondo que o valor que se deseja encontrar está na linha 0 na coluna 'A', existem várias opções para se recuperar o valor 1. A seguir são demonstradas algumas das opções.

```
print(df.iloc[0][0])
```

```
print(df.loc[0]['A'])
```

```
print(df.at[0, 'A'])
```

```
print(df.iat[0, 0])
```

Como podemos ver no código anterior, existem algumas formas de realizar o acesso aos dados. Com isso, percebe-se que os valores podem ser acessados pela posição ou pelo nome individual de cada coluna.

Após saber como é realizada a seleção de valores dentro de um *DataFrame*, agora é necessário verificar como se adiciona um índice, linha ou coluna. Por padrão, quando um *DataFrame* é criado um índice numérico é criado, esse índice que é auto incremental está associado

com cada um dos registros e se mantém associado mesmo após filtros, fatiamento, entre outras operações, ele pode ser alterado através da função `set_index()`.

Antes de aprender a realizar a inserção de novos dados em um *DataFrame* é preciso conhecer alguns conceitos presentes nos *DataFrames*, sendo eles: conceitos de `loc`, `iloc` e `ix`. Eles funcionam da seguinte maneira:

- `.loc []` funciona nos rótulos do índice do *DataFrame*. Isso significa que, se você fornecer o `loc [2]`, procurará os valores do seu *DataFrame* com um índice rotulado 2.
- `.iloc []` trabalha nas posições em seu índice. Isso significa que, se você fornecer `iloc [2]`, procurará os valores do seu *DataFrame* que estão no índice '2'.
- `.ix []` é um caso mais complexo: quando o índice é baseado em números inteiros, você passa um rótulo para `.ix []`. `ix [2]` significa que você está procurando no *DataFrame* valores com um índice rotulado como 2. Isso é como `.loc []`! No entanto, se seu índice não for baseado apenas em números inteiros, o `ix` funcionará com posições, assim como `.iloc []`.

Como o descrito, existem algumas maneiras de acessar e alterar os índices de um *DataFrame*. O código a seguir demonstra algumas possibilidades:

```
import numpy as np

import pandas as pd

df = pd.DataFrame(data=np.array([[1,2,3], [4,5,6],
[7,8,9]]), index = [2,'A', 4], columns=[48,49,50])

print(df)
```

```
print(df.loc[2])  
print(df.iloc[2])  
print(df.ix[2])
```

Observe que, no código anterior, foi utilizado um exemplo de um *DataFrame* que não é baseado apenas em números inteiros, para facilitar o entendimento das diferenças. É possível ver claramente que passar 2 para `.loc []` ou `.iloc []` / `.ix []` não retorna o mesmo resultado, como demonstrado a seguir.

```
48 49 50
```

```
2 1 2 3
```

```
A 4 5 6
```

```
4 7 8 9
```

```
48 1
```

```
49 2
```

```
50 3
```

```
Name: 2, dtype: int32
```

```
48 7
```

```
49 8
```

```
50 9
```

```
Name: 4, dtype: int32
```

```
48 7
```

```
49 8
```

```
50 9
```

```
Name: 4, dtype: int32
```


Para adicionar colunas e valores em um *DataFrame* é muito simples. Por exemplo, se em algum caso, se deseja tornar seu índice parte do seu *DataFrame*, isso pode ser feito facilmente, usando uma coluna do *DataFrame* ou adicionando uma coluna que ainda não existia e atribuindo-a à propriedade `.index`; assim:

```
import numpy as np

import pandas as pd

df = pd.DataFrame(data=np.array([[1,2,3], [4,5,6],
[7,8,9]]), columns=['A', 'B', 'C'])

df['D'] = df.index

print(df)
```

Ao definir um novo índice não existente no *DataFrame*, o mesmo será criado e poderá ser utilizado. O código anterior resultará no seguinte resultado:

	A	B	C	D
0	1	2	3	0
1	4	5	6	1
2	7	8	9	2

Com relação a como excluir índices, linhas ou colunas de um *DataFrame* do Pandas existem algumas possibilidades.

2.1. Excluindo um índice do *DataFrame*

Se existe um desejo de se remover o índice do *DataFrame*, pode se reconsiderar, porque *DataFrames* e *Series* sempre têm um índice. No entanto, o que é possível fazer é, por exemplo:

Redefinir o índice do seu *DataFrame* ou remover o nome do índice, se houver, executando `del df.index.name`; remova os valores duplicados do índice redefinindo o índice, soltando as duplicatas da coluna de índice que foram adicionadas ao seu *DataFrame* e restabelecendo a coluna sem duplicação novamente como o índice:

```
import numpy as np
import pandas as pd

df = pd.DataFrame(data=np.array([[1,2,3], [4,5,6],
[7,8,9], [40, 50, 60], [23, 35, 37]]), index = [2.5,
12.6, 4.8, 4.8, 2.5], columns=[48, 49, 50])

df.reset_index().drop_duplicates(subset= 'index',
keep='last').set_index('index')
```

Para possibilitar a exclusão de uma coluna do *DataFrame* pode ser utilizado o método `drop()`. A deleção é simples, pode ser feita utilizando o nome da coluna ou seu respectivo índice, como pode ser visto na sequência:

```
df.drop('A', axis=1, inplace=True)
df.drop(df.columns[[1]], axis=1)
```

Como podemos ver no código, alguns parâmetros extras devem ser passados. São eles: o argumento do `axis` (eixo) é 0 quando indica linhas e 1 quando é usado para descartar colunas. Você pode definir no `inplace` como `True` para excluir a coluna sem precisar reatribuir o *DataFrame*.

Para remover uma linha do seu *DataFrame*, pode-se remover linhas duplicadas do seu *DataFrame* executando `df.drop_duplicates()`. Você também pode remover linhas do seu *DataFrame*, levando em conta apenas os valores duplicados que existem em uma coluna. Como no exemplo.

- `df.drop_duplicates([48], keep='last')`

Se não houver critério de unicidade para a exclusão que se deseja executar, pode-se usar o método `drop()`, que usa a propriedade `index` para especificar o índice de quais linhas você deseja remover do *DataFrame*:

- `print(df.drop(df.index[1]))`

Além disso, é possível renomear o índice ou colunas de um *DataFrame* do Pandas, para atribuir às colunas ou aos valores de índice do *DataFrame* um valor diferente; deve ser usado o método `.rename()`, como no código a seguir.

```
newcols = {
    'A' : 'new_column_1',
    'B' : 'new_column_2',
    'C' : 'new_column_3'
}

df.rename(columns=newcols, inplace = True)
df.rename(index = {1: 'a'})
```

Essas formas de manipulação de dados, usando *DataFrame* para realizar a manipulação básica dos dados. Existem também algumas formas de realizar a manipulação de maneira mais avançada, o que facilita muito na hora de fazer a análise dos dados.

Outras funcionalidades que o *DataFrame* do Pandas proporciona é com relação à formatação dos dados. Na maioria das vezes, pode-se executar algumas operações nos valores reais contidos no seu *DataFrame*.

Uma das possibilidades é substituir todas as ocorrências de uma sequência de caracteres em um *DataFrame*. Para isto, pode ser utilizado

o `replace()`, a função recebe dois parâmetros: os valores que gostaria de alterar, seguidos pelos valores pelos quais deseja substituí-los.

- `df = df.replace(['Awful', 'Poor', 'OK', 'Acceptable', 'Perfect'], [0, 1, 2, 3, 4])`

Também há um argumento `regex` que pode ajudar muito quando existem combinações padronizadas de cadeias de caracteres:

- `df.replace({'\n': '
'}, regex=True)`

Em suma, a função `replace()` é a principal quando se precisa lidar com substituição de valores ou cadeias de caracteres no seu *DataFrame*.

Também é possível remover partes de uma sequência de caracteres nas células do *DataFrame*. Normalmente, remover partes indesejadas de uma string é um trabalho complicado. Felizmente, existe uma solução para esse problema já implementada para a utilização.

- `df['result'] = df['result'].map(lambda x: x.lstrip('+').rstrip('aAbBcC'))`

Utiliza o `map()` no resultado da coluna para aplicar a função `lambda` sobre cada elemento ou elemento da coluna. A função, por si só, pega o valor da string e retira o + ou - localizado à esquerda e também retira qualquer um dos seis `aAbBcC` à direita.

Outra funcionalidade é a divisão de texto em uma coluna em várias linhas em um *DataFrame*. Essa é uma tarefa de formatação mais difícil, mas o *DataFrame* possui funcionalidades de `split()`, entre outras, para ajudar nesse processo.

Também existe a possibilidade de aplicar uma função em uma linha ou coluna inteira do *DataFrame*. Pode-se adicionar uma função `lambda`:

```
doubler = lambda x: x*2
```

```
df['A'].apply(doubler)
```

Com o exemplo de código anterior, será aplicada a função lambda para toda a coluna 'A' do seu *DataFrame*.

► 3. Importando dados para o *DataFrame*

Além de criar *DataFrame* é possível importar arquivos como, por exemplo, arquivos CSV, que são muito utilizados para armazenamento de dados em formato texto.

```
import pandas as pd

pd.read_csv('nomeArquivo', parse_dates=True)
```

ou essa opção:

```
pd.read_csv('nomeArquivo', parse_dates=['nomeColuna'])
```

Caso acontecer de ter, por exemplo, data em formato diferente do padrão, é possível construir seu próprio analisador para lidar com isso. Por exemplo, cria-se uma função lambda que controla seu *DateTime* com uma sequência de formatação.

```
import pandas as pd

dateparser = lambda x: pd.datetime.strptime(x, '%Y-%m-%d
%H:%M:%S')
```

O que torna seu comando de leitura:

```
pd.read_csv(infile, parse_dates=['columnName'], date_
parser=dateparser)
```

Ou combine duas colunas em uma única coluna *DateTime*

```
pd.read_csv(infile, parse_dates={'datetime': ['date',
'time']}, date_parser=dateparser)
```

Outro ponto bastante importante é a condição de se iterar dentro do *DataFrame*. É possível percorrer as linhas do *DataFrame* com a ajuda de um loop em combinação com uma chamada `iterrows()`, como podemos ver no código a seguir:

```
import numpy as np

import pandas as pd

df = pd.DataFrame(data=np.array([[1,2,3], [4,5,6],
[7,8,9]]), columns=[48,49,50])

for index, row in df.iterrows():

    print(row['A'], row['B'])
```

Além de possibilitar o carregamento de um arquivo com dados para um *DataFrame*, o inverso também é possível. Tem-se a possibilidade de persistir os dados em um arquivo do tipo CSV ou Excel.

Para se fazer a saída de um *DataFrame* para CSV, pode-se usar `to_csv()`:


```
import pandas as pd

df.to_csv('nomeArquivoCSV.csv')
```

Esse trecho de código parece bastante simples, mas pode haver requisitos específicos para a saída de dados. Por exemplo, pode ser que não seja interessante utilizar a vírgula como separador dos dados e se queira utilizar outro caracter. Para ajudar nesse tipo de situação, é possível passar alguns argumentos adicionais para `to_csv()` para garantir que seus dados sejam gerados da maneira desejada. No exemplo a seguir, especificamos o separador e a codificação dos caracteres.

```
import pandas as pd

df.to_csv('meuDataFrame.csv', sep='\t',
encoding='utf-8')
```



Além disso, pode-se especificar como deseja que seu NaN (Not a Number) (ou valores ausentes sejam representados, se deseja ou não exibir o cabeçalho, se deseja ou não gravar os nomes das linhas, se deseja compactação entre outras muitas possibilidades. Para isto, a documentação da biblioteca e da função pode ajudar muito. Pesquise sobre Pandas Documentation, e encontrará tudo isso detalhado.

Outra possibilidade é a gravação do *DataFrame* no formato para MS Excel. Da mesma forma que se faz para salvar o *DataFrame* em CSV, pode-se usar `to_excel()` para gravar sua tabela no MS Excel. No entanto, é um pouco mais complicado, como podemos ver no exemplo apresentado no código a seguir.

```
import numpy as np

import pandas as pd

df = pd.DataFrame(data=np.array([[1,2,3], [4,5,6],
[7,8,9]]), columns=[48,49,50])

writer = pd.ExcelWriter('myDataFrame.xlsx')

df.to_excel(writer, 'DataFrame')

writer.save()
```

Python para ciência de dados é mais do que *DataFrames*, no entanto o Pandas possui funções essenciais para tarefas como importar, limpar e manipular seus dados para que o trabalho de ciência de dados seja feito de uma maneira mais fluida. A importância do pré-processamento na ciência de dados é enorme, pois se permitir que os algoritmos de Machine Learning trabalhem com dados incorretos ou dados sem um padrão de formatação, é pouco provável que esses algoritmos funcionem.



TEORIA EM PRÁTICA

Em Python existem algumas bibliotecas disponibilizadas para facilitar a manipulação de dados; essas bibliotecas são otimizadas e buscam a melhor performance para a importação de dados. Uma das principais e mais utilizadas é o Pandas, que disponibiliza um tipo de estrutura de manipulação de dados chamado *DataFrame*.

Para a manipulação de dados sobre clientes, uma empresa precisa importar os dados presentes em arquivos CSV e remover/alterar algumas informações contidas nesses dados. Dessa forma, é necessário executar alguma função para que a importação seja realizada.



VERIFICAÇÃO DE LEITURA

1. Utilizando *DataFrame*, qual comando deverá ser utilizado para realizar a importação desses dados?
 - a. `import_csv()`
 - b. `to_csv()`
 - c. `read_csv ()`
 - d. `import_file()`
 - e. `export_csv()`
2. Caso exista a necessidade de substituir todas as ocorrências de uma sequência de caracteres em um

DataFrame. Qual é o comando e os parâmetros que devem ser utilizados.

- a. `replace(list)`
- b. `sub(list,list)`
- c. `replace(list,list)`
- d. `change(list)`
- e. `sub(list)`

3. Nessa mesma situação, onde é necessário realizar a importação dos dados, se a intenção for aplicar uma formatação para uma coluna ou linha inteira, existe a possibilidade de aplicar uma função lambda. Qual das questões a seguir representa uma possível aplicação de uma função lambda?

- a. `df['A'].lambda(funcao)`
- b. `df['A'].function(funcao)`
- c. `df['A'].apply(funcao)`
- d. `df['A'].apply_func(funcao)`
- e. `df['A'].ap_lambda(funcao)`

Referências bibliográficas

PRESS, Gil. Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says. **Forbes**, [S. l.], p. 1-3, 23 mar. 2016. Disponível em: <https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#326d062d6f63>. Acesso em: 30 out. 2019.

SILVA, Rogério Oliveira; SILVA, Igor Rodrigues Sousa. Linguagem de Programação Python. **Tecnologias em Projeção**, v. 10, n. 1, p. 55-71, 2019.

WILLEMS, Karlijn. Apache Spark in Python: Beginner's Guide: A beginner's guide to Spark in Python based on 9 popular questions, such as how to install PySpark in Jupyter Notebook, best practices. **DataCamp Community**, [S. l.], p. 1-4, 28 mar. 2017. Disponível em: <https://www.datacamp.com/community/tutorials/apache-spark-python>. Acesso em: 31 out. 2019.

Gabarito

Questão 1 – Resposta: C

Resolução: O comando para importação de um arquivo CSV para *DataFrame* é o `to_csv`.

Questão 2 – Resposta: C

Resolução: O comando `replace` recebe duas listas, a original e a que será trocada.

Questão 3 – Resposta: C

Resolução: Para aplicação de função lambda é utilizada a função `apply` na coluna ou linha desejada, ele executará a função lambda.



Organização e visualização de dados

Autor: Danilo Rodrigues Pereira

Objetivos

Organização e visualização dos dados são tarefas essenciais para a área de ciência de dados. O objetivo dessa unidade é demonstrar diversas maneiras para organização e visualização dos dados em Python. No final da unidade você será capaz de:

- Manipular planilha Excel usando a biblioteca Pandas.
- Criar gráficos customizados utilizando a biblioteca Matplotlib e Pandas.
- Criar diferentes tipos de visualização (gráficos, tabelas, diagramas, histogramas etc.) de acordo com suas necessidades.

► 1. Organização e visualização de dados

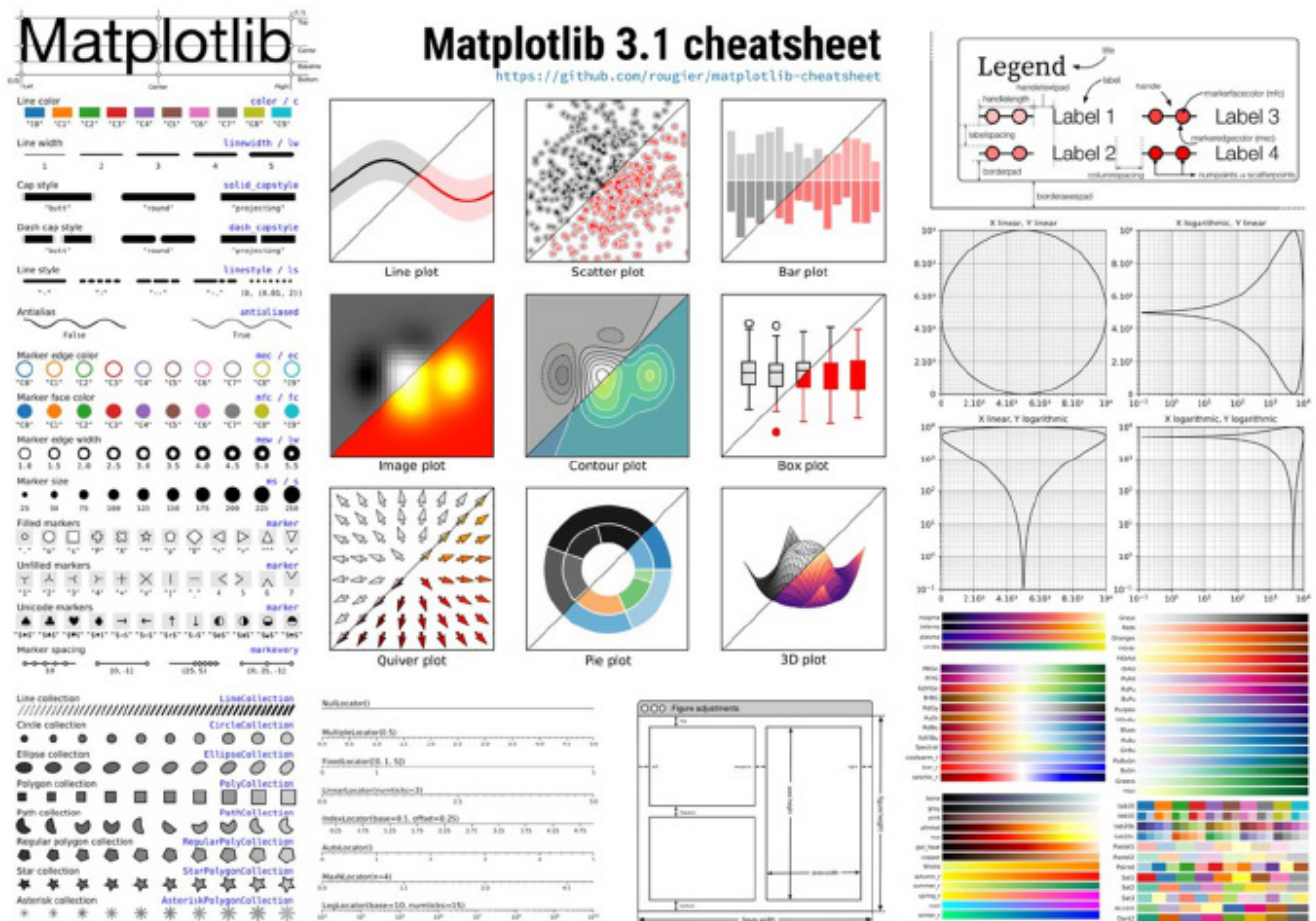
A visualização é uma técnica que consiste na criação de imagens, diagramas, histogramas, gráficos ou animação para melhor representar a informação. Através de elementos visuais, a visualização de dados é uma maneira de analisar e entender as exceções, tendências, padrões ou anormalidade nos dados.

A organização e visualização dos dados também é usada para o processo de tomada de decisão em empresas e através de inspeção e análises apresentadas visualmente é possível entender conceitos difíceis ou identificar novos padrões (TOSI, 2009; YIM, 2018; HUNTER, 2019).

1.1 Introdução à biblioteca Matplotlib

Matplotlib é a principal biblioteca de plotagem científica em Python. Ela suporta visualização interativa e não interativa, e fornece uma ampla variedade de tipos de plotagem (Figura 1). Além disso, é altamente personalizável, flexível e fácil de usar (HUNTER, 2019). O Matplotlib foi desenvolvido usando a linguagem de programação MATLAB e Python. O alto grau de compatibilidade entre MATLAB e Python fizeram com que muitos desenvolvedores mudassem do MATLAB para o Matplotlib.

Figura 1 – Visão geral da biblioteca do Matplotlib



Fonte: Rougier (2019).

1.1.1 Instalação e dependências

Antes de instalar a biblioteca Matplotlib, as seguintes bibliotecas são necessárias:

- *Python* (≥ 3.6)
- *FreeType* (≥ 2.3)
- *libpng* (≥ 1.2)
- *NumPy* (≥ 1.11)
- *setuptools*
- *cycler* ($\geq 0.10.0$)
- *dateutil* (≥ 2.1)

- *kiwisolver* ($\geq 1.0.0$)

- *pyparsing*

Existem várias maneiras de instalar o Matplotlib, que vai variar de acordo com o seu sistema operacional:

- **Linux:** O Matplotlib é fornecido para quase todas as distribuições Linux. A vantagem de usar uma distribuição Linux é que vários programas e bibliotecas são preparados pelos desenvolvedores da distribuição e disponibilizado para os usuários, ou seja, não é necessária a instalação.
- **Debian/Ubuntu:** *sudo apt-get install python3-matplotlib.*
- **Fedora:** *sudo dnf install python3-matplotlib.*
- **Red Hat:** *sudo yum install python3-matplotlib.*
- **Arch:** *sudo pacman -S python-matplotlib.*
- **Windows:** Antes da instalação do Matplotlib, você precisa instalar as principais dependências: *Python* e *NumPy*. Em seguida, visite o site oficial do Python e acesse a seção *Download >> Windows*.
- **Mac OS X:** O procedimento para instalar o Matplotlib no Mac OS X é semelhante ao Windows. Instale as bibliotecas *Python* e *NumPy*. Em seguida, visite o site oficial do Python e acesse a seção *Download >> Mac OS X*.

1.1.2 Criando gráficos customizados

Um dos pontos fortes do Matplotlib é a rapidez e a simplicidade com que podemos começar a produzir diferentes tipos de gráficos customizados, que veremos nas seções a seguir.

1.1.2.1 Gráficos de linhas

A seguir, um exemplo de código-fonte desenvolvido para criar gráficos de linhas (Figura 1). Você precisa especificar apenas uma lista de valores que representam as coordenadas verticais dos pontos a serem plotados (Quadro 1).

Quadro 1 – Exemplo de criação de gráficos de linhas usando números aleatórios

```
import matplotlib.pyplot as plt
import numpy as np

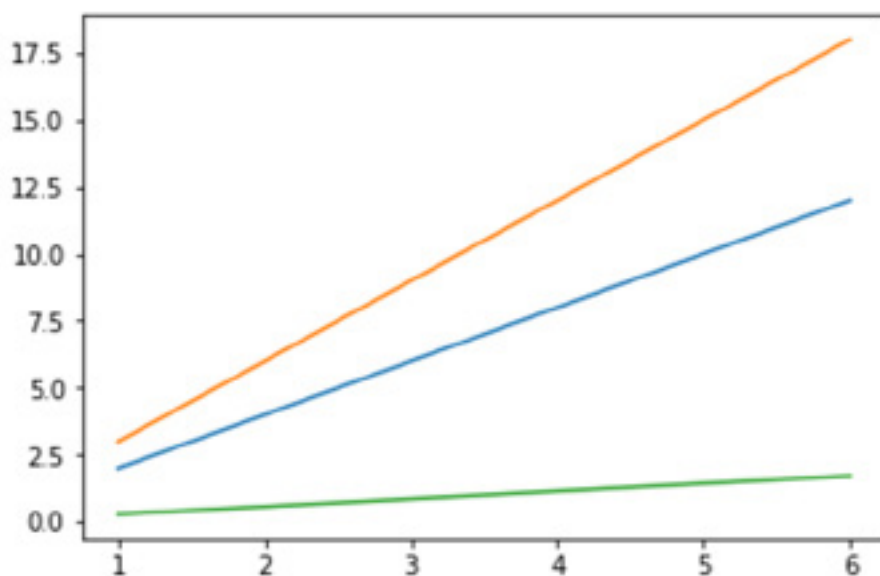
x = range(1, 7)

plt.plot(x, [xi * 2 for xi in x])
plt.plot(x, [xi * 3.0 for xi in x])
plt.plot(x, [xi / 3.5 for xi in x])

plt.show()
```

Saída:

Figura 2 – Gráficos de linhas, criados a partir de números gerados aleatoriamente



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.1.2.2 Grid, eixos, labels, título e legendas

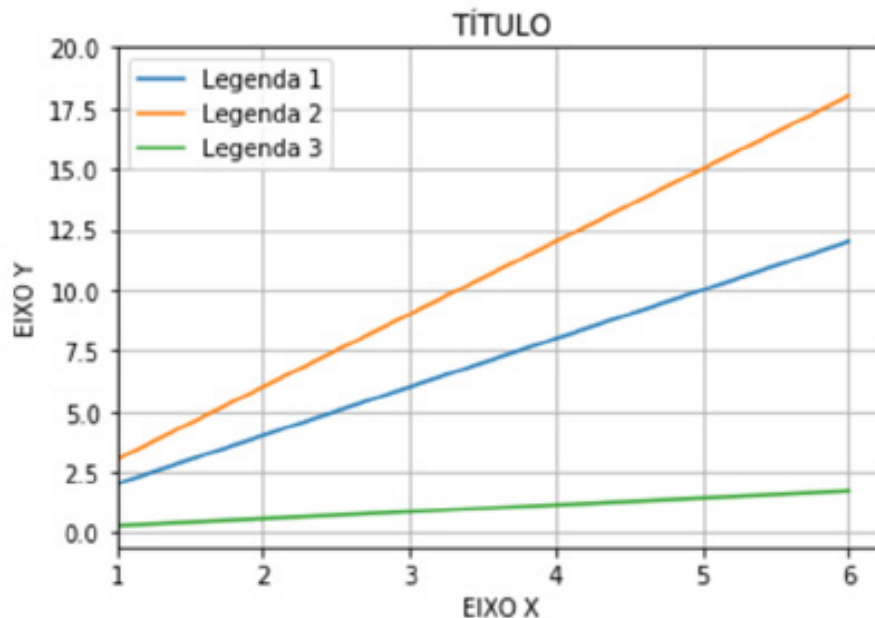
Agora vamos aprender sobre alguns recursos do Matplotlib que podemos adicionar aos gráficos, como por exemplo: *grid*, eixos, *labels*, títulos e legendas (Figura 3). No exemplo a seguir, vamos modificar o código-fonte do quadro 1 para adicionar um grid na área que os gráficos foram plotados; adicionar legenda nos eixos X e Y; colocar o título e adicionar legendas nas linhas dos gráficos (Quadro 2).

Quadro 2 – Exemplo de código-fonte para adicionar recurso no gráfico

```
# -*- coding: utf-8 -*-  
import matplotlib.pyplot as plt  
import numpy as np  
  
x = range(1, 7)  
  
plt.plot(x, [xi * 2 for xi in x], label='Legenda 1')  
plt.plot(x, [xi * 3.0 for xi in x], label='Legenda 2')  
plt.plot(x, [xi / 3.5 for xi in x], label='Legenda 3')  
  
plt.grid(True)  
plt.axis(xmin=1, ymax=20)  
plt.xlabel('EIXO X')  
plt.ylabel('EIXO Y')  
plt.title(u'GRÁFICOS DE LINHA')  
plt.legend()  
  
plt.show()
```


Saída:

Figura 3 – Gráficos de linhas customizados com grid e legenda, criados a partir de números gerados aleatoriamente.



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.1.2.3 Marcadores (*markers*) e linhas

Nas seções anteriores, todos os gráficos foram criados a partir de pontos com linhas unindo-os. Os pontos são os pares (x, y) das listas de entrada X e Y que passamos para função *plot* do Matplotlib e as linhas são os segmentos retos que conectam dois pontos adjacentes. Os pontos que conectam as retas são chamados de marcadores (*markers*) na terminologia do Matplotlib.

Por padrão (*default*), o Matplotlib desenha os pontos (*markers*) como um único ponto e linhas contínuas como segmentos; há situações em que gostaríamos de alterar o estilo do marcador (para identificá-los claramente na plotagem) ou no estilo da linha (para que a linha apareça tracejada, por exemplo).

A função *plot* suporta um terceiro argumento (opcional) que contém uma *string* de formato para cada par de argumentos X, Y na forma de:

plt.plot (X, Y, '<formato>', ...)

Existem três níveis de personalização: **cores, linhas e marcadores** (Quadro 1, 2 e 3). No quadro 3, um exemplo de criação de gráfico de linhas, com linhas e marcadores customizados (Figura 4a, 4b e 4c).

Quadro 1 – Quadro de cores e suas respectivas abreviações que estão disponíveis para customizar linhas e marcadores dos gráficos

Cor	Abreviação
<i>blue</i> (azul)	b
<i>cyan</i> (ciano)	c
<i>green</i> (verde)	g
<i>black</i> (preto)	k
<i>magenta</i> (magenta)	m
<i>red</i> (vermelho)	r
<i>white</i> (branco)	w
<i>yellow</i> (amarelo)	y

Fonte: Tosi (2019, p. 51).

Quadro 2 – Quadro dos tipos de linhas e seus respectivos caracteres que estão disponíveis para customizar linhas dos gráficos

Tipo da linha	Caractere
Linha cheia	' _ '
Linha tracejada	' - - '
Linha traço-ponto	' - . '
Linha pontilhada	' : '

Fonte: Tosi (2019, p. 53).

Quadro 3 – Quadro dos tipos de marcadores e seus respectivos caracteres disponíveis para customizar marcadores dos gráficos

Tipo da marcador	Caractere
Círculo	'o'
Triângulo	'v', '^', '>', '<', 'x', 'D'
Estrela	'*'
Hexágono	'h'
Losango	's'
Quadrado	's'

Fonte: Tosi (2019, p. 54).

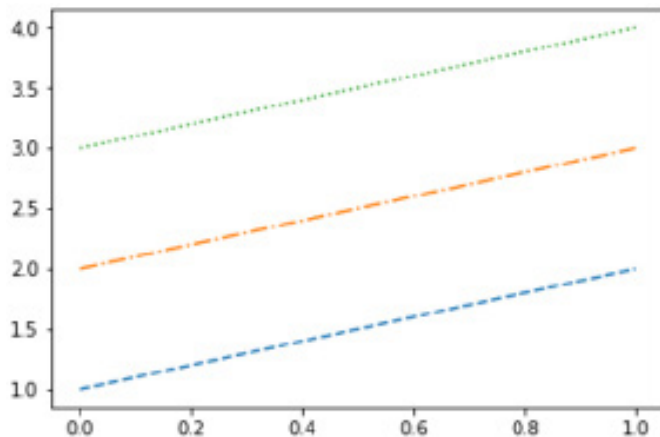
Quadro 4 – Exemplo de código em Python para criação de gráfico de linhas, alterando as propriedades das linhas e marcadores

<pre>import matplotlib.pyplot as plt import numpy as np y = np.arange(1, 3) plt.plot(y, '--', y+1, '-.', y+2, ':'); plt.show()</pre>
<pre>import matplotlib.pyplot as plt import numpy as np y = np.arange(1, 3, 0.3) plt.plot(y, 'cx--', y+1, 'mo:', y+2, 'kp-.'); plt.show()</pre>
<pre>import matplotlib.pyplot as plt import numpy as np y = np.arange(1, 3, 0.2) plt.plot(y, 'x', y+0.5, 'o', y+1, 'D', y+1.5, '^', y+2, 's'); plt.show()</pre>

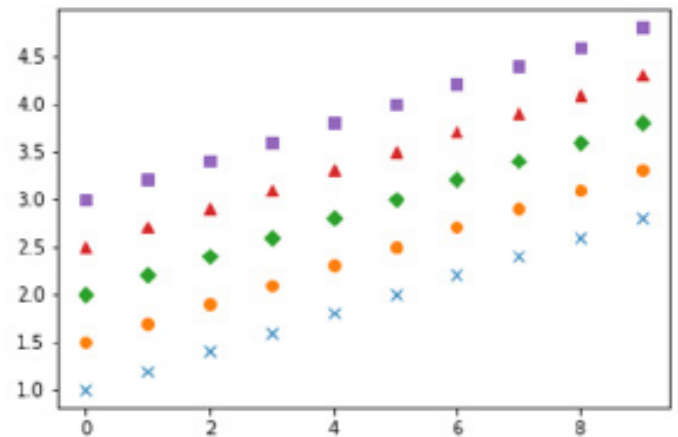
Saída:

Figura 4 – Gráficos de linhas customizados (estilos e cores das linhas), criados a partir de números gerados aleatoriamente:

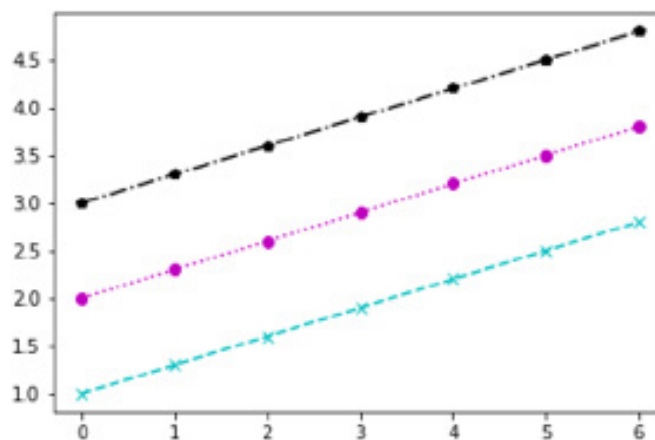
(a) linhas e (b) pontos



(a)



(b)



(c)

Fonte: elaborados pelo autor usando a interface do software Spyder 3.3.3.

1.1.2.4 Exportando os gráficos

Outra funcionalidade bastante útil do Matplotlib é a possibilidade de exportar uma plotagem (gráfico) em um arquivo (.pdf, .jpg ou .png). Utilizando a função *savefig* do Matplotlib, você informa o nome e a extensão e o arquivo será salvo no diretório atual. Veja o exemplo de código-fonte para exportar o gráfico, apresentado no Quadro 4.

Quadro 4 – Exemplo de código-fonte para exportar o gráfico em arquivos

```
# -*- coding: utf-8 -*-  
import matplotlib.pyplot as plt  
  
plt.plot([1, 2, 3])  
  
plt.savefig('grafico.pdf')  
plt.savefig('grafico.jpg')  
plt.savefig('grafico.png')  
  
plt.show()
```

1.1.2.5 Criando gráficos a partir de informações em arquivos

Muitas vezes, vamos precisar representar graficamente os dados contidos em um arquivo de texto ou binário (Figura 5). Existem muitos tipos de arquivos e várias maneiras de extrair dados de um arquivo para representá-lo graficamente em Matplotlib. Nos exemplos a seguir, vamos demonstrar algumas maneiras de fazer isso. Primeiro, usaremos o módulo CSV (Quadro 5) para carregar arquivos de texto (Quadro 6), depois mostraremos como fazer a leitura dos dados do arquivo, usando a biblioteca NumPy (quadro 7).

Quadro 5 – Conteúdo do arquivo exemplo.txt usando no desenvolvimento do código-fonte (quadros 6 e 7)

```
1,5  
2,3  
3,4  
4,7  
5,4  
6,3  
7,5  
8,7  
9,4  
10,4
```

Quadro 6 – Exemplo de código-fonte para criação de gráficos a partir de dados no arquivo de texto, usando a biblioteca Matplotlib

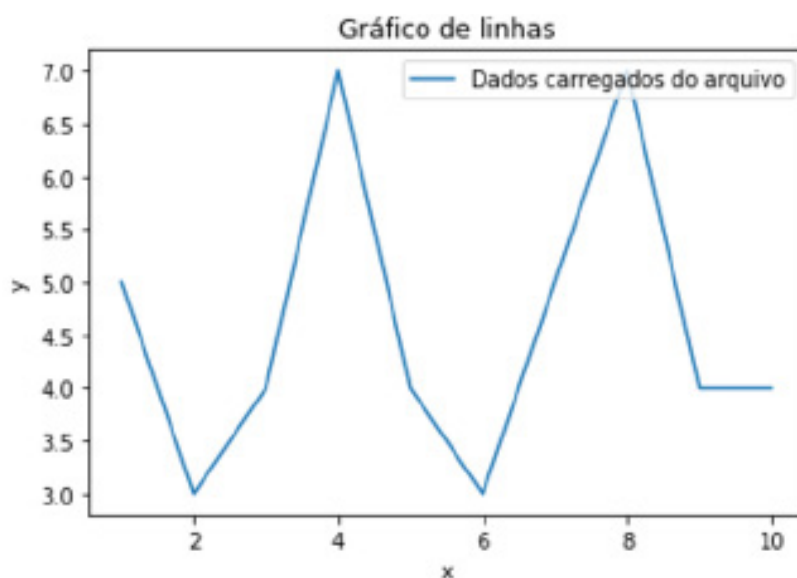
```
# -*- coding: utf-8 -*-  
import matplotlib.pyplot as plt  
import csv  
  
x,y = [], []  
  
with open('exemplo.txt','r') as csvfile:  
    plots = csv.reader(csvfile, delimiter=',')  
    for row in plots:  
        x.append(int(row[0]))  
        y.append(int(row[1]))  
  
plt.plot(x,y, label='Dados carregados do arquivo')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.title(u'Gráfico de linhas')  
plt.legend()  
plt.show()
```

Quadro 7 – Exemplo de código-fonte para criação de gráficos a partir de dados no arquivo de texto, usando a biblioteca NumPy

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x, y = np.loadtxt('exemplo.txt', delimiter=',', unpack=True)  
plt.plot(x,y, label='Loaded from file!')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.title(' u'Gráfico de linhas')  
plt.legend()  
plt.show()
```

Saída:

Figura 5 – Gráfico de linha gerado a partir de dados armazenados em um arquivo de texto



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.2 Introdução a Pandas

O Pandas é um pacote Python que fornece estruturas de dados rápidas, flexíveis e expressivas, projetadas para facilitar o trabalho com dados relacionais. Os dados manipulados no Pandas são frequentemente usados para trabalhar com análises estatísticas no SciPy, plotando funções do Matplotlib e algoritmos de aprendizado de máquina no *Scikit-learn* (MCKINNEY, 2019).

A biblioteca Pandas contém um conjunto de ferramentas para manipulação de arquivos de diferentes formatos: CSV (*Comma-Separated Values*, são arquivos de texto que fazem uma ordenação de bytes, separando os valores com vírgula), TXT, JSON (*JavaScript Object Notation* é um formato compacto, de padrão aberto independente, utilizado para troca de dados simples e rápida entre sistemas), Microsoft Excel, bancos de dados SQL e o formato HDF5.

Existem duas estruturas de dados principais no *Pandas*: *Series* e *DataFrames*. O objeto *Series* é um array de 1 dimensão (1D).

Você pode considerar um objeto *Series* também como uma coluna de uma tabela. Já o *DataFrame* é um encapsulamento da função *Series* que se estende a duas dimensões (2D). Ele pode ser criado usando várias entradas, como listas, dicionários, series, *arrays* (NumPy) ou outro *DataFrame*.

1.2.1 Instalação

A maneira mais fácil de instalar a biblioteca Pandas é como parte da distribuição Anaconda, uma distribuição de plataforma para análise de dados e computação científica. Este é o método de instalação recomendado para a maioria dos usuários. As instruções para instalação do código-fonte, PyPI, ActivePython, várias distribuições Linux ou uma versão de desenvolvimento são também fornecidas. Para mais informações visite o site do Pandas e acesse a seção *Documentation*.

1.2.2 Criando gráficos e histogramas

Assim como a biblioteca Matplotlib, a biblioteca Pandas contém diversas funções, que vão o ajudar a produzir diferentes tipos de gráficos e histogramas de forma simples e rápida.

1.2.2.1 Histograma

O histograma é uma distribuição de frequência de um conjunto de dados previamente tabulado. Sua representação gráfica é feita em colunas (ou em barras) dividida em classes (categorias) uniformes ou não uniformes. O histograma é considerado uma ferramenta de qualidade muito importante para análises estatísticas e bastante utilizado na área de ciência de dados e na área de processamento de imagens.

Na biblioteca Pandas, os histogramas podem ser desenhados usando os métodos *DataFrame.plot.hist* e *Series.plot.hist*. O exemplo a seguir (Quadro 8), mostra um exemplo de código-fonte para criação de histogramas (Figura 6a e 6b).

Quadro 8 – Exemplo de código-fonte para criação de histogramas

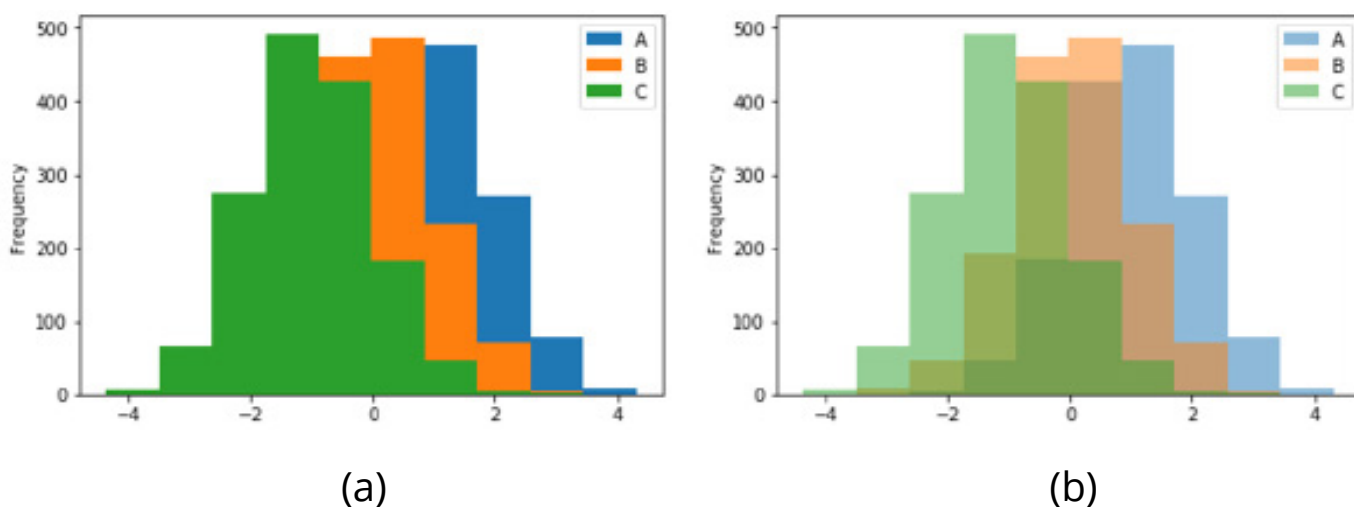
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df4 = pd.DataFrame({'A': np.random.randn(1500) + 1, 'B': np.random.randn(1500), 'C':
np.random.randn(1500) - 1}, columns=['A', 'B', 'C'])

plt.figure();
df4.plot.hist(stacked=False, bins=10)
df4.plot.hist(alpha=0.5)
```

Saída:

Figura 6 – Histogramas criados a partir de 3 conjuntos de dados (A, B, C) gerados aleatoriamente



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.2.2.2 Gráfico de linhas

O gráfico de linha é um tipo de gráfico que exibe informações com uma série de pontos (marcadores) ligados por segmentos de linha reta. Os gráficos de linhas são normalmente utilizados para analisar ou controlar alterações ao longo do tempo e para facilitar a identificação de tendências ou de anomalias em um conjunto de dados (Figura 7).

A seguir, um exemplo de código-fonte desenvolvido para criar gráficos de linhas (Quadro 9), usando as estruturas *DataFrame* e *Series* do Pandas.

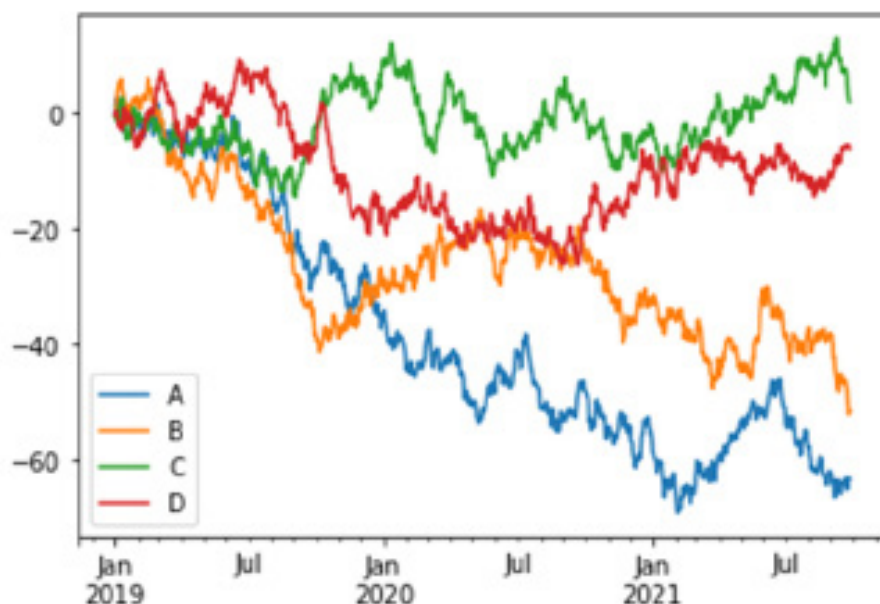
Quadro 9 – Exemplo de código-fonte para criação de gráficos de linhas

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

ts = pd.Series(np.random.randn(1000), index=pd.date_range('01/01/2019', periods=1000))
df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=list('ABCD'))
df = df.cumsum()
plt.figure();
```

Saída:

Figura 7 – Gráficos de linhas gerados a partir de 4 conjuntos de valores (A, B, C, D) gerados aleatoriamente e distribuídos ao longo do tempo em uma série da biblioteca Pandas



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.2.2.3 Gráfico de barras (Bar plot)

O gráfico de barras é um tipo de gráfico com barras retangulares (horizontais ou verticais) e comprimento proporcional aos valores que

ele representa. Quanto maior o comprimento de uma barra, maior o valor que representa; quanto menor o comprimento, menor o valor. O gráfico de barras pode ser usado para comparar quantidades de uma determinada categoria de valores, na apresentação de pesquisas de intenção de votos ou opinião pelas redes de televisão.

A seguir, um exemplo de código-fonte (Quadro 10) para criar gráficos de barras (Figura 8).

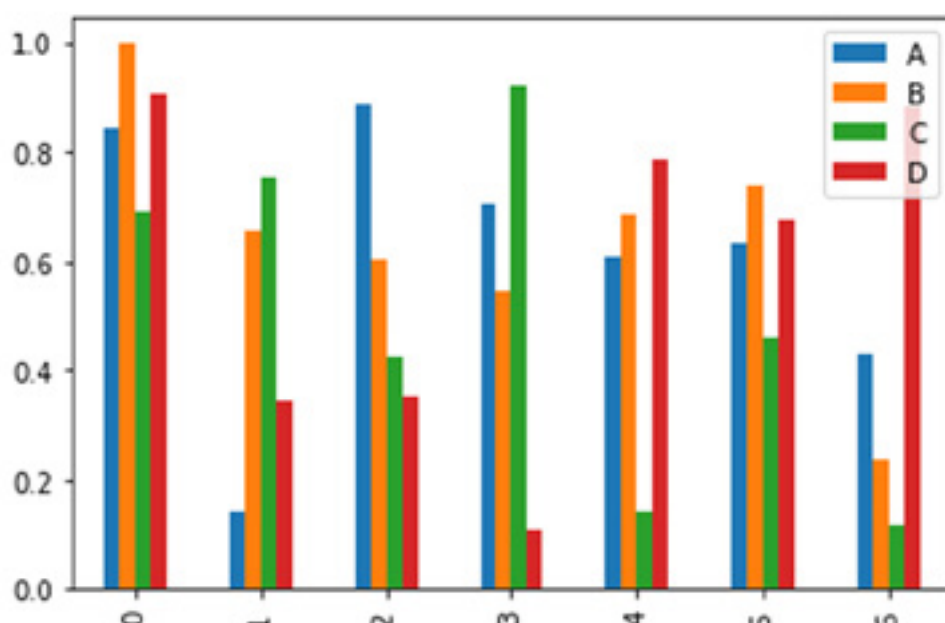
Quadro 10 – Exemplo de código-fonte para criação de gráficos de barras

```
import numpy as np
import pandas as pd

df2 = pd.DataFrame(np.random.rand(7, 4), columns=['A', 'B', 'C', 'D'])
df2.plot.bar();
```

Saída:

Figura 8 – Gráficos de barras desenvolvidos a partir de 4 conjuntos de dados (A,B, C e D) gerados aleatoriamente



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.2.2.4 Gráfico de área (Area plot)

Um gráfico de área é formado por gráficos de linhas, no qual a área entre a linha e o eixo é representada com uma cor. Estes gráficos normalmente são utilizados para representar os totais acumulados e enfatizar a magnitude das mudanças ao longo do período.

A seguir, um exemplo de código-fonte (Quadro 11) para criar gráficos do tipo área (Figura 9a e 9b).

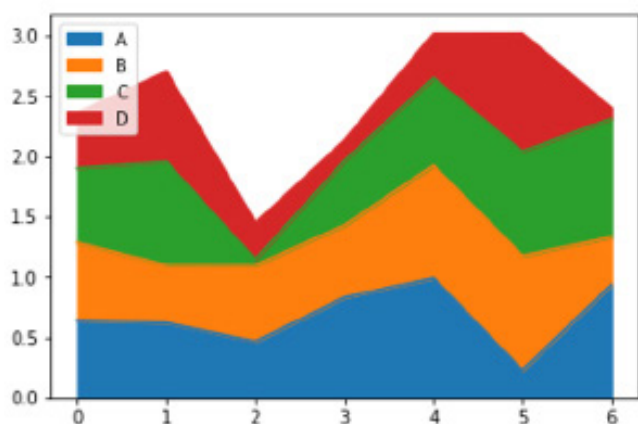
Quadro 11 – Exemplo de código-fonte para criação de gráficos de área

```
import numpy as np
import pandas as pd

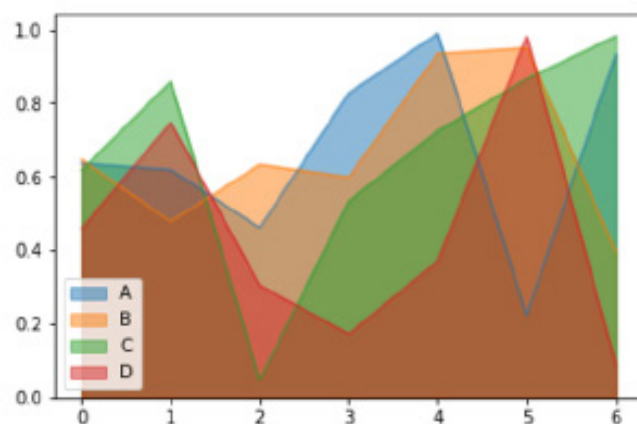
df2 = pd.DataFrame(np.random.rand(7, 4), columns=['A', 'B', 'C', 'D'])
df2.plot.area();
df2.plot.area(stacked=False);
```

Saída:

Figura 9 – Gráficos de área desenvolvidos a partir de 4 conjuntos de dados (A, B, C e D) gerados aleatoriamente



(a)



(b)

Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.2.2.5 Gráfico de pizza

O gráfico de pizza é um dos mais utilizados devido à facilidade de visualização e compreensão dos dados. Neste gráfico, são necessárias duas ou mais categorias e um valor correspondente para cada uma delas.

A seguir, um exemplo de código-fonte desenvolvido (Quadro 12) para criar gráficos de pizza (Figura 10).

Quadro 12 – Exemplo de código-fonte para criação de gráficos de pizza.

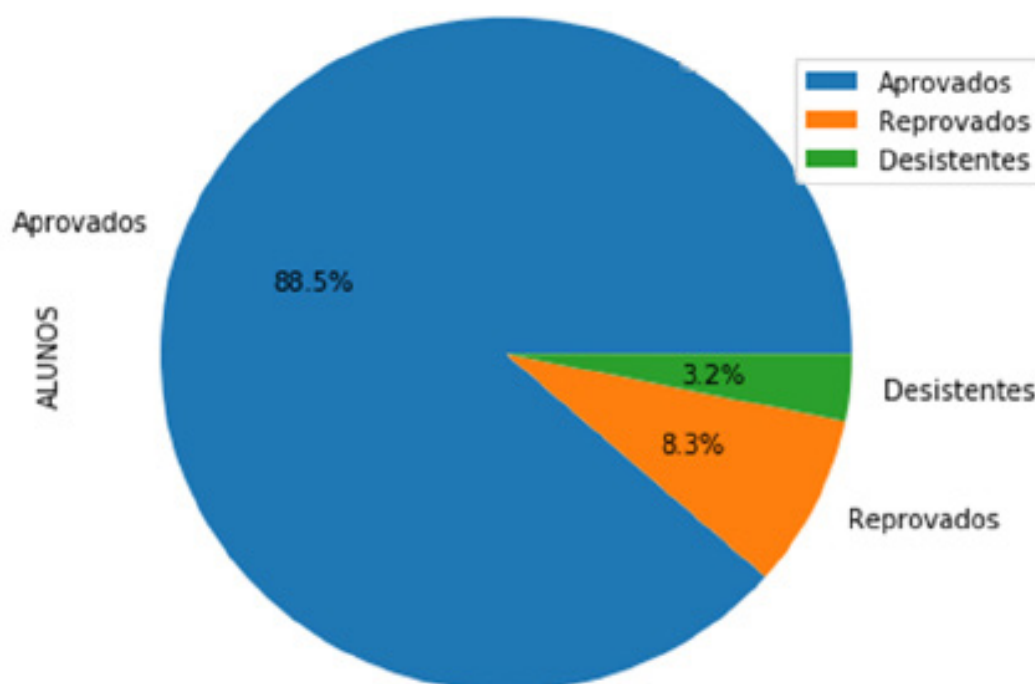
```
import numpy as np
import pandas as pd

df = pd.DataFrame({'ALUNOS': [88.5, 8.3, 3.2]}, index=['Aprovados',
'Reprovados', 'Desistentes'])

plot = df.plot.pie(y='ALUNOS', figsize=(6, 6), autopct='%1.1f%%')
```

Saída:

Figura 10 – Gráfico de pizza utilizado para representar a situação dos alunos no final de semestre de um determinado curso de pós-graduação



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.2.3 Manipulação de arquivos

A API de I/O da biblioteca Pandas é um conjunto de funções para manipulação de arquivos de diferentes formatos. O quadro 4 exibe um conjunto de funções disponíveis para manipular os tipos de arquivos mais conhecidos.

Quadro 4 – Funções da biblioteca Pandas para manipulação de arquivos

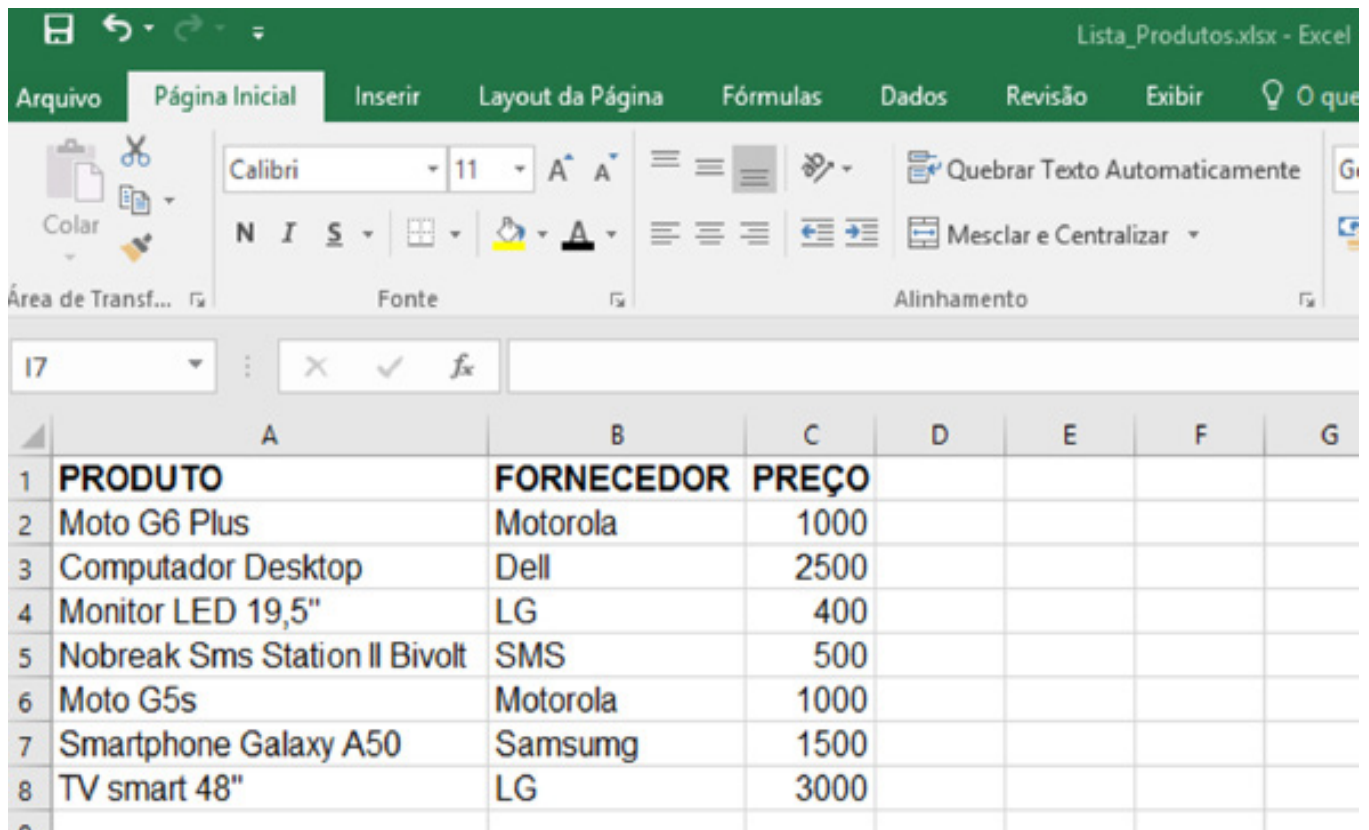
Formato	Descrição	Função para leitura	Função para escrita
Texto	CSV	<i>read_csv</i>	<i>to_csv</i>
Texto	JSON	<i>read_json</i>	<i>to_json</i>
Texto	HTML	<i>read_html</i>	<i>to_html</i>
Binário	MS Excel	<i>read_excel</i>	<i>to_excel</i>
Binário	Open Document Spreadsheet (ODS)	<i>read_excel</i>	-
Binário	HDF5 Format	<i>read_hdf</i>	<i>to_hdf</i>
Binário	Python Pickle Format	<i>read_pickle</i>	<i>to_pickle</i>
SQL	SQL	<i>read_sql</i>	<i>to_sql</i>
SQL	Google Big Query	<i>read_gbq</i>	<i>to_gbq</i>

Fonte: McKinney (2019, p. 181).

1.2.3.1 Leitura e escrita em arquivos MS Excel

Nesta seção vamos aprender a usar a função *read_excel* para ler dados de um arquivo de planilha MS Excel. Os Quadros 13 e 14 mostram exemplos de código-fonte em Python usando a biblioteca Pandas, para fazer a leitura de dados (Figura 11) e escrita de dados em planilha MS Excel.

Figura 11 – Planilha MS Excel com uma listagem de produtos



	A	B	C	D	E	F	G
1	PRODUTO	FORNECEDOR	PREÇO				
2	Moto G6 Plus	Motorola	1000				
3	Computador Desktop	Dell	2500				
4	Monitor LED 19,5"	LG	400				
5	Nobreak Sms Station II Bivolt	SMS	500				
6	Moto G5s	Motorola	1000				
7	Smartphone Galaxy A50	Samsung	1500				
8	TV smart 48"	LG	3000				

Fonte: elaborada pelo autor, utilizando o MS Excel.

Quadro 13 – Exemplo de código-fonte para leitura de dados de uma planilha MS Excel usando a biblioteca do Pandas

```
import pandas as pd

df = pd.read_excel('Lista_Produtos.xlsx')
df.head()

# Imprimindo os valores da coluna 'PRODUTO'
print("PRODUTO")
for i in df.index:
    print(df["PRODUTO"][i])
```

Quadro 14 – Exemplo de código-fonte em Python para gravar dados em planilha MS Excel, a partir de informações contida em um objeto *DataFrame* (Pandas)

```
import pandas as pd

df = pd.DataFrame({'PRODUTO':['HD Externo 4TB', 'Fone de Ouvido JBL Free Bluetooth - Preto'],
                  'FORNECEDOR':['Seagate', 'Harman do Brasil'],
                  'PRECO':[500, 520]})

df.to_excel('Lista_Produtos_2.xlsx')
```

PARA SABER MAIS



O Matplotlib e suas dependências estão disponíveis como pacotes para as distribuições do MacOS X, Windows e Linux. Você pode baixar e instalar manual ou instalar utilizando o comando *pip* (recomendado). Abra o terminal do seu sistema operacional e digite os seguintes comandos:

```
python -m pip install -U pip
python -m pip install -U numpy
python -m pip install -U matplotlib
```

ASSIMILE



É possível a criação de gráficos customizados e manipulação de arquivos (CSV, JSON, MS Excel etc.) usando **Python** e as bibliotecas **Matplotlib** e **Pandas**. Ambas APIs são amplamente utilizadas pelos cientistas de dados. Portanto, a escolha de qual biblioteca utilizar vai depender de suas necessidades e características do projeto que você esteja desenvolvendo.



TEORIA EM PRÁTICA

Atualmente, as planilhas MS Excel são uma ferramenta bastante utilizada no controle e planejamento de diversos setores das empresas. Entretanto, as planilhas, em muitos casos, não são suficientes para armazenar todas as informações conforme o tamanho da empresa. Muitas vezes, é necessário buscar uma solução mais robusta e mais confiável.

Diante desse cenário, imagine que você foi contratado pela empresa XPTO para fazer a integração dos dados, que atualmente estão salvos em planilhas MS Excel e salvar essas informações em um sistema ERP (*Enterprise Resource Planning*).

Utilizando Python e as bibliotecas Matplotlib e Pandas, como você faria a integração dos dados armazenados nas planilhas Excel para o sistema ERP?



VERIFICAÇÃO DE LEITURA

1. Sobre a biblioteca Matplotlib, analise as seguintes afirmações:
 - I. A API do Matplotlib contém funções para obter informações de páginas HTML.
 - II. *DataFrames* e *Series* são as principais estruturas de dados no Matplotlib.

- III. Na versão atual do Matplotlib é possível criar plotagem em 3D.
- IV. É possível exportar as plotagens desenvolvidas em Matplotlib, utilizando a função *export_plot*.

a. Todas as afirmações são corretas.

b. I - II - IV

c. Apenas I

d. I - III

e. I - IV

2. A API Pandas fornece estruturas de dados de alto desempenho e fáceis de usar para o desenvolvimento de gráficos, diagramas, animação e também manipulação de arquivos.

Sobre a biblioteca Pandas, assinale a alternativa correta:

a. Pandas não fornece funcionalidades para manipulação de arquivos *Open Document Spreadsheet (ODS)*.

b. *DataFrame*, *DataSet*, *Series* e *Resilient Distributed Dataset (RDD)* são as principais estruturas de dados no Pandas.

c. Histograma é uma representação da distribuição de frequências dos dados. A função *pandas.DataFrame.hist* recebe com parâmetro um objeto *DataFrame* e retorno é um histograma, para cada série do *DataFrame*.

- d. Um objeto Series é uma estrutura de dados de 2 dimensões (2D) que representa linhas e colunas.
 - e. A API Pandas não tem suporte para a estrutura de dados dicionário do Python.
3. O Pandas é uma biblioteca amplamente utilizada na área de ciências de dados. Suas principais características são: desempenho e capacidade de simplificar tarefas complexas de manipulação de dados. Sobre a estrutura *pandas.DataFrame*, assinale a alternativa correta:
- a. Usando as funções *pandas.DataFrame.read_csv* e *pandas.DataFrame.show* podemos fazer a leitura e visualizar todas as linhas do contidas em um arquivo CSV.
 - b. Através do método *pandas.DataFrame.remove*, podemos apagar colunas ou linhas de um objeto *DataFrame*.
 - c. Pandas fornece a função *pandas.DataFrame.isnull* para a verificação se todas as linhas e colunas do objeto contêm valores faltantes ou nulos.
 - d. Na biblioteca Pandas não é possível a criação de subgráficos (*subplots*).
 - e. A função *pandas.DataFrame.stats* retorna as estatísticas descritivas das colunas *DataFrame*.

► Referências bibliográficas

HUNTER, J.; DALE, D.; FIRING, E.; DROETTBOOM, M. **Matplotlib User's Guide**. Matplotlib Release 3.1.1, 2019.

MCKINNEY, W. **Pandas: powerful Python data analysis toolkit. Release 0.25.3**. Python for High Performance and Scientific Computing, 2019.

ROUGIER, N, P. Scientific Visualization – Python & Matplotlib. **Scientific Python**. Volume II, 2019.

TOSI, S. **Matplotlib for Python developers**. Packt Publishing Ltd., 2009.

YIM, A.; CHUNG, C.; YU, A. **Matplotlib for Python Developers**: Effective techniques for data visualization with Python. Packt Publishing Ltd., 2018.

► Gabarito

Questão 1 – Resposta: D

Resolução: A API do Matplotlib contém funções para obter informações de páginas HTML, através da função *read_html* e *to_html*.

Na versão atual do Matplotlib é possível criar plotagem em 2D e 3D. Portanto as opções I e III são verdadeiras.

DataFrames e *Series* são as principais estruturas de dados no Pandas e não do Matplotlib.

É possível exportar as plotagens desenvolvidas em Matplotlib, utilizando a função *savefig*. Portanto, as opções II e IV são falsas.

Questão 2 – Resposta: C

Resolução: Histograma é uma representação da distribuição de dados. A função *pandas.DataFrame.hist* recebe como parâmetro um objeto *DataFrame* e o retorno é um histograma, para cada serie do *DataFrame*. Portanto a opção c é verdadeira.

Pandas fornece suporte para manipulação de arquivos *Open Document Spreadsheet (ODS)*. Portanto, a opção a é falsa.

Questão 3 – Resposta: C

Resolução: Pandas fornece a função *pandas.DataFrame.isnull* para a verificação de todas as linhas e colunas do objeto, se contêm valores faltantes ou nulos. Portanto a opção c é verdadeira.

A função *pandas.DataFrame.head* retorna todas as linhas contidas em um arquivo CSV. A opção a é falsa.

Através do método *pandas.DataFrame.drop*, podemos apagar colunas ou linhas de um objeto *DataFrame*. A opção b é falsa.

A biblioteca Pandas fornece suporte para criação de subgráficos, portanto a opção d é falsa.

A função *pandas.DataFrame.describe* retorna as estatísticas descritivas das colunas *DataFrame*. A opção e é falsa.



Análise estatísticade dados

Autor: Danilo Rodrigues Pereira

► Objetivos

Análise estatística de dados é uma área que nos ajuda a entender melhor os dados e as informações (valores) contidas neles. Estatística e probabilidade são muito importantes para quase todas as áreas do conhecimento (biologia, medicina, engenharia, física, psicologia, matemática, computação etc.). O objetivo dessa unidade é fornecer conhecimentos teóricos e práticos necessários para você analisar e processar dados utilizando métodos estatísticos, usando a linguagem de programação Python. Ao final da unidade você será capaz de:

- Conhecer os principais métodos estáticos que podem ser aplicados na área de ciência de dados.
- Utilizar a linguagem Python para analisar e processar os dados utilizando métodos estáticos.
- Organizar os dados para análise estatística.

► 1. Introdução à análise estatística dos dados

Conhecimentos básicos em estatísticas e probabilidade são extremamente importantes para trabalhar na área de ciência de dados. A probabilidade é uma área da Matemática que estuda as chances de ocorrência de experimentos ou eventos. É por meio de probabilidade, por exemplo, que podemos saber a chance de ganharmos na loteria, de obter cara ou coroa no lançamento de uma moeda e até cálculo de erro em pesquisas. Já a estatística é a área responsável pela coleta, organização e interpretação de dados experimentais e pela extrapolação dos resultados da amostra para a população. Através da estatística podemos extrair informação dos dados para obter melhor compreensão das situações que representam. A linguagem de programação Python é considerada uma das melhores linguagens de programação que fornecem suporte para trabalhar com análise estatística dos dados (UNPINGCO, 2016; DOWNEY, 2011; DUCHESNAY, 2018; GRUS, 2019).

As bibliotecas NumPy e o Pandas revolucionaram o processamento de dados no ecossistema Python. À medida em que os dados e os sistemas tornam-se mais complexos, a movimentação e o processamento desses dados tornam-se mais difíceis. O Python possui ferramentas para trabalhar conjuntos de dados na memória, mas inevitavelmente queremos escalar esse processamento e tirar proveito de hardware. É aqui que a ferramenta Blaze é útil, ao fornecer uma interface uniforme para uma variedade de tecnologias para processar e analisar dados em *batch* ou *streaming* (CHRISTINE, 2015). Os *back-end* suportados incluem bancos de dados como Postgres ou MongoDB, sistemas de armazenamento em disco como PyTables, BColz e HDF5 ou sistemas distribuídos como Hadoop e Spark (Figura 1).

Existem duas maneiras de instalar as bibliotecas NumPy, Pandas, SciPy, Matplotlib e Statsmodels: usando a distribuição científica do Python ou via comando **pip**.

- **Distribuição científica (recomendado): Anaconda** é uma distribuição *open-source* que contém um pacote de ferramentas (NumPy, Spyder IDE, Jupyter Notebook, SciPy, Pandas, Matplotlib, Scikit-learn) para área da ciência de dados e pode ser instalado nas plataformas Linux, Windows ou macOS. O Anaconda fornece uma versão *lite* chamada Miniconda, que ainda fornece acesso ao gerenciador de pacotes Conda. Para mais informações, acesse o site oficial do Anaconda e visite a seção *Download*.
- **Instalação via comando *pip*** – Por padrão, o Python vem com um sistema de gerenciamento de pacotes embutido chamado *pip*. Através do comando *pip* podemos instalar, atualizar ou excluir qualquer pacote oficial do Python. Você pode instalar pacotes através da linha de comando digitando:

```
python -m pip install --user numpy scipy matplotlib pandas statsmodels
```

Figura 1 – Tecnologias utilizadas no processamento de dados na área de ciências de dados



Fonte: adaptada de CHRISTINE (2015).

1.1.1 NumPy

O NumPy é uma poderosa biblioteca do Python usada principalmente para realizar cálculos em vetores (*arrays*) e matrizes multidimensionais. O NumPy fornece um conjunto de funções que ajudam os programadores a executar de forma simples cálculos numéricos. Esses tipos de cálculos são amplamente utilizados para resolver problemas nas áreas de *deep learning*, *machine learning*, processamento de imagens e computação gráfica (VAN DER WALT, 2011). Para mais informações, visite o site oficial do NumPy e acesse as seções *Getting Started* e *Documentation*.

A biblioteca NumPy fornece algumas funções para criação de *arrays*, que são: ***numpy.arange***, ***numpy.array***, ***numpy.ones*** e ***numpy.zeros***. A seguir, alguns exemplos em Python.

- ***numpy.arange(start, stop, step, dtype=None)*** - Cria um *array* começando 0 (*start*) até o valor *stop* incrementando em 1 (*step*).

Código-fonte

```
import numpy as np
A = np.arange(10)
print A
```

Saída:

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

- ***numpy.array(objeto)*** - Cria um *array* a partir de objeto: listas (mutáveis), tuplas (imutáveis), etc.

Código-fonte

```
import numpy as np
A = np.array([0,1,2,3,4,5,6,7,8,9])
print A
```

Saída:

[0 1 2 3 4 5 6 7 8 9]

Outra maneira de criar vetores (*arrays*) ou matrizes em Numpy é extraindo informações contidas em arquivos de texto. Vamos utilizar nos exemplos apenas função *genfromtxt*. Resumidamente, a função *genfromtxt* executa dois loops: o primeiro loop converte cada linha do arquivo em uma sequência de caracteres e o segundo loop converte cada sequência no tipo de dados apropriado.

Exemplo – Criação de uma matriz (4x3) a partir de informações contidas em um arquivo de texto: **Arquivo: teste.txt**

```
# Início do arquivo
1, 2, 3
4, 5, 6
7, 8, 9
10, 11, 12
# Fim do arquivo
```

Código-fonte

```
# -*- coding: utf-8 -*-
import numpy as np
from StringIO import StringIO
arquivo = '<caminho_do_arquivo>/teste.txt'
A = np.genfromtxt(arquivo, delimiter=",", dtype=int, comments='#')
print 'Matriz = \n', A
print '\nDimensões = ', A.shape
```

Saída:

Array =

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Dimensões = (4L, 3L)

PARA SABER MAIS



Em alguns casos, não estamos interessados em todas as colunas contidas no arquivo, mas apenas em algumas delas. Podemos selecionar quais colunas importar com o argumento ***usecols***. Este argumento aceita um único número inteiro ou uma sequência de números inteiros correspondentes aos índices das colunas a serem importadas. Por exemplo, se queremos importar apenas a primeira e a última coluna, podemos usar ***usecols = (0, -1)***.

1.1.2 Pandas

A biblioteca Pandas é uma das ferramentas mais importantes à disposição dos cientistas e analistas de dados que trabalham atualmente em Python. Os dados manipulados no *Pandas* são frequentemente usados para trabalhar com análises estatísticas no SciPy, plotando funções do Matplotlib e algoritmos de aprendizado de máquina no *scikit-learn* (MCKINNEY, 2011). Existem duas estruturas de dados principais na biblioteca Pandas, são elas: *Series* e *DataFrames*. O objeto *Series* combina um índice e valores de dados correspondentes. Já o *DataFrame* é um encapsulamento da função *Series* que se estende a duas dimensões. Ele pode ser criado usando várias entradas, como listas, dicionários, series, arrays (NumPy) ou outro *DataFrame*. Para mais informações e exemplos, acesse o site oficial do Pandas e visite a seção *Documentation*.

1.1.3 SciPy

O SciPy é o pacote principal de rotinas científicas em Python, que se destina a operar de forma eficiente em matrizes NumPy (JONES, 2001). Utiliza como base a biblioteca Numpy para lidar eficientemente com grandes quantidades de números de maneira eficiente. A biblioteca SciPy contém um conjunto de funções para trabalharmos com estatísticas, pesquisa operacional, otimização, processamento de sinais e imagens, solução de equações diferenciais, polinômios etc.

1.1.4 Matplotlib

O matplotlib é a principal ferramenta de visualização bidimensional (2D) de gráficos científicos em Python. Como todos os grandes projetos de *open-source*, ele se originou para satisfazer uma necessidade pessoal. Seu criador John Hunter usou principalmente o MATLAB para visualização científica, mas como ele começou a integrar dados de fontes diferentes usando Python, percebeu que precisava de uma solução Python para visualização; ele escreveu sozinho o Matplotlib (UNPINGCO, 2016). O Matplotlib pode ser usado em scripts Python, nos *shell* Python e IPython, no notebook Jupyter e servidores de aplicativos da *web* para desenvolvimento de interface gráfica do usuário. Para informações sobre instalação e tutorial, acesse o site oficial e visite as seções: *Installation, Documentation e Tutorials*.

1.1.5 Statsmodels

Statsmodels é um módulo Python que fornece classes e funções para a estimativa de muitos modelos estatísticos, bem como para a realização de testes e a exploração de dados estatísticos. Desde a versão 0.5.0 do *statsmodels*, é possível usar fórmulas da linguagem de programação R junto com os *dataframe* da biblioteca Pandas para ajustar seus modelos (SEABOLD, 2010). Para maiores informações, acesse o site oficial do Statsmodels e visite a seção *Documentation*.

1.2 Análise estatística dos dados utilizando Python

É difícil trabalhar com ciência de dados sem algum tipo de entendimento de probabilidade e estatística. Para nossos propósitos como cientistas de dados, você deve pensar em probabilidade e estatística como uma maneira de quantificar a incerteza associada a eventos escolhidos em algum universo de eventos e aprender utilizar os métodos estatísticos para entender e extrair informações dos dados. Nesta seção vamos aprender quais as diferenças das métricas, como a média, mediana, moda, desvio padrão e variância, utilizando Python (Figura 2).

1.2.1 Média, mediana, moda, desvio padrão e variância

- a. Média:** Seja um conjunto de n valores (x_i), a média (\bar{x}) é a soma de todos os valores dividido pelo número de elementos desse conjunto, no qual:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Código-fonte

```
# -*- coding: utf-8 -*-

import scipy
import numpy as np

lista_idades = np.array([20, 49, 41, 33, 25, 10, 29, 40])
media = scipy.mean(lista_idades)
print "A média das idades é: ", round(media, 3)

A = [[1, 3, 27], [3, 4, 6], [7, 6, 3], [3, 6, 8]]

print "\nA média da matriz A é:», round(scipy.mean(A),3)
print "\nMédia de cada coluna da matriz A\n", scipy.mean(A, axis = 0)
print "\nMédia de cada linha da matriz A\n", scipy.mean(A, axis = 1)
```

Saída:

A média das idades é: 30.875

A média da matriz A é: 6.417

Média de cada coluna da matriz A

[3.5 4.75 11.]

Média de cada linha da matriz A

[10.33333333 4.33333333 5.33333333 5.66666667]

- b. Mediana:** É o valor que separa a metade maior e a metade menor de uma amostra, uma população ou uma distribuição de probabilidade. Em outras palavras, a mediana pode ser o valor do meio de um conjunto de dados. A mediana é usada principalmente em distribuições distorcidas, que resumem a tendência central dos dados diferentemente da média aritmética.

Código-fonte

```
# -*- coding: utf-8 -*-  
import statistics  
import numpy as np  
  
A = [1, 3, 5, 7, 10, 15]  
  
print "A mediana do conjunto é: ", statistics.median(A)  
print "A mediana (baixa) do conjunto é: ", statistics.  
median_low(A)  
print "A mediana (alta) do conjunto é: ", statistics.median_high(A)
```

Saída:

A mediana do conjunto é: 6.0

A mediana (baixa) do conjunto é: 5

A mediana (alta) do conjunto é: 7

c. Moda: É o elemento que ocorre com mais frequência em uma amostra, população ou distribuição.

Código-fonte

```
# -*- coding: utf-8 -*-  
import statistics  
import numpy as np  
  
A = [1, 3, 5, 5, 7, 10, 10, 10, 15]  
print "A moda do conjunto A é: ", statistics.mode(A)  
  
B = [1, 3, 5, 5, 7, 10, 10, 15]  
print "A moda do conjunto do conjunto B: ", statistics.mode(B)  
print "A moda da sequência de caracteres é: ", statistics.  
multimode('AABBBBCCDDDDDEEEFFFGG')
```

Saída:

A moda do conjunto A é: 10

StatisticsError: no unique mode; found 2 equally common values

A moda da sequência de caracteres é: ['B', 'D', 'F']

PARA SABER MAIS



A função `statistics.mode` retorna o erro `StatisticsError` quando mais de um valor moda é encontrado, ou seja, quando mais de um elemento do conjunto tem a mesma frequência de ocorrência. Exemplo: `A = [1, 3, 5, 5, 7, 10, 10, 15]`. Nesse caso, você deve utilizar a função `statistics.multimode`.

d. Variância: É uma medida de dispersão e é usada também para expressar o quanto um conjunto de dados se desvia da média. O cálculo da variância é obtido através da soma dos quadrados da diferença entre cada valor e a média aritmética (\bar{x}), dividida pela quantidade de elementos na amostra (n):

$$\sigma^2 = \frac{1}{n} \sum_i (x_i - \bar{x})^2$$

Código-fonte

```
import statistics
import numpy as np

idades = [10, 30, 55, 15, 17, 22, 38, 41, 15]

print "O desvio padrão do conjunto de idades é: ", round(statistics.pstdev(idades), 3)
```

Saída: o desvio padrão do conjunto de idades é: 14.189

e. Desvio padrão: É uma medida que expressa o grau de dispersão de um conjunto de dados ou amostra. Em outras palavras, o desvio padrão (DP) indica o quanto um conjunto de dados é uniforme. Quanto mais próximo de 0 for o desvio padrão, mais homogêneos são os dados. O desvio padrão (σ) é definido como a raiz quadrada da variância (σ^2).

$$\sigma = \sqrt{\sigma^2}$$

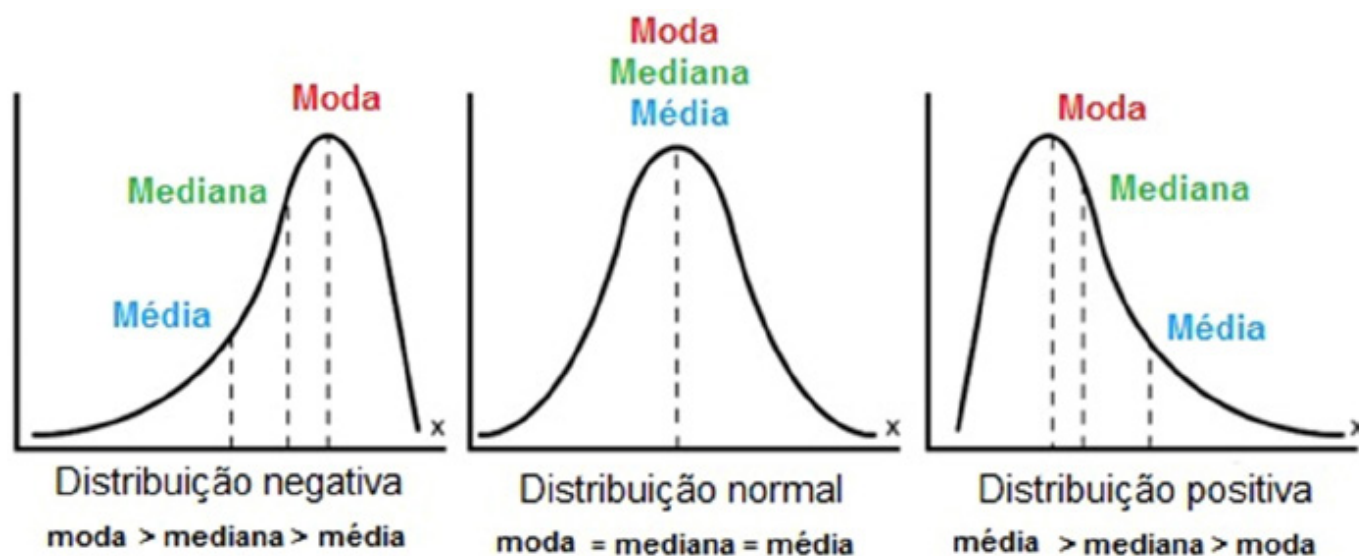
```
# -*- coding: utf-8 -*-
import statistics
import numpy as np

A = [0.0, 0.25, 0.25, 1.25, 1.5, 1.75, 2.75, 3.25]

print "A variância do conjunto de dados A é: ", round(statistics.variance(A),3)
```


Saída: a variância do conjunto de dados A é: 1.429

Figura 2 – Comparação entre moda, mediana e média nas distribuições negativa, normal e positiva



Fonte: elaborada pelo autor.

1.2.2 Histogramas

Histograma é uma representação gráfica (barras verticais ou horizontais) da distribuição de frequências de um conjunto de dados. Dependendo do tipo de dados que estamos trabalhando ou do problema que queremos resolver, usamos uma ferramenta diferente. Assim, o primeiro passo para criar um histograma é sempre coletar dados. Utilizando Python é possível criar histogramas de diversas maneiras, elas são: **Numpy, Pandas ou Matplotlib**.

Exemplo: Histograma de *scores* de teste de quociente de inteligência (Figura 3).

Código-fonte

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# mu = média da distribuição
# sigma = desvio padrão da distribuição
mu, sigma = 100, 15
x = mu + sigma*np.random.randn(10000) #10.000 amostras

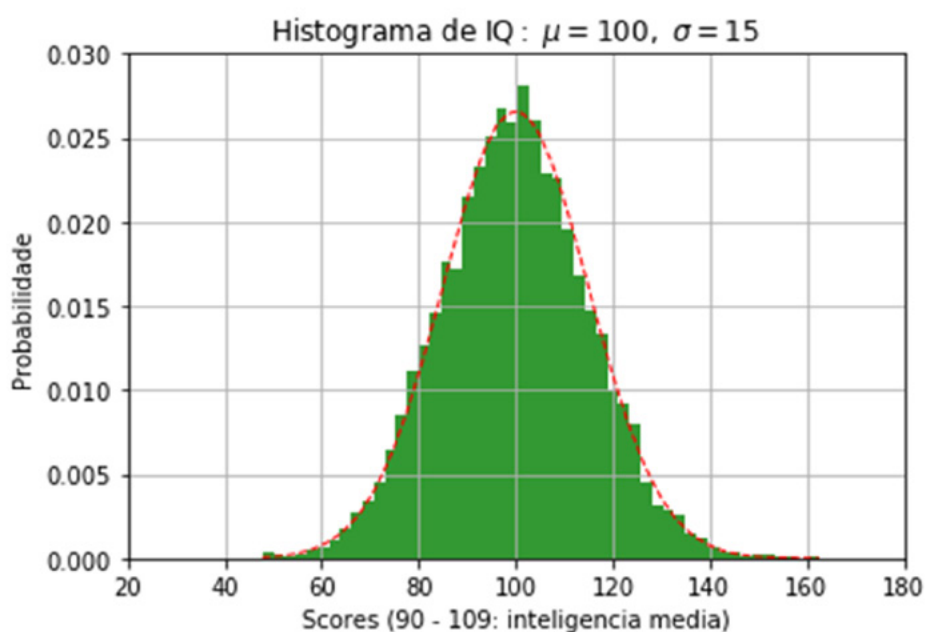
# histograma
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='green', alpha=0.8)

y = mlab.normpdf( bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=1)

plt.xlabel("Scores (90 - 109: inteligencia media)")
plt.ylabel("Probabilidade")
plt.title(r'$\mathrm{Histograma\ de\ IQ:}\ \mu=100,\ \sigma=15$')
plt.axis([20, 180, 0, 0.03])
plt.grid(True)
plt.show()
```

Saída:

Figura 3 – Exemplo de histograma de scores de teste de quociente de inteligência utilizando a biblioteca Matplotlib



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

1.2.3 Correlação

É uma análise descritiva que mede o grau de dependência entre duas variáveis (GRUS, 2019; BEN, 2017). Os valores de correlação variam entre -1 e 1. Existem dois componentes principais de um valor de correlação:

- **Magnitude** – Quanto maior a magnitude (mais próxima de 1 ou -1), mais forte a correlação.
 - **Sinal** – Se negativo, há uma correlação inversa. Se positivo, há uma correlação regular.
- a. **Correlação positiva** – Na biblioteca Numpy a função ***corrcoef*** retorna uma matriz de correlações de x com x, x com y, y com x e y com y (Figura 4). Estamos interessados nos valores da correlação de x com y (então posição (1, 0) ou (0, 1)).

Código-fonte adaptado (BEM, 2017)

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')

np.random.seed(1)

# Gerando 1000 numeros randomicamente entre 0 a 50
x = np.random.randint(0, 50, 1000)

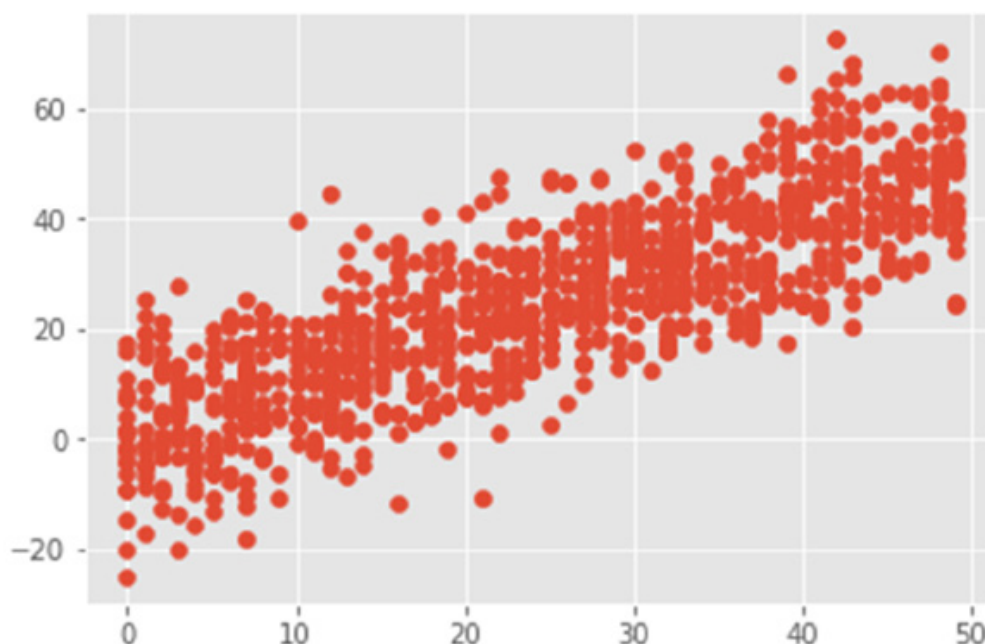
# Adicionando ruidos no conjunto de dados gerados
y = x + np.random.normal(0, 10, 1000)

print "A correlação entre (x e y) é : ", np.corrcoef(x, y)

plt.scatter(x, y)
plt.show()
```

Saída: A correlação entre (x e y) é: 0.81543901

Figura 4 – Exemplo de correlação positiva de um conjunto de dados gerados aleatoriamente



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

b. Correlação negativa – O que acontece com a correlação se a invertermos, de modo que um aumento em x resulte em uma diminuição em y? (Figura 5).

Código-fonte adaptado (BEM, 2017)

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

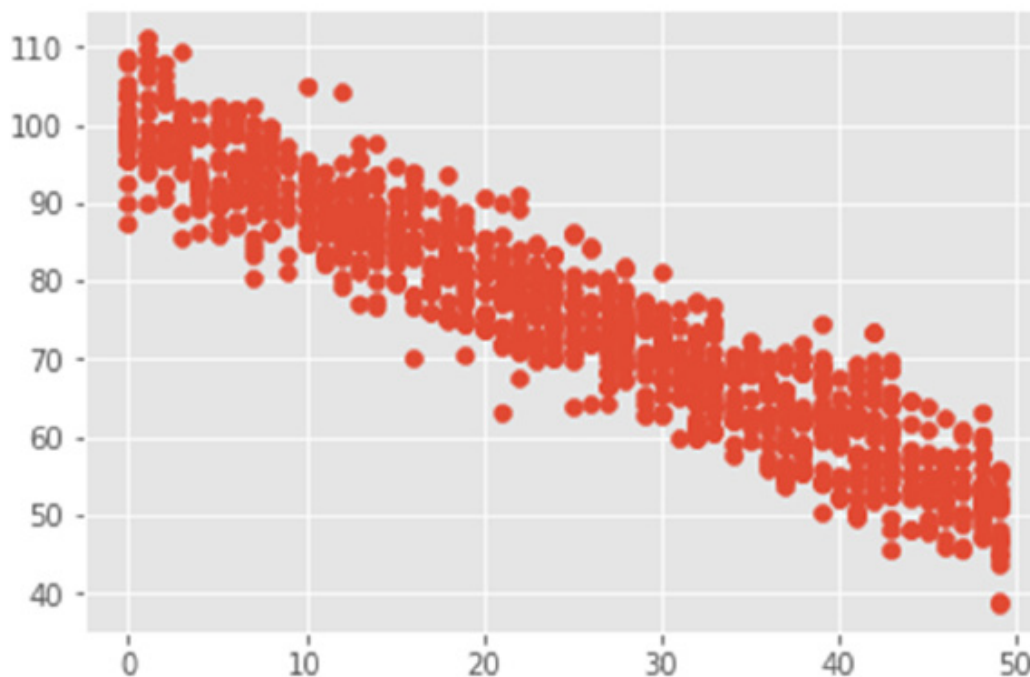
x = np.random.randint(0, 50, 1000)
y = 100 - x + np.random.normal(0, 5, 1000)

print "A correlação entre (x e y) é: ", np.corrcoef(x, y)

plt.scatter(x, y)
plt.show()
```

Saída: A correlação entre (x e y) é: -0.94957116

Figura 5 – Exemplo de correlação negativa de um conjunto de dados gerados aleatoriamente



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

- c. Sem correlação** – Em alguns casos, pode não existir correlação entre as variáveis ou conjunto de dados. Em alguns casos, o valor da correlação será 0 (zero) ou um valor muito próximo de zero (Figura 6).

Código-fonte adaptado (BEM, 2017)

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

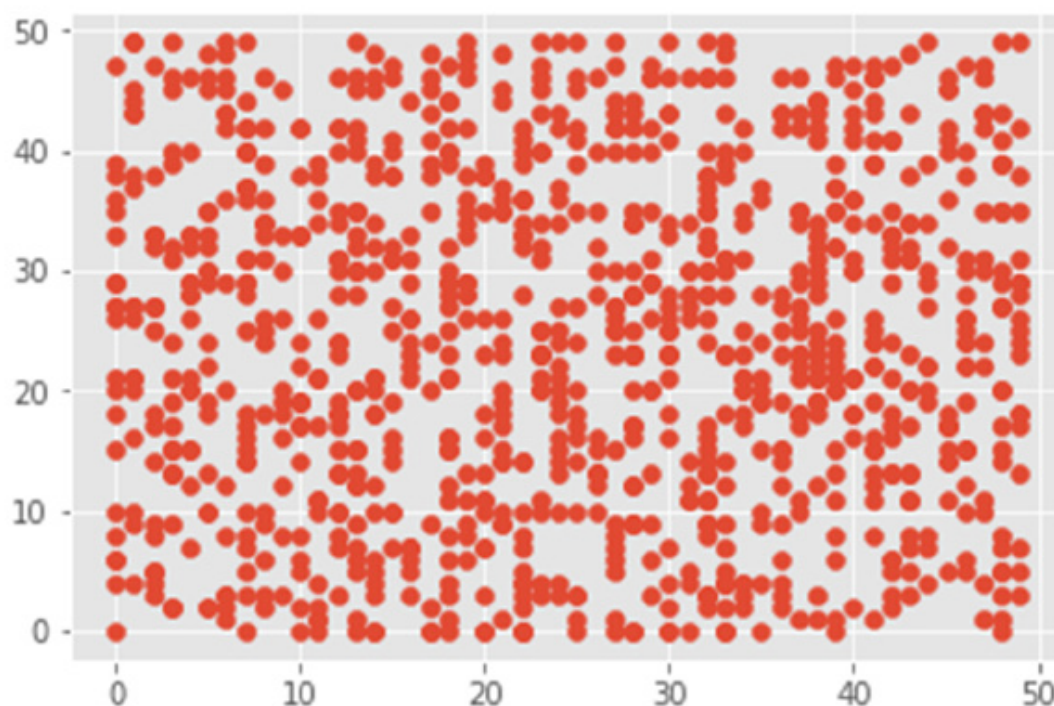
x = np.random.randint(0, 50, 1000)
y = 100 - x + np.random.normal(0, 5, 1000)

print "A correlação entre (x e y) é : ", np.corrcoef(x, y)

plt.scatter(x, y)
plt.show()
```

Saída: A correlação entre (x e y) é: 0.00404702

Figura 6 – Exemplo de correlação nula (sem correlação) de um conjunto de dados gerados aleatoriamente.



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.

d. Matriz de correlação – Utilizando a biblioteca Pandas, podemos criar uma matriz de correlação para visualizar as correlações entre diferentes variáveis em um quadro de dados (Figuras 8(a) e 8(b)):

Código-fonte

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame({'a': np.random.randint(0, 50, 1000)})
df['b'] = df['a'] + np.random.normal(0, 10, 1000) # positively correlated
df['c'] = 100 - df['a'] + np.random.normal(0, 5, 1000) # negatively correlated
df['d'] = np.random.randint(0, 50, 1000) # not correlated with 'a'

print df.corr()
```



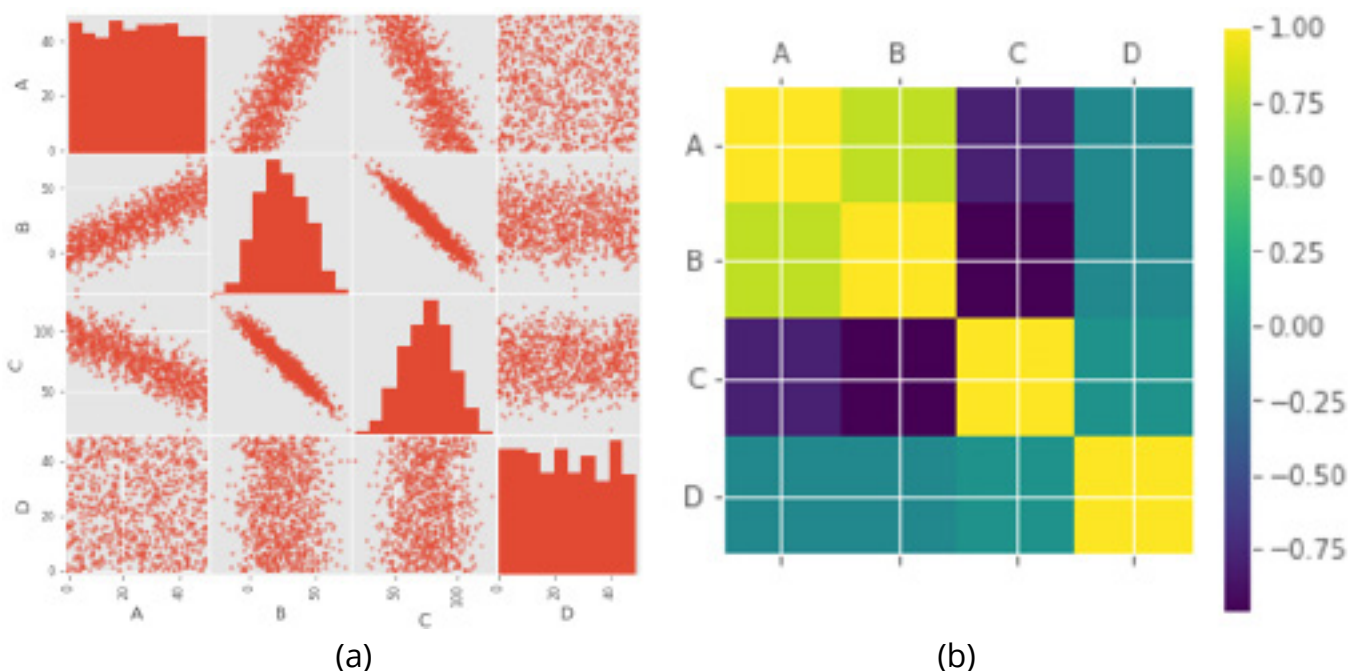
```
pd.scatter_matrix(df, figsize=(7, 7))
plt.show()

plt.matshow(df.corr())
plt.xticks(range(len(df.columns)), df.columns)
plt.yticks(range(len(df.columns)), df.columns)
plt.colorbar()
plt.show()
```

Saída:

	A	B	C	D
A	1.000000	0.816254	-0.947983	0.004675
B	0.816254	1.000000	-0.771339	0.032378
C	-0.947983	-0.771339	1.000000	0.010372
D	0.004675	0.032378	0.010372	1.000000

Figura 8 – Exemplos de representação gráfica de matriz de correlação desenvolvida a partir de 4 conjuntos de dados (A, B, C e D) gerados aleatoriamente



Fonte: elaborada pelo autor usando a interface do software Spyder 3.3.3.



ASSIMILE

Algumas bibliotecas do Python (NumPy, SciPy, Statistics, Pandas e Matplotlib) usadas para trabalhar com análise estatística dos dados precisam ser instaladas, pois elas não estão contidas no pacote Python. A melhor maneira de instalar essas bibliotecas é utilizando o comando ***pip install***. Exemplo: ***pip install <nome_biblioteca1, nome_biblioteca2, nome_biblioteca3>***



TEORIA EM PRÁTICA

Reflita sobre a seguinte situação: Uma empresa de venda *on-line* (*e-commerce*) percebeu que suas vendas tiveram queda em determinado período do ano. Diante desse cenário, a empresa precisa descobrir as causas e tomar as devidas providências. Outras informações importantes que a empresa precisa saber são: **Quais os meses em que as vendas tiveram queda, quais foram os meses mais produtivos, produtos mais vendidos e as informações dos consumidores como: idade, sexo e localização.** Utilizando os conceitos de estatística e probabilidades aprendidas nessa unidade, como você organizaria essas informações para a empresa?

VERIFICAÇÃO DE LEITURA



1. Estatística e probabilidade são extremamente importantes para trabalhar na área de ciência de dados, assim como em outras áreas como medicina, ciências biológicas, psicologia, engenharia, computação etc. Sobre estatística e probabilidade básica, assinale a alternativa correta:
 - a. A estatística é uma área da matemática que estuda as chances de ocorrência de um determinado experimento ou evento.
 - b. A probabilidade é a área responsável pela coleta, organização e interpretação de dados experimentais e pela extrapolação dos resultados da amostra para a população.
 - c. A média é o valor que separa a metade maior e a metade menor de uma amostra, uma população ou uma distribuição de probabilidade.
 - d. A moda é o elemento que ocorre com mais frequência em uma amostra, população ou distribuição.
 - e. A média de um conjunto é a soma de todos os valores contidos nele.
2. O _____ é uma poderosa biblioteca do Python usada principalmente para realizar cálculos em *arrays* e matrizes multidimensionais. Na biblioteca _____, existem duas estruturas de dados principais: Serie

e *DataFrames*. O _____ é a principal ferramenta de visualização bidimensional de gráficos científicos em Python. A API _____ é um módulo Python que fornece classes e funções para a estimativa de muitos modelos estatísticos, bem como para a realização de testes e a exploração de dados estatísticos.

Assinale a alternativa que completa adequadamente as lacunas.

- a. numpy; Pandas; Matplotlib; Statsmodels.
 - b. statsmodels; Pandas; Matplotlib; Numpy.
 - c. numpy; Matplotlib; Pandas; Statsmodels.
 - d. statsmodels; Matplotlib; Pandas; Numpy.
 - e. numpy; SciPy; Matplotlib; Statsmodels.
3. O histograma, também conhecido como distribuição de frequências, é a representação gráfica em colunas ou em barras de um conjunto de dados e dividido em classes uniformes ou não uniformes. Sobre histograma, assinale a alternativa INCORRETA:
- a. Utilizando algumas bibliotecas do Python como Numpy, Pandas e Matplotlib é possível criar e visualizar histogramas.
 - b. Um histograma também pode ser normalizado para mostrar frequências relativas.
 - c. Para a construção de histograma os dados devem estar ordenados em ordem crescente ou decrescente.

- d. O histograma de uma imagem digital indica o número de pixels que a imagem tem em determinado nível de cinza ou cor.
- e. No gráfico do histograma, o eixo horizontal (x) indica os valores dos intervalos de variável de interesse (intervalos de classe). Já, o eixo vertical (y) mostra os valores das alturas das barras (frequência)

Referências bibliográficas

BEN, K. **Correlation in Python**, 2017. Disponível em: <http://benalexkeen.com/correlation-in-python/>. Acesso em: 27 out. 2019.

CHRISTINE, D. Scale your data, not your process: Welcome to the blaze ecosystem. Bilbao, **EuroPython**, 2015. Disponível em: <https://ep2015.europython.eu/en/>. Acesso em: 28 out. 2019.

DOWNEY, A. B. **Think stats – Probability and Statistics for Programmers**. O'Reilly Media, Inc, 2011.

DUCHESNAY, E.; LÖFSTEDT, T.; FEKI Y. **Statistics and Machine Learning in Python**, Open Access, Release 0.3 beta, 2019.

GRUS, J. **Data science from scratch: First principles with Python**. O'Reilly Media, 2019.

JONES, E.; OLIPHANT, T., PETERSON, P. **SciPy: Open source scientific tools for Python**, 2001. Disponível em: <https://www.scipy.org/docs.html>. Acesso em: 28 out. 2019.

MCKINNEY, W. Pandas: a foundational Python library for data analysis and statistics. **Python for High Performance and Scientific Computing**, 2012.

SEABOLD, S; JOSEF P. Statsmodels: Econometric and statistical modeling with python. **Proceedings of the 9th Python in Science Conference**. 2010.

UNPINGCO, J. Python for probability, statistics, and machine learning. **Springer International Publishing**, 2016.

VAN D, W, S.; COLBERT, S. C., VAROQUAUX, G. The NumPy array: a structure for efficient numerical computation. **Computing in Science & Engineering**, 13, 22, 2011.

Gabarito

Questão 1 – Resposta: D

Resolução: A probabilidade é uma área da matemática que estuda as chances de ocorrência de experimentos e a estatística é a área responsável pela coleta, organização e interpretação de dados experimentais e pela extrapolação dos resultados da amostra para a população. Portanto, as opções (a) e (b) são falsas. A mediana (e não a média) é o valor que separa a metade maior e a metade menor de uma amostra, uma população ou uma distribuição de probabilidade. A média de um conjunto é a soma de todos os valores contidos nele, dividido pela quantidade de elementos do conjunto.

Questão 2 – Resposta: A

Resolução: O NumPy é a biblioteca do Python usada principalmente para realizar cálculos em *arrays* e matrizes multidimensionais. Na biblioteca Pandas existem duas estruturas de dados principais: *Serie* e *DataFrames*. O Matplotlib é a principal ferramenta de visualização bidimensional de gráficos científicos em Python. A API Statsmodels é um módulo Python que fornece classes e funções para a estimativa de muitos modelos estatísticos, bem como para a realização de testes e a exploração de dados estatísticos.

Questão 3 – Resposta: C

Resolução: Para a construção de histograma os dados não precisam estar ordenados, portanto essa opção está incorreta. As demais alternativas estão corretas.



***Machine Learning* em Python**

Autor: Danilo Rodrigues Pereira

Objetivos

- Introdução aos principais conceitos de Aprendizado de Máquina (*Machine Learning*).
- Aprendizado de Máquina: modelos supervisionados (*Supervised learning model*) e não supervisionados (*Unsupervised learning model*).
- Exemplos práticos utilizando a linguagem de programação Python.


► 1. Introdução

Aprendizado de máquina (*Machine Learning*, em inglês) é uma área muito importante da Inteligência Artificial. Ela teve início por volta de 1950, quando alguns métodos estatísticos foram descobertos. Atualmente *machine learning* é usado para resolver problemas em diversas áreas como engenharia, medicina, biologia e computação. Iniciar uma carreira nessa área de ciência de dados não é tão complicado quanto pode parecer. Alguns conhecimentos em matemática, estatística e computação são fundamentais. O objetivo dessa unidade é fornecer conhecimentos teóricos e práticos necessários para você iniciar seus estudos na área de aprendizado de máquina. Nesse capítulo, vamos utilizar a linguagem de programação Python para aprender os principais conceitos de aprendizado de máquina (*machine learning*).

► 2. Introdução ao aprendizado de máquina (*Machine Learning*)

O aprendizado de máquina (*Machine Learning*) é uma área da Inteligência Artificial (IA) que tem como objetivo fazer a extração de conhecimento a partir de um conjunto de dados. É uma área bem complexa que requer conhecimento em matemática e computação. Resumidamente, podemos dizer que aprendizado de máquina **é o conceito de ensinar máquinas** (computadores) a executar tarefas com ou sem supervisão, através de algoritmos de programação (MÜLLER, 2016; MARSLAND, 2014).

Nos últimos anos, os métodos de aprendizado de máquina estão cada vez mais comuns em nossa vida cotidiana. Sem que percebamos, muitos aplicativos e sites que utilizamos contêm algoritmos que usam conceitos de *machine learning*: auto recomendações de filmes, comida ou produtos, rádio online personalizado e sugestão de novos amigos



em nossas redes sociais. Grandes empresas como Facebook, Amazon e Netflix vêm utilizando aprendizado de máquinas em suas atividades (KWOK, 2013; BELL, 2010; CASTELLI, 2017).

Antes de começar a trabalhar com *machine learning*, você precisa saber que existem vários estágios do aprendizado de máquina:

- Seleção do conjunto de dados.
- Análise de dados.
- Desenvolvimento de algoritmo.
- Análise dos resultados gerados pelo algoritmo.

Os tipos mais bem-sucedidos de algoritmos de aprendizado de máquina são aqueles que automatizam processos de tomada de decisão generalizando a partir de modelos conhecidos. Nesta configuração, esse modelo é conhecido como **aprendizado supervisionado**, ou seja, o usuário fornece ao algoritmo entradas (rotuladas) e saídas desejadas, e o algoritmo encontra uma maneira de produzir a saída desejada. Em particular, o algoritmo é capaz de criar a saída para uma entrada nunca vista antes e sem a ajuda de um humano. Analise o exemplo: você foi contratado por uma empresa para desenvolver um algoritmo para identificação de e-mail do tipo *spam*. Utilizando técnicas de aprendizado de máquina, o seu algoritmo tem como uma entrada de dados (grande número de e-mails) juntamente com informações (atributos) que indiquem se um e-mail é *spam* ou não. Dado um novo e-mail, o algoritmo produzirá uma previsão (porcentagem de chance) se o novo e-mail é *spam* (MÜLLER, 2016).

No aprendizado de máquinas **não supervisionado**, apenas os dados de entrada são conhecidos e nenhum dado de saída conhecido é fornecido ao algoritmo. Nesse modelo, o algoritmo não tem informações/atributos que ele possa aprender para os resultados de saída.



PARA SABER MAIS

Exemplos de atividades **supervisionadas** de aprendizado de máquina incluem: identificação de placas de automóveis, a partir de imagens obtidas de um sistema de radar; identificação de tumores no cérebro, a partir de uma imagem de ressonância; detecção de fraudes nas transações com cartão de crédito.

Como exemplos de **atividades não supervisionadas** de aprendizado de máquina podemos citar: identificação dos hotéis mais recomendados em sites de viagens e identificação de usuários em sites de relacionamentos com preferências semelhantes.

Figura 1 – Visão geral dos modelos mais utilizados em *machine learning*



Fonte: elaborada pelo autor.

Figura 2 – Visão geral do processo de *machine learning*

Machine Learning



Fonte: adaptada de Rajesh (2017).

2.1 Entendendo o conjunto de dados (*dataset*)

A etapa mais importante no processo de aprendizado de máquina é **entender os dados que você irá utilizar** e como eles se relacionam com o problema que você deseja resolver. É necessário entender o que está acontecendo no seu conjunto de dados antes de começar construindo um modelo (supervisionado ou não-supervisionado). Cada algoritmo tem suas particularidades e você deve ser capaz de entender isso e escolher qual configuração do problema funciona melhor para um dos modelos.

Os algoritmos de aprendizado de máquina são apenas uma parte de um processo de construção para resolver um problema específico. Muitas pessoas passam muito tempo construindo complexas soluções de aprendizado de máquina, apenas para descobrir que eles não resolvem o problema certo. Então, analise seus dados cuidadosamente e verifique se as informações contidas nelas serão suficientes para o desenvolvimento do algoritmo de aprendizado de máquina.

2.2 Bibliotecas em Python para trabalhar com *machine learning*

A linguagem de programação **Python** é uma das mais utilizadas para o desenvolvimento de aplicações na área de ciência de dados. Python possui bibliotecas de funções para visualização de dados e gráficos, estatísticas, processamento de imagem e muito mais (Figura 3). Essas bibliotecas fornecem aos usuários uma grande variedade de funções desenvolvidas que vão facilitar no desenvolvimento dos algoritmos. Uma das vantagens do uso do Python em relação a outras linguagens de programação é a *capacidade* para interagir diretamente com o código, usando um terminal ou outras ferramentas como o Jupyter Notebook. Como uma linguagem de programação de uso geral, o Python também permite a criação de interfaces gráficas de usuário (GUIs) complexas e serviços da *web*, e para integração em sistemas existentes (MÜLLER, 2016; PEDREGOSA, 2011; HARRINGTON, 2012). A seguir, um breve resumo sobre essas bibliotecas:

- **scikit-learn** – É um projeto *open-source*, o que significa que ele é gratuito para uso e distribuição, e qualquer pessoa pode obter facilmente o código-fonte das funções. O **scikit-learn** contém uma grande variedade de eficientes ferramentas para mineração e análise de dados. Para obter mais informações acesse o site oficial do *scikit-learn* na seção *Documentation*.
- **NumPy** – É um pacote essencial para trabalhar com programação científica e processamento de imagens em Python. Ele permite que você trabalhe com vetores, arrays e matrizes de N dimensões, de forma simples é bem eficiente. Numpy contém um conjunto de funções matemáticas de alto nível como operações de álgebra linear, transformada geométrica, de Fourier, entre outras. Para obter informações sobre instalação e exemplos, acesse o site oficial do NumPy e acessando a seção *Getting Started*.
- **SciPy** – É uma coleção de funções para computação científica em Python. Ele fornece funções avançadas de álgebra linear,

otimização e estatística. O *scikit-learn* baseia-se na coleção de funções do SciPy para implementar seus algoritmos.

A biblioteca muito importante do SciPy é a ***scipy.sparse***.

Para mais informações, acesse o site oficial do SciPy: Seções: *Install, Getting Started e Documentation*.

- **matplotlib** – É a principal biblioteca de plotagem científica em Python. Ela fornece funções para fazer visualizações em forma de gráficos de diversos tipos e histogramas. A visualização correta dos dados vai fornecer informações importantes e ajudar na análise e entendimentos dos resultados. Para mais informações, acesse o site oficial o matplotlib e acesse as seções: *Installation, documentation, examples e tutorials*.

A seguir, um exemplo de código em Python (Quadro 1) que produz um gráfico no formato pizza (Gráfico 1), de uma pesquisa feita em 2018 das linguagens de programação mais usadas e recomendadas pelos cientistas de dados (BOB, 2019), utilizando a biblioteca matplotlib.

Quadro 1 – Código para geração de gráfico em formato de pizza

```
import matplotlib.pyplot as plt

labels = 'Python', 'R', 'SQL', 'Java', 'Matlab', 'Outras'
porcentagens = [54, 13, 8, 6, 2, 17]

fig1, ax1 = plt.subplots()

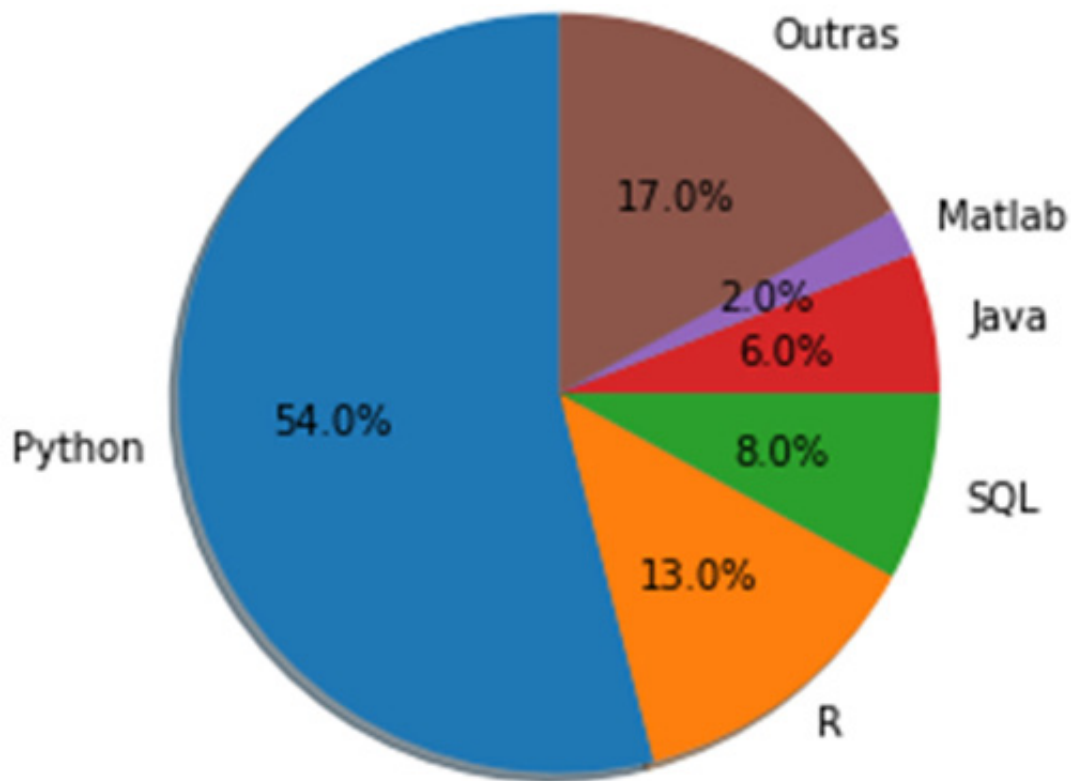
ax1.pie(porcentagens, labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)

ax1.axis('equal')

plt.show()
```

Fonte: adaptado de Bob (2019).

Gráfico 1 – Linguagens de programação mais usadas e recomendadas pelos cientistas de dados



Fonte: elaborado pelo autor.

- **pandas** – É uma biblioteca que fornece estruturas de dados de alto desempenho e fáceis de usar, e ferramentas de análise de dados para a linguagem de programação Python. Ela é construída em torno de estrutura de data chamada *DataFrame*, que é uma tabela semelhante a uma planilha do Excel. A biblioteca *pandas* fornece uma variedade de funções para modificar e operar nesta tabela; em particular, permite o uso de consultas e junções de tabelas. Outra ferramenta valiosa fornecida pelo *pandas* é sua capacidade de ingerir de uma grande variedade de formatos de arquivo e bancos de dados, como SQL, arquivos do Excel e CSV. Para mais informações sobre o Pandas, acesse o site oficial e visite a seção *Documentation*.

No quadro 2 é apresentado um exemplo de criação de um *DataFrame*, usando a estrutura de dados dicionário.

Quadro 2 – Criando um *DataFrame* no pandas

```
import pandas as pd

# Criando um conjunto de pessoas e informações sobre a sua idade e naturalidade
data = {'Naturalidade': ["São Paulo", "Belo Horizonte", "Bahia", "Sergipe"], 'Idade': [50, 17, 10, 35], 'Nome': ["João", "Pedro", "Maria", "Alex"]}

data_pandas = pd.DataFrame(data)

display(data_pandas)
```

Fonte: elaborado pelo autor.

Saída:

	Idade	Naturalidade	Nome
0	50	São Paulo	João
1	17	Belo Horizonte	Pedro
2	10	Bahia	Maria
3	35	Sergipe	Alex

Existem várias maneiras possíveis de consultar esta tabela. Por exemplo, se você quiser selecionar apenas pessoas maiores de 18 anos (Quadro 3).

Quadro 3 – Consultas em objetos *DataFrames* do pandas

```
import pandas as pd

# Criando um conjunto de pessoas e informações sobre a sua idade e naturalidade
data = {'Naturalidade': ["São Paulo", "Belo Horizonte", "Bahia", "Sergipe"], 'Idade': [50, 17, 10, 35], 'Nome': ["João", "Pedro", "Maria", "Alex"]}

data_pandas = pd.DataFrame(data)

display(data_pandas)

# Selecionando todas as linhas que tenha pessoas com idade > 18
display(data_pandas[data_pandas.Idade > 18])
```

Fonte: elaborado pelo autor.

Saída:

	Idade	Naturalidade	Nome
0	50	São Paulo	João
3	35	Sergipe	Alex

- **IPython** – O IPython fornece uma arquitetura rica para computação interativa como: um poderoso *shell* interativo; suporte para visualização interativa de dados; uso de kits de ferramentas da GUI; ferramentas fáceis de usar e de alto desempenho para computação paralela. Para obter informações sobre instalação e exemplos de uso, acesse o site oficial do IPython e visite as seções *Install* e *Documentation*.

Figura 3 – Bibliotecas do Python como suporte para criação de algoritmos de aprendizado de máquina (*machine learning*)



Fonte: Oleksii (2019).

2.3 Instalando a biblioteca *scikit-learn*

A biblioteca *scikit-learn* depende dos seguintes pacotes do Python:

- Python (≥ 3.5)
- NumPy ($\geq 1.11.0$)

- SciPy ($\geq 0.17.0$)
- joblib (≥ 0.11)
- matplotlib ($\geq 1.5.1$).
- scikit-image ($\geq 0.12.3$)
- pandas ($\geq 0.18.0$)

Todas as instruções detalhadas para instalação podem ser encontradas no site oficial do *scikit-learn* na seção *Documentation*.

Dica:

Algumas distribuições do Python já fornecem todos os pacotes necessários para trabalhar com *scikit-learn*, tais como Anaconda, Conapy e Python (x,y). Para fazer o download desses pacotes, acesse o site oficial do Python e acesse a seção *Download* → *Alternative Python Implementations*.

► 3. Aprendizado de máquina supervisionado

Os algoritmos de aprendizado de máquina supervisionados são modelos mais usados e bem-sucedidos. O aprendizado supervisionado é usado sempre que queremos prever um certo resultado de uma determinada entrada e temos exemplos de pares de entrada/saída. A aprendizagem supervisionada é a tarefa de encontrar um resultado a partir de conjunto de dados de treinamento já rotulados (MARSLAND, 2014).

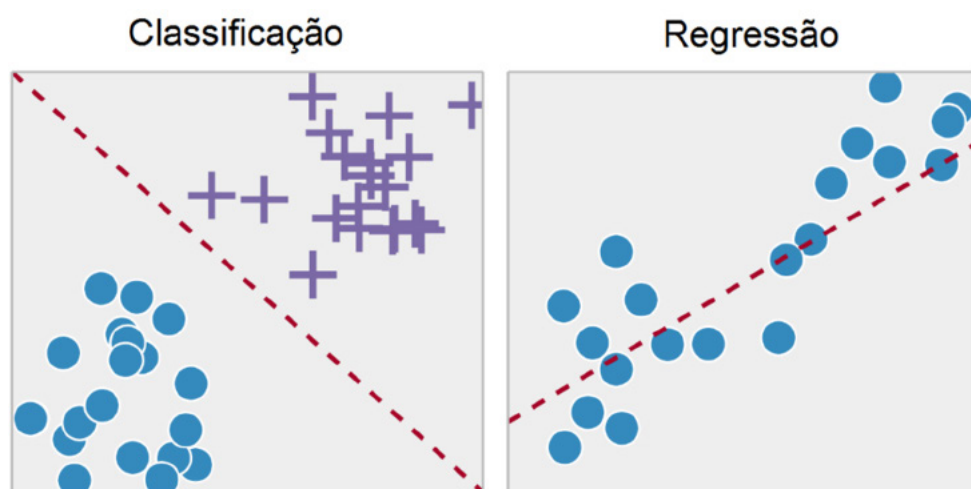
O principal objetivo é encontrar o conjunto de parâmetros ideais (atributos) que ajustem o modelo e que possa prever rótulos desconhecidos no conjunto de teste. Se o rótulo é um **número real**, a tarefa chama-se **regressão**. Se o rótulo vem de um **conjunto finito e não ordenado**, então a tarefa chama-se **classificação** (Figura 4).

3.1 Regressão linear e classificação

O principal objetivo da **regressão linear** é encontrar a linha reta que mais se aproxima do conjunto de dados fornecido. Em um problema de regressão linear, estamos tentando prever os resultados em uma saída contínua, o que significa que estamos tentando descobrir o melhor de conjunto de variáveis de entrada (atributos e características). Já na **classificação**, o objetivo é prever um rótulo de classe, que é uma escolha dentre uma lista de possibilidades. Em um problema de classificação, queremos prever os resultados em uma saída discreta. Em outras palavras, dado um conjunto de dados de entrada, o algoritmo terá que ser capaz de dividir os dados em várias categorias distintas. Uma maneira fácil de distinguir entre tarefas de classificação e regressão é perguntar se existe algum tipo de continuidade na saída. Se houver continuidade entre possíveis resultados, o problema é de **regressão**. Para ilustrar melhor, analise os seguintes exemplos:

- **Exemplo (regressão):** Dado um conjunto de dados contendo algumas informações de apartamento de alto padrão e sua localização no estado de São Paulo, o algoritmo de aprendizado de máquina tentará prever o preço do apartamento, com base nas informações de entrada (metragem, quantidade de cômodos e suítes) e localização.
- **Exemplo (classificação):** Uma empresa recebe milhares de e-mails por dia e, preocupada com a segurança de suas informações, desenvolve um algoritmo capaz de classificar esses e-mails em 2 grupos distintos: Não spam e spam. Desta forma, os e-mails classificados como spam, serão automaticamente removidos.

Figura 4 – Modelos utilizados no desenvolvimento de algoritmos de aprendizado de máquina supervisionado



Fonte: adaptada de Laura (2015).

3.2 Exemplo de regressão linear utilizando a biblioteca *scikit-learn*

Cenário:

Vamos desenvolver um algoritmo para tentar prever o preço de imóveis na cidade de Boston nos Estados Unidos (Quadro 4). Para esse exemplo, vamos utilizar uma base de dados gratuita do *scikit-learn* chamada ***Boston house prices dataset***. Todas as informações desse conjunto de dados, estão disponíveis no site **oficial do *scikit-learn***, na seção de base de dados.

Passo a passo:

1. Primeiramente, é necessário importar o conjunto de dados (***Boston house prices dataset***).
2. Após carregar os dados, devemos separá-los em dois conjuntos: **treinamento e teste**.
3. Em seguida, vamos escolher o *modelo*, em nosso caso vamos utilizar o modelo ***LinearRegression***.

4. Após selecionar o modelo, vamos fazer o treinamento nos dados de treino, utilizando uma função **fit()** da classe `LinearRegression`.
5. Selecionado o modelo, o algoritmo realizará previsões com base nas informações de entrada, retornando um *score* (valor entre 0 e 1). O *score* indica a porcentagem de variação nos preços dos imóveis.

Quadro 4 – Exemplo de algoritmo de regressão linear

```
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# carregando os dados
house_data = load_boston()

data = house_data['data']
target = house_data['target']

# separando o conjunto em 2 grupos: treinamento e teste
data_train, data_test, target_train, target_test = train_test_split(data, target, test_size=0.33,
random_state=42)

regr = LinearRegression()
regr.fit(data_train, target_train)

score_treinamento = regr.score(data_train, target_train)
score_teste = regr.score(data_test, target_test)

print("Tamanho do dataset: {}".format(house_data.data.shape[0]))

print('Variações nos preços no conjunto de treinamento são: ' + str(round(score_treinamento *
100, 2)) + '%')

print('Variações nos preços no conjunto de teste são: ' + str(round(score_teste * 100, 2)) + '%')
```

Fonte: elaborado pelo autor.

Analisando os resultados do algoritmo (Quadro 4), indicou que houve 73% da variação nos preços dos imóveis na cidade de Boston. O conjunto de dados contém apenas 506 observações, talvez teríamos um resultado melhor se o conjunto de dados (*dataset*) fosse maior e com mais atributos.

3.3 Exemplo de classificação utilizando a biblioteca *scikit-learn*

Cenário:

Nesse exemplo vamos desenvolver um classificador (Quadro 5) utilizando uma base de dados gratuita do *scikit-learn* que contém informações sobre tumores em câncer de mama. O objetivo desse classificador é descobrir quais tumores são malignos e benignos. Todas as informações dessa base de dados estão disponíveis no site oficial do *scikit-learn*: *Documentation* → *API* → *sklearn.datasets: Datasets* → *sklearn.datasets.load_breast_cancer*.

Passo a passo:

1. Primeiramente é necessário importar o conjunto de dados (***Breast Cancer Wisconsin Diagnostic Database***). O dataset inclui informações sobre tumores de câncer de mama com os rótulos de classificação como malignos ou benignos e sobre 30 atributos (características dos tumores).
2. Após carregar os dados, devemos organizá-los, separando-os em dois conjuntos: **treinamento (70%) e teste (30%)**.
3. Em seguida, vamos escolher o modelo; neste exemplo vamos utilizar o ***Naive Bayes (NB)***, bastante utilizado para tarefas de classificação binária.
4. Após selecionar o modelo, vamos iniciar a etapa de treinamento dos dados de treino, criado na etapa 2. Para isso, vamos utilizar a função ***fit()*** da classe *GaussianNB*.
5. Depois da etapa de treinamento do modelo, o algoritmo fará as previsões no nosso conjunto de teste, utilizando a função ***predict()***. Ele retornará uma matriz de previsões para cada instância de dados no conjunto de testes. A matriz contém valores 0 e 1 que representa nossos valores previstos para a classe tumor (maligno vs. benigno).

6. Finalmente, vamos analisar a precisão do classificador. Utilizaremos a função ***accuracy_score()*** do *scikit-learn* para determinar a precisão do nosso classificador.

Quadro 5 – Exemplo de algoritmo de classificação

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Carregando o dataset
dataset = load_breast_cancer()

# Organizar os dados
labels = dataset['target']
features = dataset['data']

# Separando os dados em 2 conjuntos: treinamento e teste
treinamento, teste, treinamento_labels, teste_labels = train_test_split(features,
labels, test_size=0.33, random_state=42)

gnb = GaussianNB()
model = gnb.fit(treinamento, treinamento_labels)
preds = gnb.predict(teste)

# Imprimindo o score do classificador
print('Score do classificador: ' + str(round(accuracy_score(teste_labels, preds)*100,2)) + '%')
```

Fonte: elaborado pelo autor.

Analisando o resultado do *score* do classificador ***Naive Bayes (NB)*** é 94%, isso indica que ele é capaz de fazer a previsão correta se o tumor é maligno ou benigno. Outra análise importante que podemos observar é que esses resultados indicam que nosso conjunto de 30 atributos (*features*) é um número considerado bom para classificação dessa classe do tumor.

► 4. Aprendizado de máquina não-supervisionado

O aprendizado **não-supervisionado** é um tipo de aprendizado de máquinas no qual não há saída conhecida. O algoritmo recebe como entrada, um conjunto de dados não rotulados. O algoritmo deverá ser capaz de extrair conhecimento (atributos e características) somente através dos dados de entrada (MÜLLER, 2016).

Muitas vezes os algoritmos não-supervisionados são usados em um cenário exploratório, quando um cientista de dados deseja entender melhor os dados, e não descobrir uma solução automática e eficaz. Para reduzir o consumo de memória e tempo dos computadores, uma etapa de pré-processamento ou transformações do conjunto de dados (dataset) é extremamente importante para os algoritmos não-supervisionados (MÜLLER, 2016).

O principal desafio no aprendizado não-supervisionado é avaliar se o algoritmo aprende algo que de fato irá ajudar na resolução do problema. Algoritmos de aprendizado não supervisionado são aplicados a dados que não contêm informações sobre rótulos, por isso não sabemos prever qual será a saída. Portanto, é muito difícil avaliar se o modelo teve um desempenho satisfatório ou não, vai depender do contexto do problema em questão (MÜLLER, 2016).

4.1 Agrupamento (*Clustering*)

O agrupamento (*clustering*, em inglês) é a tarefa de particionar o conjunto de dados (*dataset*) em grupos chamados **clusters**. Após o agrupamento, o cientista de dados analisa o resultado para entender as características fundamentais de agrupamento. Em seguida, é fornecido um *dataset* de teste, e o algoritmo é capaz de informar a qual agrupamento pertence cada elemento do conjunto de teste.

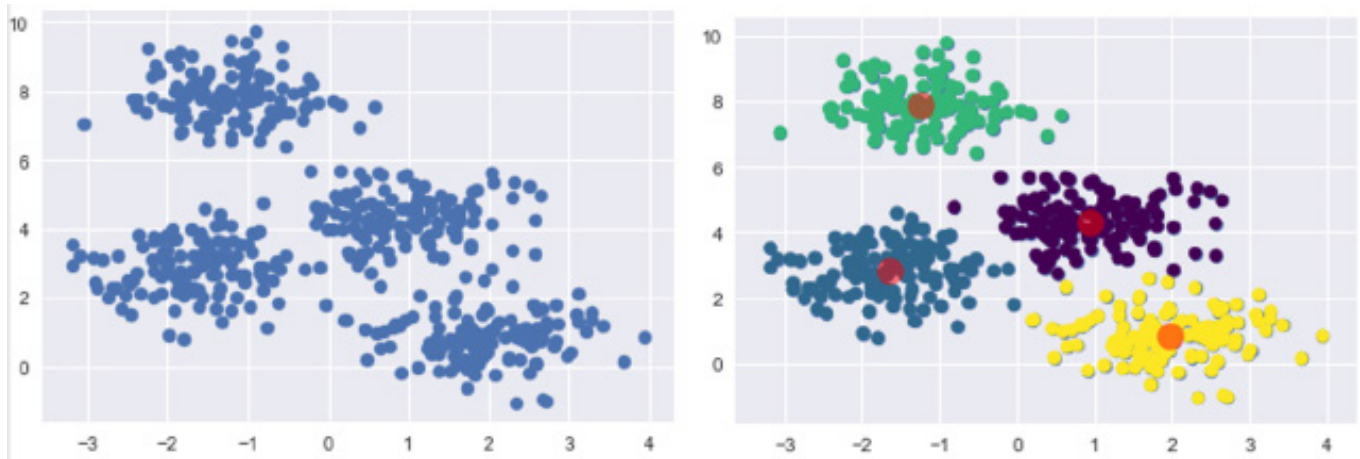
4.2 *k*-Means clustering

O agrupamento *k-means* é um dos algoritmos de agrupamento mais simples e mais popular de aprendizado não-supervisionado (Quadro 6). O algoritmo gera *k* agrupamentos a partir de um conjunto (*dataset*) de treinamento. A saída gerada é semelhante a diagrama de Voronoi, que é um particionamento do conjunto de dados com alguns pontos centrais (centroides). Os centroides são os dados que melhor representam o agrupamento e o “centro do *cluster*” é a média aritmética de todos os pontos pertencentes ao cluster (MÜLLER, 2016).

Exemplo prático:

1. Primeiro, vamos gerar um conjunto de dados bidimensional contendo quatro blobs distintos utilizando a função *make_blobs* da biblioteca *sklearn*. Podemos facilmente separar os dados gerados 4 grupos, porém o algoritmo *k-means* fará isso automaticamente (Figura 4).
2. Vamos agora visualizar os resultados plotando os dados coloridos por esses rótulos. Também serão marcados os centros de *cluster* (círculo vermelho), conforme determinado pelo estimador *k-means* (Figura 4).

Figura 4 – Conjunto de dados gerados de forma aleatória através da função *make_blobs* da biblioteca *sklearn* (esquerda) e conjunto de dados agrupados (*clusters*) gerado através da classe *KMeans* da biblioteca *sklearn* (direita)



Fonte: elaborada pelo autor.

Quadro 6 – Exemplo de algoritmo de agrupamento (*k-means*)

```
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans

X, Y = make_blobs(n_samples=500, centers=4, Vcluster_std=0.70, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);

kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.5);
```

Fonte: adaptado de Vanderplas (2016).



ASSIMILE

Aprendizado de máquina **supervisionado** é a tarefa de encontrar um resultado a partir de conjunto de dados de treinamento já rotulados (*classification and regression*). Já o aprendizado de máquina **não-supervisionado** apenas os dados de entrada são conhecidos e nenhum dado de saída conhecido é fornecido ao algoritmo (*clustering*).



TEORIA EM PRÁTICA

Você foi contratado por uma empresa produtora de soja no Brasil para resolver o seguinte problema: em um determinado período do ano, a empresa percebeu, através de imagens de satélite que a plantação está sendo prejudicada por pragas. Diante desse cenário, você deverá desenvolver um algoritmo de aprendizado de máquina que seja capaz de identificar as regiões mais afetadas pelas pragas e separar as imagens em grupos ou categorias, que serão os tipos de pragas encontradas, para que a empresa possa tomar as providências para resolver esse problema e evitar prejuízos futuros.

► 5. Considerações finais

Descreva todas as etapas necessárias para a resolução desse problema, considerando as seguintes questões:

- Quais bibliotecas do Python poderão ser utilizadas?
- Qual modelo de aprendizado de máquina (supervisionado ou não-supervisionado) é mais adequado para resolver o problema?

- c. Quais atributos e características serão necessários extrair do conjunto de imagens de satélites.

VERIFICAÇÃO DE LEITURA



1. O aprendizado de máquina (*Machine Learning*) é uma área da Inteligência Artificial que tem por objetivo fazer a extração de conhecimento partir de um conjunto de dado e os principais estágios (etapas) são:
 - a. Seleção do conjunto de dados, análise de dados, seleção do modelo (supervisionado ou não supervisionado), desenvolvimento do algoritmo e análise dos resultados.
 - b. Desenvolvimento do algoritmo, seleção do modelo (supervisionado ou não supervisionado), seleção do conjunto de dados e análise dos resultados gerados.
 - c. Classificação dos dados, desenvolvimento do algoritmo, seleção do modelo (supervisionado ou não supervisionado) e análise dos resultados.
 - d. Desenvolvimento do algoritmo, seleção do modelo (supervisionado ou não supervisionado), classificação do conjunto de dados e análise dos resultados gerados.
 - e. Desenvolvimento do algoritmo, seleção do modelo (supervisionado ou não supervisionado), agrupamento do conjunto de dados e análise dos resultados gerados.

2. Identificação de placas de automóveis a partir de imagens obtidas de um sistema de radar; identificação de tumores no cérebro, a partir de uma imagem de ressonância e detecção de fraudes nas transações com cartão de crédito. São exemplo de algoritmos:
- a. Aprendizado de máquina supervisionado por reforço.
 - b. Aprendizado de máquina supervisionado (regressão).
 - c. Aprendizado de máquina não-supervisionado (agrupamento).
 - d. Aprendizado de máquina não-supervisionado (agrupamento *k-means*).
 - e. Aprendizado de máquina supervisionado (classificação).
3. Analise as seguintes afirmações:
- I. O aprendizado de máquina supervisionado é usado sempre que queremos prever um certo resultado a partir de um conjunto de dados de treinamento não rotulados.
 - II. No aprendizado de máquina não-supervisionado, apenas os dados de entrada são conhecidos e nenhum dado de saída conhecido é fornecido ao algoritmo.
 - III. O agrupamento (*clustering*) é a tarefa de particionar o conjunto de dados (*dataset*) em grupos, chamados *clusters*.

- a. Todas afirmações são verdadeiras.
- b. As afirmações I e II são verdadeiras.
- c. Apenas a afirmação II é verdadeira.
- d. As afirmações II e III são verdadeiras.
- e. Todas as afirmações são falsas.

Referências bibliográficas

- BELL, R. M; KOREN, Y; VOLINSKY, C. All together now: A perspective on the netflix prize. Chance, v. 23, n. 1, p. 24-29, 2010. BOB, H., **Programming Languages Most Used and Recommended by Data Scientists**, Analytics, Data Science, Machine Learning, 2019.
- CASTELLI, M; MANZONI, L; VANNESCHI, L; POPOVIČ, A. **An expert system for extracting knowledge from customers' reviews: The case of Amazon**. Com. Inc. Expert Systems with Applications, 2017.
- HARRINGTON, P. Machine learning in action. Manning Publications Co., 2012.
- KWOK, L and BEI Y. **Spreading social media messages on Facebook: An analysis of restaurant business-to-consumer communications**. Cornell Hospitality Quarterly, 2013.
- LAURA, E. **Is machine learning the new EPM black?** Musings on ML, Deep learning & AI, 2015.
- MATHEUS, F. **Aplicando Regressão Linear com Scikit-Learn**, 2017.
- MARSLAND, S. **Machine learning: an algorithmic perspective**. Chapman and Hall/ CRC, 2014.
- MICHELLE M. How To Build a Machine Learning Classifier in Python with Scikit-learn. **The Digital Ocean Community**, 2017.
- MÜLLER, A. C.; SARAH, G. Introduction to machine learning with Python: a guide for data scientists. **O'Reilly Media**, Inc., 2016.
- OLEKSII, K. Beginner's Guide to Machine Learning with Python. **Towards Data Science**, 2019.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; VANDERPLAS, J. Scikit-learn: Machine learning in Python. **Journal of machine learning research**, 2011.

RAJESH, K. Introduction to Machine Learning. **Towards Data Science**, 2017.

VANDERPLAS, J. Python Data Science Handbook: essential tools for working with data. **O'Reilly Media**, Inc., 2016.

Gabarito

Questão 1 – Resposta: A

Resolução: As principais etapas no processo de *machine learning* são: seleção do conjunto de dados, análise de dados, seleção do modelo (supervisionado ou não supervisionado), desenvolvimento do algoritmo e análise dos resultados

Questão 2 – Resposta: E

Resolução: Identificação de placas de automóveis a partir de imagens obtidas de um sistema de radar; identificação de tumores no cérebro, a partir de uma imagem de ressonância e detecção de fraudes nas transações com cartão de crédito são exemplos que podem ser resolvidos utilizando classificadores.

Questão 3 – Resposta: D

Resolução: A afirmação I é falsa, pois o aprendizado de máquina supervisionada é a tarefa de encontrar um resultado a partir de conjunto de dados de treinamento rotulado.



Processando *Big Data* com Apache Spark

Autor: Danilo Rodrigues Pereira

► Objetivos

Big data é um termo bastante popular na área de ciência de dados e descreve o grande volume de dados. A quantidade de dados criados e armazenados globalmente continua crescendo a cada ano. Essa crescente criação de dados pelas mídias sociais, aplicativos de negócios e telecomunicações e vários outros domínios está levando à formação de *Big Data* (FOSTER, 2019; CHAMBERS, 2018; KARAU, 2015). Entretanto, não é a quantidade de dados disponíveis que importa, mas sim, tem relevância o que vamos fazer com eles para conseguir extrair informações importantes (conhecimentos). O objetivo dessa unidade é fornecer conhecimentos teóricos e práticos necessários para você analisar e processar de maneira simples e eficiente grande volume de dados. No final da unidade você será capaz de:

- Analisar e processar grandes volumes de dados (*Big Data*) com o *framework* Apache Spark.
- Utilizar as principais bibliotecas do Apache Spark para realizar operações e consultas em *Big Data*.
- Criar algoritmos para manipulação de arquivos utilizando a linguagem de programação Python e Apache Spark.

► 1. Introdução ao Apache Spark e *Big Data*

O Apache Spark é um *framework* para processamento de grande volume de dados (*Big Data*) e tem como principais características: velocidade no processamento de grande volume de dados, suporte para diversos formatos de dados, facilidade de uso, e suporte para diversos tipos de linguagem de programação como Python, Java, R e Scala.

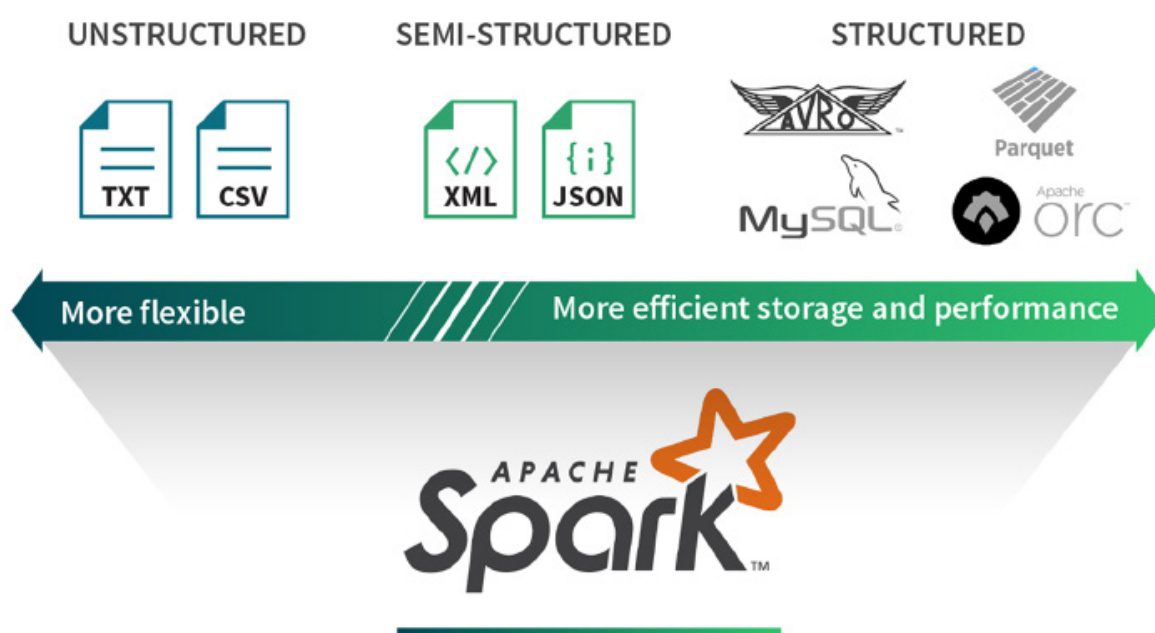
Antes de iniciarmos a análise e processamento de *Big Data* usando o Apache Spark, é importante que você entenda o que significa *Big Data*, quais são suas principais características e também como os dados são criados e armazenados. Resumidamente, podemos dizer que dados é o conjunto de informações (textos, imagens e vídeos) que podem ser armazenados ou utilizados por um computador. Aplicativos de negócios, telecomunicações, internet das coisas e mídias sociais geram uma quantidade enorme desses dados diariamente. Esses dados são criados e armazenados em vários formatos, que podemos classificar em três grupos (Figura 1):

- **Não-estruturado:** São geralmente textos de forma livre ou objetos binários que não contêm marcação ou metadados. Artigos de jornais, imagens e logs de aplicativos geralmente são tratados como **dados não-estruturados**. Esses tipos de dados geralmente exigem que o contexto em torno dos dados seja analisável. A desvantagem desses tipos de dados está no processo de extração de valores desses fontes de dados, pois são necessárias muitas transformações e técnicas de mineração e filtragem para interpretar esses conjuntos de informações.
- **Semiestruturado:** São dados salvos em estruturas de dados chamada registros, mas não necessariamente possuem um esquema global bem definido, por exemplo: JSON e XML. Os benefícios dos formatos de dados semiestruturados são que eles fornecem maior flexibilidade para expressar seus dados, pois cada registro é auto-descritivo. Esses formatos são muito comuns em

muitos aplicativos, e a maioria das linguagens de programação *contêm* bibliotecas específicas para ler e escrever diferentes formatos de dados semiestruturados.

- **Estruturado:** Os dados estruturados geralmente estão armazenados em bancos de dados relacionais (*RDBMS – Relational Database Management Systems*). Nesse esquema, os dados estão bem organizados para facilitar a consulta e extração de informação. Por esse motivo, os dados estruturados trazem benefícios ao lidar com grandes volumes de informações, pois uma vez que os dados estão organizados, o custo em termos de velocidade de acesso é menor.

Figura 1 – Tipos de dados suportados pelo Apache Spark



Fonte: Burak (2017).

O *Big Data* possui características e propriedades específicas que não apenas vão o ajudar a decifrá-lo, mas também fornecem uma visão de como e quais ferramentas podemos usar para análise e processamento dos dados, de forma rápida e eficiente (FIRICAN, 2017; SHAFER, 2017) e principalmente extrair informações importantes (conhecimentos) desses dados (Figura 2). Dentre as principais características podemos citar (Figura 3):

- **Volume (*Volume*):** É a principal característica de um *Big Data*. A quantidade atual de dados pode realmente ser bastante impressionante e isso requer boas práticas de processamento e extração das informações (conhecimentos) desses dados.
- **Velocidade (*Velocity*):** Refere-se à velocidade na qual os dados estão sendo criados, armazenados e atualizados.
- **Variedade (*Variety*):** Refere-se aos diferentes tipos de dados encontrados no *Big Data* (não-estruturados, semiestruturados e estruturados). A maioria dos dados em *Big Data* são *não-estruturados* (*áudio, imagem, vídeo, atualizações de mídia social, arquivos de log, dados de máquinas e sensores etc.*).
- **Veracidade (*Veracity*):** Essa é uma característica “ruim” do *Big Data*. A veracidade refere-se mais à proveniência ou confiabilidade da fonte de dados, seu contexto e a importância da análise com base nela.
- **Valor (*Value*):** Refere-se ao processo de extração de informações “valiosas” dos dados. Essas informações podem ajudar no processo de otimização de atividades e tomadas de decisões nos negócios das empresas.
- **Validade (*Validity*):** Semelhante à veracidade, validade refere-se à precisão e correção dos dados para o uso pretendido. Segundo Foster (2019), estima-se que 60% do tempo de um cientista de dados é gasto “limpando” seus dados antes de poder fazer qualquer análise.
- **Variabilidade (*Variability*):** A variabilidade no contexto do *Big Data* refere-se a inconsistências nos dados. Elas precisam ser encontradas por métodos de detecção de anomalias e *outliers* (valor que foge da normalidade) para que uma análise significativa ocorra.

- **Distribuição (*Vanue*):** Essa característica indica a forma de distribuição dos dados (plataformas, sistemas e aplicativos).
- **Vocabulário (*Vocabulary*):** Refere-se a esquema, modelos de dados, semântica, ontologias, taxonomias e outros metadados baseados em contexto e conteúdo que descrevem a estrutura, sintaxe, conteúdo e proveniência dos dados.
- **Indecisão (*Vagueness*):** Significa confusão sobre o conteúdo dos dados e qual a melhor maneira de analisá-los.

Figura 2 – Visão geral do processamento de *Big Data*



Fonte: Enis Aksoy/iStock.com.

Figura 3 – Principais características e propriedades de um *Big Data*

THE 10 Vs OF BIG DATA



Fonte: Pamela (2019).

1.1 Visão geral da API do Spark

O Apache Spark é uma plataforma de computação em *cluster* projetada para trabalhar com grande volume de dados (*Big Data*) de forma simples e eficiente (KARAU, 2015). O projeto Spark foi desenvolvido na linguagem Scala e executa em uma máquina virtual Java (*JVM – Java Virtual Machine*). A versão 2.4.4 tem suporte para as seguintes linguagens de programação: Python, R, Scala e Java (CHAMBERS, 2018; KARAU, 2015).

Além da API principal do Apache Spark, existem diversas bibliotecas adicionais para processamento de dados, SQL, grafos e aprendizado de máquina (*machine learning*). Essas bibliotecas são (Figura 4):

- **Spark SQL** – É a biblioteca mais importante do *framework* Apache Spark. Através dela você pode executar consultas SQL nativas em dados estruturados ou semiestruturados. Tem suporte para linguagem em Java, Scala, Python e R.

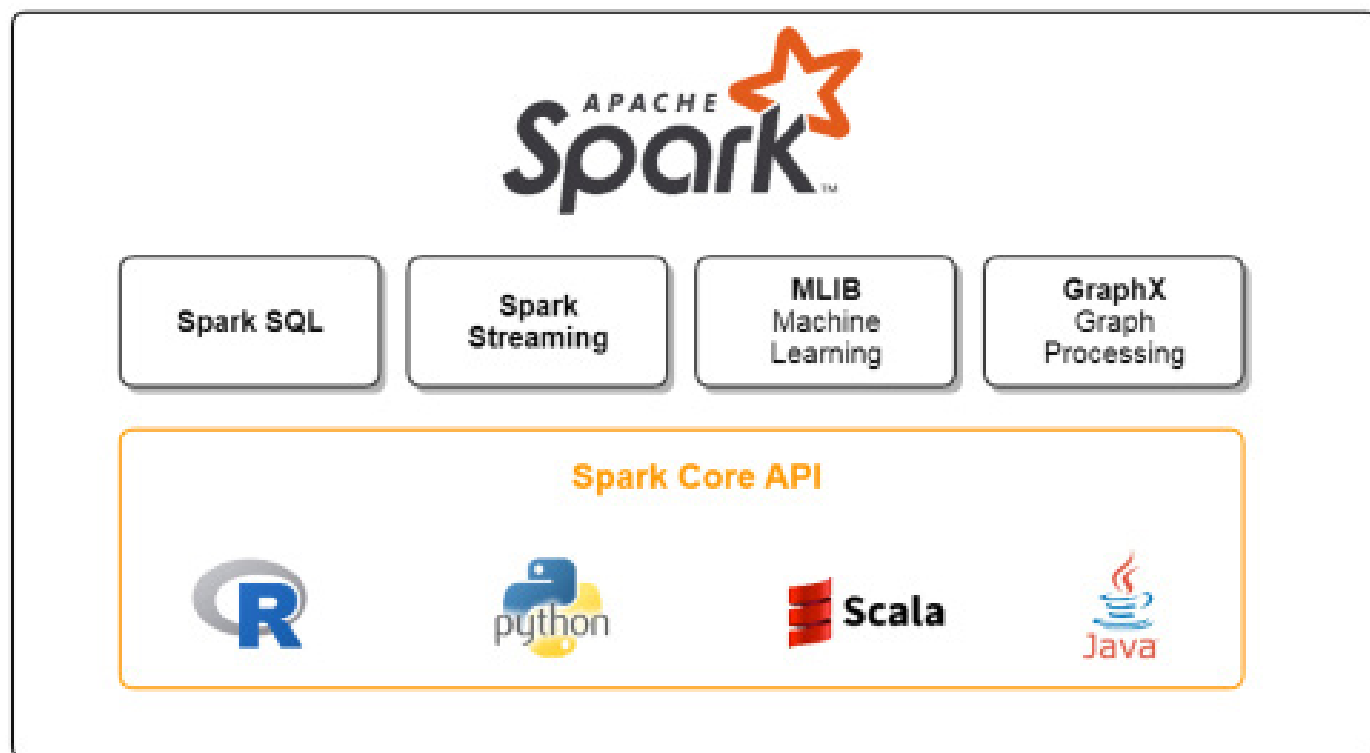
- **Spark Streaming** – É uma biblioteca usada para processar dados de *streaming* em tempo real. Dessa forma, podemos desenvolver algoritmos para processamento de dados à medida em que os dados chegam (em tempo real) e não em um processo em lote. O processamento e a análise de dados em tempo real estão se tornando um componente importante na estratégia de *Big Data* para a maioria das empresas e negócios (EITI, 2019).
- **Spark MLlib** – É uma biblioteca de aprendizado de máquina (*machine learning*), que consiste em diversos algoritmos de aprendizagem de máquina supervisionado e não-supervisionado, incluindo classificação, regressão e agrupamento (*clustering*).
- **Spark GraphX** – É uma API do Spark para trabalhar com grafos e computação paralela. O GraphX contém uma biblioteca de algoritmos para simplificar tarefas de análise de grafos.

PARA SABER MAIS



O Spark foi desenvolvido em 2009 pelo grupo de pesquisa do AMPLab da Universidade de Califórnia, Berkeley. No ano seguinte, foi publicado um manuscrito intitulado *Spark: Cluster Computing with Working Sets* e seu código-fonte foi distribuído gratuitamente (*open-source*) e gerenciado pela empresa Apache, dando origem ao nome Apache Spark (CHAMBERS, 2018).

Figura 4 – Visão geral da arquitetura do Apache Spark



Fonte: elaborada pelo autor.

1.2 Processamento de dados no Apache Spark utilizando Python

As bibliotecas (APIs) do *framework* Apache Spark possibilitam o desenvolvimento de aplicações usando algumas linguagens de programação bastante utilizadas da área de ciência dos dados, como Scala, Java, Python, SQL e R.

1.3 Conjuntos

O Apache Spark contém duas estruturas de dados para trabalharmos com coleções distribuídas: **DataFrame e DataSet**. Elas são estruturas de dados que armazenam os dados em forma de **tabela com linhas e colunas**. Alguns recursos e características do *DataFrame*, que podemos citar:

- Capacidade de processar os dados no tamanho de Kilobytes para Petabytes em um *cluster* de nó único.
- Suporta diferentes formatos de dados (Elasticsearch e Cassandra) e sistemas de armazenamento (HDFS, MySQL etc.).
- Pode ser facilmente integrado a todas as ferramentas e estruturas de *Big Data* via Spark-Core.
- Fornece bibliotecas de funções para Python, Java, Scala e R.

PARA SABER MAIS



A *DataFrame* API contém funções para o desenvolvimento de aplicativos que utilizam grandes volumes de dados (*Big Data*). Ela foi inspirada em outras APIs similares existentes nas linguagens R e Python (pandas).

1.3.1 Criando um *DataFrame*

Primeiramente, precisamos importar as classes **SparkSession** e **SparkContext** do pacote **pyspark** do Python. Nesse exemplo, vamos criar o *DataFrame*, a partir de dados armazenados num arquivo CVS (*voos_setembro_2019.csv*), que contém informações sobre número de voos internacionais realizados no mês de setembro de 2019.

- ***voos_setembro_2019.csv***

EMPRESA, PAIS_ORIGEM, PAIS_DESTINO, QTDE_VOO
Latam, Brasil, EUA, 3000
KLM, Brasil, Itália, 500
Gol, Brasil, Irlanda, 700
KLM, Brasil, Londres, 2500
Azul, Brasil, Portugal, 100

- **voos_setembro_2019.json**

```
{“EMPRESA”:“Latam”,“PAIS_ORIGEM”:“Brasil”,“PAIS_DESTINO”:  
“EUA”, “QTDE_VOO”:“3000”}  
{“EMPRESA”:“KLM”,“PAIS_ORIGEM”:“Brasil”,“PAIS_DESTINO”:“Itália”,  
“QTDE_VOO”:“500”}  
{“EMPRESA”:“Gol”,“PAIS_ORIGEM”:“Brasil”,“PAIS_DESTINO”:“Irlanda”,  
“QTDE_VOO”:“7000”}  
{“EMPRESA”:“KLM”,“PAIS_ORIGEM”:“Brasil”,“PAIS_  
DESTINO”:“Londres”, “QTDE_VOO”:“2500”}  
{“EMPRESA”:“Azul”,“PAIS_ORIGEM”:“Brasil”,“PAIS_  
DESTINO”:“Portugal”, “QTDE_VOO”:“100”}
```

Código-fonte

```
import pyspark  
from pyspark.context import SparkContext  
from pyspark.sql.session import SparkSession  
  
sc = SparkContext.getOrCreate()  
spark = SparkSession(sc)  
  
df_csv = spark.read.option(“inferSchema”, “true”).option(“header”, “true”).csv(“<caminho_do_  
arquivo_csv>”)  
  
print df_csv.show()
```

```
import pyspark  
from pyspark.context import SparkContext  
from pyspark.sql.session import SparkSession  
  
sc = SparkContext.getOrCreate()  
spark = SparkSession(sc)  
  
df_json = spark.read.format(“json”).load(“<caminho_do_arquivo_json>”)  
  
print df_json.show()
```

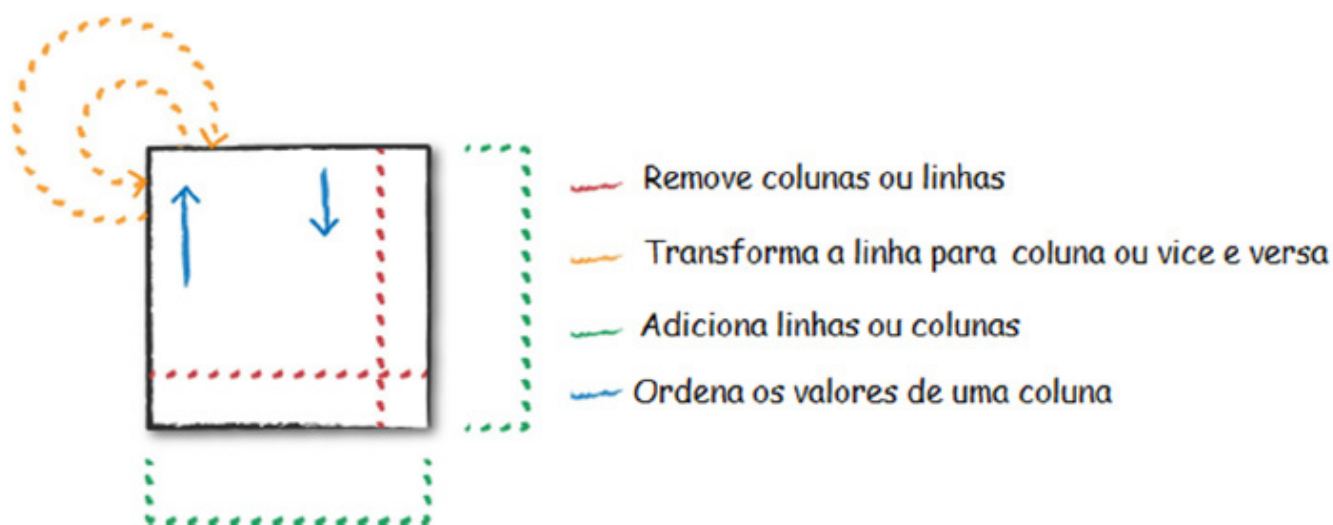
Saída:

EMPRESA	PAIS_DESTINO	PAIS_ORIGEM	QTDE_VOO
Latam	EUA	Brasil	3000
KLM	Itália	Brasil	500
Gol	Irlanda	Brasil	7000
KLM	Londres	Brasil	2500
Azul	Portugal	Brasil	100

PARA SABER MAIS

A *DataFrame* API contém outras funções interessantes, como, por exemplo: selecionar os valores de uma determinada coluna e transformações (adicionar ou remover linha e colunas; transformar colunas para linhas e vice e versa; ordenar valores de uma determinada coluna) (Figura 5).

Figura 5 – Diferentes tipos de transformações possíveis no *DataFrame*



Fonte: adaptada de Chambers (2018, p. 72).

1.3.2 Spark SQL

Através da biblioteca Spark SQL você também pode executar consultas SQL através do método `sql` da classe `SparkSession`. O método `SQL` retorna um objeto *DataFrame*. Utilizando os dados do arquivo `voos_setembro_2019.json`, da seção *Criando um DataFrame*, vamos desenvolver um algoritmo para salvar os dados numa visão (*view*) temporária.

Código-fonte

```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)

df = spark.read.json("<caminho_do_arquivo_json>")

# Criando um view temporária usando o DataFrame
df.createOrReplaceTempView("nome_view")

# Visualizando os dados da tabela
sc.sql("select name from <nome_view>").show()
```

1.3.2.1 Criando tabelas a partir de arquivo CSV e JSON

Para fazer algo útil com o Spark SQL, você precisa primeiro definir tabelas. As tabelas são equivalentes a um *DataFrame*, pois eles são uma estrutura de dados na qual você executa comandos. A principal diferença entre tabelas e *DataFrame* é a seguinte: você define *DataFrame* no escopo de uma linguagem de programação, enquanto você define tabelas em uma base de dados.

Utilizando a biblioteca Spark SQL, você consegue criar uma tabela com os valores a partir de um arquivo CSV ou JSON. Veja os exemplos:

Código-fonte

```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.sql import SQLContext

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)
sqlContext = SQLContext(sc)

df = spark.read.json("<caminho_do_arquivo_json>")

# Criando um view temporária usando o DataFrame
df.createOrReplaceTempView("<nome_tabela>")
# Visualizando os dados da tabela
sc.sql("select name from <nome_tabela>").show()
```

```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.sql import SQLContext

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)
sqlContext = SQLContext(sc)

df_csv = spark.read.option("inferSchema", "true").option("header", "true").
csv("<caminho_do_arquivo_csv>")

# Criando um view temporária usando o DataFrame
df_csv.createOrReplaceTempView("nome_tabela")

# Visualizando os dados da tabela
sqlContext.sql("select * from <nome_tabela>").show()
```

1.3.3 SQL Databases

SQL datasources são um dos conectores mais poderosos, porque há uma variedade de banco de dados à qual você pode se conectar (MySQL, PostgreSQL, Oracle, SQLite, entre outros). A seguir, um exemplo de código-fonte para fazer uma conexão com o banco de dados SQLite e PostgreSQL.

Código-fonte

```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.sql import SQLContext

driver = "org.sqlite.JDBC"

path = "<caminho_arquivo.db>"

url = "jdbc:sqlite:" + path

tablename = "nome_tabela"

# Exemplo de conexão (SQLite)

sqliteDF = spark.read.format("jdbc").option("url", url).option("dbtable", tablename).
option("driver", driver).load()

# Exemplo de conexão (Postgres)

postgresDF = spark.read.format("jdbc").option("driver", "org.postgresql.Driver").option("url",
"jdbc:postgresql://database_server").option("dbtable", "schema.tablename").option("user",
"usuario").option("password", "senha").load()

#Visualizando os dados da tabela
print sqliteDF.show()
print postgresDF.show()
```



ASSIMILE

Embora o SQLite seja um banco que pode ser usado para diversas aplicações, provavelmente não é recomendado que seja o banco de dados de produção. Além disso, o SQLite não funcionará necessariamente bem em uma configuração distribuída por causa de seu requisito para bloquear todo o banco de dados no momento da gravação ou atualização de uma tabela.



TEORIA EM PRÁTICA

Big Data é um termo bastante popular na área de ciência de dados e descreve o grande volume de dados. A quantidade de dados criados e armazenados globalmente continua crescendo a cada ano. Esses dados são classificados em 3 grupos: não-estruturado (*logs* de servidores e aplicativos, imagens e vídeos de câmera de segurança), semiestruturado (XML, CSV e JSON) e estruturado (banco de dados). Utilizando o *framework* Apache Spark, como você processaria esses dados, de forma a extrair informações importantes para a empresa?



VERIFICAÇÃO DE LEITURA

1. O Apache Spark é um *framework* para processamento de *Big Data* e tem como principal característica a eficiência no processamento dos dados.

Sobre as características do Apache Spark, assinale a alternativa correta:

- a. Velocidade no processamento dos dados, suporte para diversos formatos de dados (não-estruturados, semiestruturado e estruturado), facilidade de uso, e suporte para diversos tipos de linguagem de programação como Python, Java, R e MATLAB.
- b. Velocidade no processamento dos dados, suporte apenas para dados estruturados, facilidade de uso e suporte para diversos tipos de linguagem de programação como Python, Java, R e Scala.
- c. Velocidade no processamento dos dados, suporte para diversos formatos de dados (não-estruturados, semiestruturado e estruturado), facilidade de uso, e suporte para diversos tipos de linguagem de programação como Python, Java, R e Scala.
- d. Velocidade no processamento, suporte para diversos formatos de dados (semiestruturado e estruturado) e suporte para diversos tipos de linguagem de programação como Python, Java, R, MATLAB e Scala.
- e. Suporta apenas dados semiestruturados (CVS, JSON e XML) e sua API suporta as seguintes linguagens de programação: Python, Java, R e Scala.

2. Sobre o Apache Spark, analise as seguintes afirmações:

- I. A arquitetura do Apache Spark é formada pelas seguintes bibliotecas: Spark SQL, Spark *Streaming* e Spark MLlib.

II. Apache Spark executa em uma máquina virtual Java (JVM – Java Virtual Machine) e a instalação do JDK é opcional.

III. Fornece biblioteca de funções apenas para Python e Scala.

a. Todas as afirmações são verdadeiras.

b. Apenas a afirmações II é verdadeira.

c. As afirmações I e II são verdadeiras.

d. As afirmações II e III são verdadeiras.

e. Todas as afirmações são falsas.

3. *Big Data* possui diversas características que nos fornecem uma visão de como e quais ferramentas podemos usar para extrair conhecimento dos dados, de forma rápida e eficiente. Qual característica refere-se a inconsistências nos dados?

a. Variabilidade.

b. Variedade.

c. Veracidade.

d. Volume.

e. Validade.

Referências bibliográficas

BURAK, Y; MICHAEL, A; TATHAGATA, D; TYSON C. Working with Complex Data Formats with Structured Streaming in Apache Spark 2.1. **Engineering Blog**, 2017.

CHAMBERS, B.; ZAHARIA, M. **Spark: The Definitive Guide: Big Data Processing Made Simple**. San Francisco: O'Reilly Media, 2018.

EITI K. Processamento de Dados em "Tempo Real" com Apache Spark Structured Streaming: Parte 2. Info Q Brasil, 2019.

FIRICAN, George. **The 10 Vs of big data**. Upside where Data means business, 2017.

FOSTER, J. **What is Big Data?** A Beginner's Guide to the World of Big Data, Data Driver Investor, 2019.

KARAU, H; KONWINSKI, A; WENDELL, P; ZAHARIA, M. **Learning spark: lightning-fast big data analysis**. O'Reilly Media, Inc, 2015.

PAMELA A. Big Data – The Raw Material For 4.0 Business. Interlogica SRL, 2019.

SHAFFER, Tom. The 42 V's of big data and data science. Elder Research, 2017.

Gabarito

Questão 1 – Resposta: C

Resolução: A principal característica do *framework* Spark Apache é a velocidade no processamento dos dados de diferentes formatos de dados (não-estruturados, semiestruturado e estruturado), além de ser simples de usar, e sua API tem suporte para diversos tipos de linguagem de programação: Python, Java, R e Scala.

Questão 2 – Resposta: E

Resolução: A arquitetura do Apache Spark é formada pelas seguintes bibliotecas: Spark SQL, Spark *Streaming*, Spark MLlib e **Spark GraphX**. O Apache Spark executa em uma máquina virtual Java (*JVM – Java Virtual Machine*) e a instalação do JDK é obrigatória. Apache Spark tem bibliotecas de funções para Python, Java, R e Scala.

Questão 3 – Resposta: A

Resolução: A variabilidade no contexto do *Big Data* refere-se a inconsistências nos dados. Elas precisam ser encontradas por métodos de detecção de anomalia e outlier para que uma análise significativa ocorra.



***Real Time Analytics* com Python e Spark**

Autor: Danilo Rodrigues Pereira

Objetivos

A análise em tempo real (*Real Time Analytics*) é referida ao processo de análise de grande volume de dados (*Big Data*) no momento em que está disponível para processamento. Dos arquivos de *log* de servidores e/ou dispositivos aos dados do sensor, os cientistas de dados estão cada vez mais tendo que lidar com fluxos (*streaming*) de dados. Esses dados chegam em um fluxo constante, geralmente de várias aplicações simultaneamente. Embora seja certamente viável armazenar esses fluxos de dados no disco e analisá-los no futuro, às vezes pode ser importante para o negócio da empresa processar e agir com base nos dados que chegam. Transações bancárias e de cartão de crédito, por exemplo, são processadas em tempo real para identificar e recusar transações potencialmente fraudulentas. O objetivo dessa unidade é fornecer conhecimentos teóricos e práticos necessários para você analisar e processar dados em tempo real. No final da unidade você será capaz de:

- Analisar e processar dados em tempo real usando Spark e Python.
- Aprender a utilizar a API Apache Spark *Streaming* para analisar dados de diferentes fontes.
- Fazer integração do API Apache Spark com outras fontes de dados: Apache Kafka e Elasticsearch.


► 1. Introdução à *Real Time Analytics* com Python e Spark

Análise em tempo real (*Real Time Analytics*) é um termo usado para se referir a análises de dados que podem ser acessadas e processadas quando entram em um sistema. Em geral, o termo **análise** é usado para definir padrões de dados que fornecem significado (valor), dos quais os analistas coletam informações valiosas a partir dos dados gerados em tempo real. Como exemplos de aplicações que analisam e processam dados em tempo real, podemos citar: rastreamento de estatísticas sobre visualizações e comentário de página *web*, monitoramento de estradas e rodovias, ou detecção automática de fraudes, por exemplo, fraudes em transação de cartão de crédito (KARAU, 2015). Essas análises de dados podem ser extremamente importantes para as empresas que desejam realizar relatórios dinâmicos para responder rapidamente às tendências de comportamento do usuário.

1.1 Diferença entre processamento de dados em *batch* e *streaming*

O processamento em lote (*batch*) é quando ocorre a análise dos dados que já foram armazenados por um período de tempo. Esses dados geralmente podem ser armazenados em arquivo ou banco de dados etc. O processamento em lote é usado em situações em que você não precisa de resultados de análises em tempo real e quando é mais importante processar grandes volumes de dados para obter informações mais detalhadas do que para obter resultados rápidos de análises. O Hadoop MapReduce é uma ferramenta bastante utilizada para o processamento de dados em lote (RAMÍREZ-GALLEGO, 2018; FERRUCCI, 2018; GLUSHKOVA, 2019).

Um fluxo de dados (*data streaming*) é uma sequência ilimitada de dados que chega continuamente por diferentes fontes de dados.

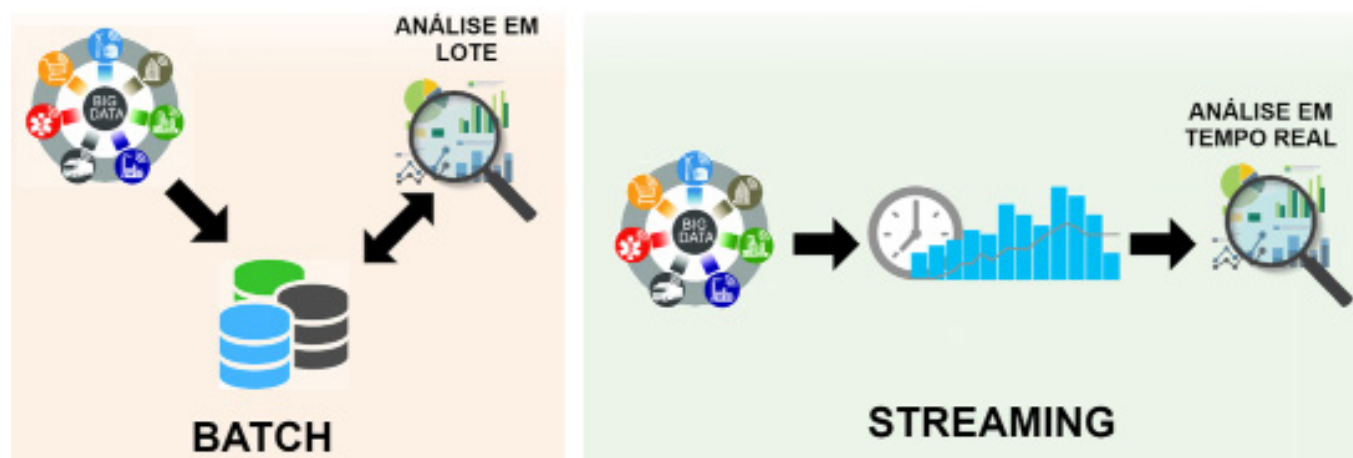


O processamento de fluxo (*streaming*) permite-nos processar dados em tempo real à medida que eles chegam e detectar rapidamente as condições em um pequeno período a partir do ponto de recebimento dos dados. O processamento de *streaming* permite também alimentar dados em ferramentas de análise assim que elas são geradas e obter resultados instantâneos de análise. Existem várias plataformas de processamento de *streaming open-source*, como Apache Kafka (GARG, 2013), Apache Flink (CARBONE, 2015), Apache Storm (IQBAL, 2015), Apache Samza (NOGHABI, 2015) etc.

As desvantagens no processamento em lote (*batch*) a análise e extração das informações são feitas em dados já armazenados (arquivos ou banco de dados). Em contrapartida, no processamento em fluxo (*streaming*), a análise é feita sobre os dados (arquivos) atuais ou mais recentes. Portanto, o processamento em *batch* lida com um grande lote de dados, enquanto o processamento em *streaming* lida com registros individuais ou micro lotes de arquivos com poucos registros de dados (Figura 1). Em relação ao desempenho, a latência do processamento em *batch* será de minutos a horas, enquanto a latência do processamento de *streaming* será em segundos ou milissegundos.

O processamento de dados em fluxo (*streaming*) pode ser melhor para situações em que o tempo de resposta é importante. Já o processamento em *batch* funciona bem quando é necessário fazer uma análise bem detalhada das informações. Portanto, não podemos afirmar quais das técnicas é a melhor. Você deverá escolher a melhor técnica que possa contribuir com o objetivo de negócio da empresa, ou seja, quais informações valiosas você deseja extrair dos dados. Em alguns cenários, você poderá combinar as duas técnicas para resolução do problema.

Figura 1 – Processamento *batch* x *streaming*



Fonte: elaborada pelo autor.

1.2 Apache Spark *Streaming* API

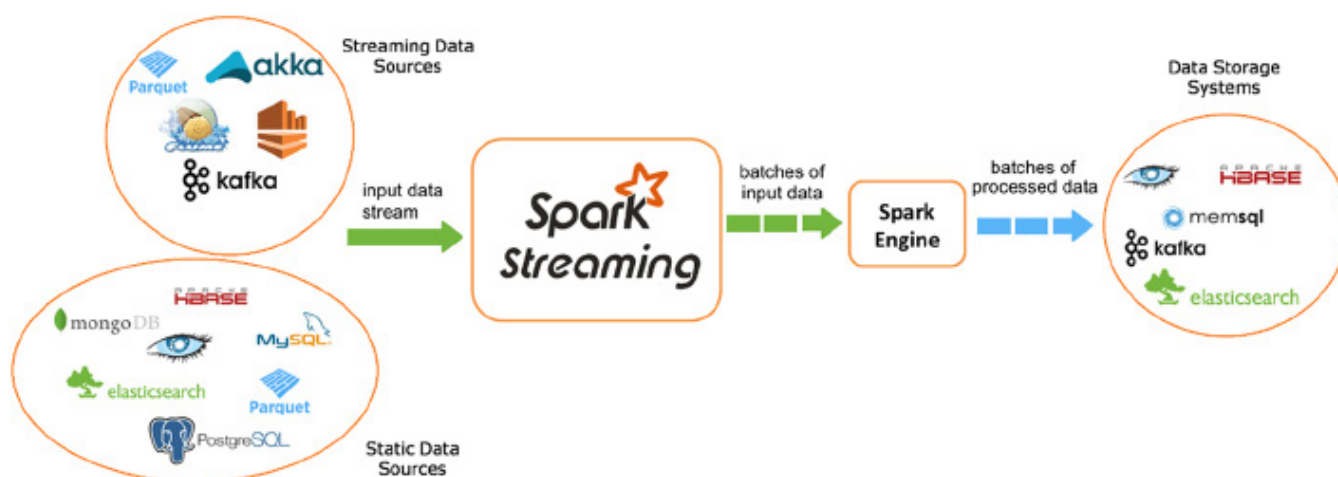
O processamento de fluxo (*streaming*) é um requisito essencial em muitas aplicações que utilizam *Big Data*. Em 2012, o Apache Spark incorporou a biblioteca *Spark Streaming* e sua API *DStreams*, uma das primeiras APIs a ativar processamento de *streaming* usando operadores funcionais de alto nível, como mapear (*map*) e reduzir (*reduce*), e fornecer uma API em Scala, Java e Python. Atualmente muitas empresas usam *DStreams* na produção para aplicativos em tempo real, geralmente processando terabytes de dados por hora. Muito parecido com a biblioteca de conjuntos de dados resilientes – *Resilient Distributed Dataset (RDD)*, no entanto, a API do *DStreams* é baseada em operações de nível relativamente baixo em classes Java e Python (CHAMBERS, 2018; KARAU, 2015).

Assim, em 2016, o projeto Apache Spark adicionou o ***Structured Streaming***, uma nova API de *streaming* criada diretamente em *DataFrames* que suporta tanto otimizações e integração significativamente mais simples com outros códigos *DataFrame* e *Dataset* (CHAMBERS, 2018). A biblioteca *Spark Streaming* pode ser usada para processar dados de *streaming* em tempo real de diferentes fontes,

como sensores, dispositivos de IoT (*Internet Of Things*), redes sociais e transações online e os resultados gerados podem ser armazenados em software como Kafka, HDFS, Cassandra e Elasticsearch (Figura 2).

Um DStream é uma sequência de dados chegando ao longo do tempo. Internamente; cada DStream é representado como uma sequência de RDDs chegando a cada etapa do tempo. DStreams podem ser criados de várias fontes de entrada, como Apache Flume, Apache Kafka ou HDFS. Uma vez construídos, eles oferecem dois tipos de operações: **transformações**, que produzem um novo DStream, e as **operações**, que gravam dados em um sistema externo.

Figura 2 – Visão geral da arquitetura Apache Spark Streaming



Fonte: adaptada de Ashok (2019).

1.2.1 Funcionalidades

O módulo Apache Spark Streaming fornece uma API para manipular fluxos de dados (*streaming*) que correspondem à API *Resilient Distributed Dataset (RDD)* ou também conhecido como **PySpark RDD**. Existem três fases do Apache Spark Streaming:

- **Coleta dos dados (Gathering)** – O Spark Streaming fornece duas categorias de fontes de *streaming* incorporadas e acessa dados de diferentes fontes:

- **Fontes básicas (*Basic sources*)** – Essas são as fontes disponíveis na API `StreamingContext`. Exemplos: sistemas de arquivos e conexões de soquete.
- **Fontes avançadas (*Advanced sources*)** – Essas são fontes como Apache Kafka, Apache Flume, Kinesis, Elasticsearch, entre outros.
- **Processamento (*Processing*)** – Os dados coletados são processados usando algoritmos complexos usando funções de alto nível. Por exemplo: *map*, *reduce*, *join* e *window*.
- **Armazenamento (*Data storage*)** – Os dados processados são enviados para sistemas de arquivos ou bancos de dados. O Apache Spark *Streaming* também fornece uma estrutura de dados de alto nível (DStream API).

1.2.2 *Datasets e Dataframes*

Desde o Apache Spark 2.0, os *DataFrames* e os *Datasets* podem representar dados estáticos e limitados, bem como dados ilimitados de *streaming*. Para criarmos os *DataSet* e *DataFrames* de *streaming*, primeiramente é necessário iniciar uma sessão de trabalho usando a classe **SparkSession**. Com a sessão iniciada, você pode criar os objetos *DataFrames/Dataset* a partir de fontes de *streaming* (*datasources*) e aplicar as mesmas operações que os *DataFrames/Datasets* estáticos (Apache Spark Core).

Em Python, os *DataFrames* de *streaming* podem ser criados por meio da interface *DataStreamReader* retornada por `SparkSession.readStream()`. Semelhante à interface para criar *DataFrame* estático, você pode especificar qual é a fonte de dados de entrada (*datasources*). Na API Spark *Streaming* existem algumas funções para leitura de dados de diferentes fontes: arquivo semiestruturado (CSV, JSON, TXT, XML, ORC, PARQUET), *socket* e Apache Kafka.

Código-fonte

```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.sql.types import StructType

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)

# Lendo textos de um socket
socketDF = spark.readStream.format("<socket>").option("host", "<nome_host>").option("port",
<valor_porta>).load()

# Lendo todos os arquivos .csv no diretorio
userSchema = StructType().add("name", "string").add("age", "integer")

csvDF = spark.readStream.option("sep", ";").schema(userSchema).load("/path/to/directory")
# Equivalent to format("csv").load("/path/to/directory")
```

1.3 PySpark RDD

PySpark é a biblioteca (API) do Python do Apache Spark. No PySpark, os RDDs suportam os mesmos métodos que os equivalentes da linguagem de programação Scala, mas recebem funções do Python e retornam os tipos de coleção do Python.

O PySpark requer a instalação do Python 2.6 ou superior. Por padrão, o PySpark exige que o Python esteja disponível no PATH do sistema operacional ou um executável Python alternativo pode ser especificado configurando a variável de ambiente PYSPARK_PYTHON. Todas as dependências da biblioteca do PySpark, incluindo o Py4J, são incluídas no PySpark e importadas automaticamente.

1.3.1 Porque utilizar RDD?

O *Resilient Distributed Dataset (RDD)* é considerado a estrutura de dados mais importantes no PySpark. É um dos pioneiros na estrutura de dados

sem esquema fundamental, que pode manipular dados estruturados e não estruturados. O compartilhamento de dados na memória torna os RDDs 10-100x mais rápidos que o compartilhamento de rede e disco.

Uma característica importante do RDD é que ele é uma estrutura de dados imutável, ou seja, um objeto cujo estado não pode ser modificado após a criação, mas certamente pode ser transformado.

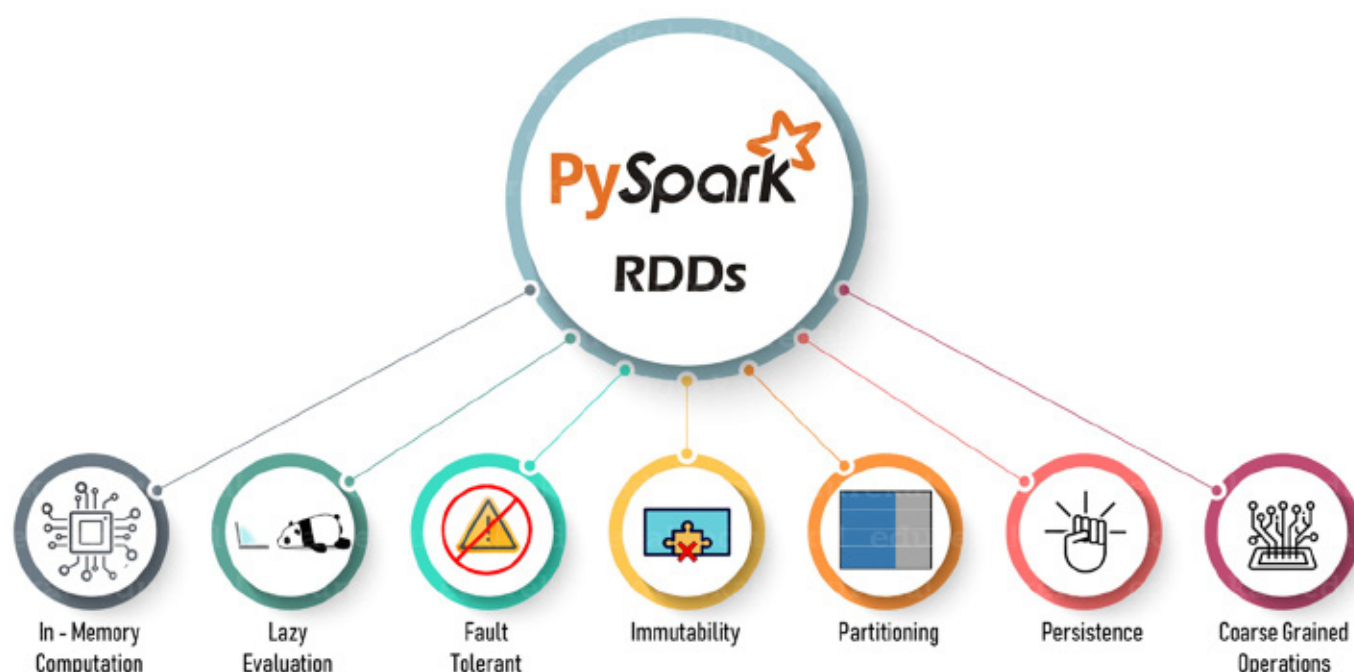
1.3.2 Funcionalidades

A maioria dos programas desenvolvidos em Apache Spark consiste em manipular RDDs, portanto é importante conhecer as funcionalidades dos RDD (Figura 3).

- **Computações em memória (*In-Memory Computations*):** Melhora o desempenho no processamento dos dados.
- **Avaliação “preguiçosa” (*Lazy Evaluation*):** Todas as transformações em RDDs são preguiçosas, ou seja, não calculam seus resultados imediatamente.
- **Tolerante a falhas (*Fault Tolerant*):** Os RDDs rastreiam as informações da linhagem de dados para reconstruir os dados perdidos automaticamente.
- **Imutabilidade (*Immutability*):** Os dados podem ser criados ou recuperados a qualquer momento e, uma vez definidos, seu valor não pode ser alterado.
- **Particionamento (*Partitioning*):** É a unidade fundamental do paralelismo no PySpark RDD.
- **Persistência (*Persistence*):** Os usuários podem reutilizar os RDDs do PySpark e escolher uma estratégia de armazenamento para eles.

- **Operações de granulação (*Coarse-Grained Operations*):** Essas operações são aplicadas a todos os elementos nos conjuntos de dados por meio de mapas (*maps*), filtro (*filter*) ou agrupamento (*group by*) de operação.

Figura 3 – Principais características dos *Resilient Distributed Dataset (RDD)*



Fonte: Swatee (2019).

1.3.3 Criando um objeto RDD

Agora, como já vimos o que é RDD, vamos criar os RDDs usando Python (PySpark). Existem três maneiras de criar um RDD no Apache Spark:

- **Paralelizando a coleção:** os RDDs geralmente são criados por coleção paralelizada, ou seja, pegando uma coleção existente no programa e passando-a para o método `parallelize()` do `SparkContext`.
- Referenciar um conjunto de dados (*dataset*) em um sistema de armazenamento externo (por exemplo, HDFS, Hbase, sistema de arquivos compartilhado).

- Criando RDD a partir de RDDs já existentes.

Mais informações sobre RDD em PySpark, visite o site oficial do Apache Spark e acesse *Documentation* → *RDD Programming Guide*.

1.3.4 Operações em RDD

Existem dois tipos de operações do Apache Spark RDD: **transformações** (*map, filter, join, union* etc) e **ações** (*reduce, count, first* etc). Uma transformação é uma função que produz um novo RDD a partir dos RDDs existentes, mas quando queremos trabalhar com o conjunto de dados real, nesse ponto, a ação é executada. Quando a ação é acionada após o resultado, o novo RDD não é criado como transformação.

PARA SABER MAIS



O *Apache Spark Streaming* utiliza o conceito de *Resilient Distributed Dataset (RDD)*, para manipulação dos dados. O RDD é uma coleção de objetos imutáveis (somente leitura), particionados em um conjunto de nós do *cluster*, podendo somente ser criado através de funções como *map, filter, join* e *groupBy*, executadas em outros RDDs ou meios sistemas de armazenamento como Hadoop Filesystem, Elasticsearch, Kafka etc.

1.4 Integrações com fonte de dados usando Apache Spark Streaming API

O *Apache Streaming API* suporta várias fontes de dados com Apache Kafka e Elasticsearch. Provavelmente, as fontes de dados mais simples que você pode pensar são os arquivos do tipo TXT, JSON, XML, CSV e Parquet.

1.4.1 Apache Kafka

O Apache Kafka é uma plataforma distribuída de código-fonte livre (*open-source*) de processamento de mensagens e *streams* desenvolvida pela Apache Software Foundation, escrita na linguagem de programação Java e Scala. O principal objetivo do projeto Kafka é fornecer uma plataforma unificada, de alta capacidade e baixa latência para tratamento de dados em tempo real (GARG, 2013).

- **Mensagens** – São os principais recursos do Apache Kafka. Cada mensagem em Kafka consiste em uma chave, um valor e data/hora. Todos os eventos podem ser resumidos em mensagens, sendo consumidas e produzidas através de **tópicos**. Uma mensagem pode ser desde uma simples em String com “Olá mundo!” ou até mesmo um JSON contendo *logs* de servidores de aplicações. Outro conceito importante são os tópicos.
- **Tópicos** – É a maneira de categorizar ou classificar grupos de mensagens dentro do Kafka. Todas as mensagens enviadas para o Kafka permanecem em um tópico.
- **Produtor (Producer)** – É responsável por enviar a mensagem para um tópico específico. Uma vez que uma mensagem é produzida em um tópico, o próprio Kafka organiza a mensagem em uma partição, garantindo sempre a ordem das mensagens produzidas, como citado anteriormente.
- **Leitura de dados** – Para iniciar a leitura dos dados em Kafka, primeiro você precisa escolher uma das seguintes opções: `assign`, `subscribe` ou `subscribePattern`. Somente uma delas pode estar presente como opção. Atribuir (`assign`) é uma maneira refinada de especificar não apenas o tópico, mas também as partições de tópico a partir do qual você gostaria de ler. Isso é especificado como uma sequência JSON `{"topicoA": [0,1], "tópico B": [2,4]}`. Já as opções `subscribe` e `subscreeverPattern` são formas de assinar um

ou mais tópicos, especificando uma lista de tópicos, ou através de um padrão. Em seguida, você precisará especificar os endereços que o Kafka fornece para conectar para o serviço: kafka.
bootstrap.servers.

A seguir, um exemplo de código-fonte em Python que faz a leitura de dados em tópicos e salva o resultado na estrutura de dados (*dataFrame*).

Código-fonte

```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)

# Conectando em um tópico específico
df1 = spark.readStream.format("kafka").option("kafka.bootstrap.servers",
"<host1:port1>,<host2:port2>").option("subscribe", "topicoA").load()

# Conectando em vários tópicos
df2 = spark.readStream.format("kafka").option("kafka.bootstrap.servers",
"<host1:port1>,<host2:port2>").option("subscribe", "topicoA,topicoB").load()

# Conectando no tópico padrão (pattern)
df3 = spark.readStream.format("kafka").option("kafka.bootstrap.servers",
"<host1:port1>,<host2:port2>").option("subscribePattern", "topic.*").load()
```

1.4.2 Elasticsearch

O Elasticsearch é uma ferramenta de distribuição gratuita e utilizada para realização de buscas e análise de dados em grandes volumes de dados, permitindo indexar documentos e realizar buscas em tempo real (*streaming*). Outra característica importante do Elasticsearch é o desempenho. Ele pode ser implementado em qualquer sistema independentemente da plataforma, pois fornece uma API REST.

Além disso, o Elasticsearch é altamente escalável, podendo utilizar apenas um servidor ou muitos servidores simultâneos (GORMLEY, 2015).

O Elasticsearch pode ser executado como um serviço de nuvem (*cluster*), no seu próprio servidor ou máquina virtual. O Elasticsearch oferece uma versão gratuita que você pode baixar e instalar. Como ele é executado na JVM, você também precisa fazer a instalação do Java. Para obter mais informações sobre download e instalação visite o site oficial acessando a seção *Download*.

Para usar o Elasticsearch com Apache Spark, primeiramente precisamos fazer o *download* do *conector* chamado *elasticsearch-hadoop*, disponível no site oficial do Elasticsearch. Infelizmente, não é possível fazer a instalação do *elasticsearch-hadoop* via comando *pip install*. Você deve fazer o *download* do arquivo binário (.jar) e indicar o caminho do jar para o Apache Spark. Se você estiver executando um Jupyter Notebook, poderá adicionar o jar no *environment* antes de estabelecer um ContextSpark. Veja o exemplo:

Código-fonte

```
import os
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)

# setando environment variavel PYSPARK_SUBMIT_ARGS
os.environ['PYSPARK_SUBMIT_ARGS'] = '--jars elasticsearch-hadoop-6.1.1/dist/elasticsearch-spark-20_2.11-6.1.1.jar pyspark-shell'

sc = SparkContext(appName="<Nome da aplicação>")

....
```



PARA SABER MAIS

O Apache Spark *Streaming* API contém uma biblioteca de funções para suporte no desenvolvimento de programas para leitura e escrita de dados para o formato do Elasticsearch apenas em Scala e Java, mas não em Python.

1.4.2.1 Criando Spark RDD a partir de arquivos de logs do Apache e enviando para o Elasticsearch

A chave para entender o processo de leitura e escrita no Elasticsearch é que o Elasticsearch grava os dados em um banco de dados no formato JSON. Para o exemplo a seguir, vamos desenvolver um código, e a partir de um arquivo de log do Apache vamos criar um Spark RDD. Em seguida, vamos escrever uma função `conversao()` para ler cada string em grupos de expressão regular, escolher os campos do log que queremos salvar no Elasticsearch e passá-la de volta como um dicionário (*map*).

Código-fonte adaptado (WALKER, 2018)

```
import os
import json
import hashlib
import re
import pyspark
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)

def add_ID_JSON(data):
    j=json.dumps(data).encode('ascii', 'ignore')
    data['document_id_json '] = hashlib.sha224(j).hexdigest()
```

```
return (data['document_id_json'], json.dumps(data))

def conversao(string):
    s = p.match(string)
    d = {}
    d['IP'] = s.group(1)
    d['Data'] = s.group(4)
    d['Operacao'] = s.group(5)
    d['URI'] = s.group(6)

    return d

expressao_regular = '^(\S+) (\S+) (\S+) \[([w:/+|-\d{4}])\] "(?!\S+)\s?(?!)\s?(?!)" (\d{3}|-) (\d+|-)\s?"?("[^"]*)"?\s?"?("[^"]*)"?"?$'

p=re.compile(expressao_regular)

RDD= sc.textFile("/<caminho_dos_arquivos/apache_logs>")
RDD_2 = RDD.map(conversao)
RDD_3 = RDD_2.map(add_ID_JSON)

#Configurando o Elasticsearch (nó, porta e caminho do servidor principal)
es_write_conf = {

    "es.nodes": "localhost",
    "es.port": "9200",
    "es.resource": 'testes/apache',
    "es.input.json": "yes",
    "es.mapping.id": "document_id_json"

}

RDD_3.saveAsNewAPIHadoopFile(path='-', outputFormatClass="org.elasticsearch.hadoop.  
mr.EsOutputFormat", keyClass="org.apache.hadoop.io.NullWritable", valueClass="org.  
elasticsearch.hadoop.mr.LinkedMapWritable", conf=es_write_conf)
```

o desenvolvimento de suas aplicações, verifique a documentação para obter mais informações sobre o Kafka e versões disponíveis.

1.5 Estudo de caso: Identificando as *hashtags* do Twitter usando Apache Spark *Streaming* e Python

Nesse estudo de caso, vamos criar um aplicativo para extrair e contabilizar as *hashtags* em tempo real da media social chamada Twitter, usando Python e Apache Spark *Streaming* (Figura 4) (exemplo adaptado de Hanée, 2019).

1. Obtendo credenciais para as APIs do Twitter

Antes de começar o desenvolvimento, você precisará ter uma conta no Twitter. Após isso, **é necessário** registrar o novo projeto no site oficial do TwitterApps, clicando em “Criar um aplicativo” e, em seguida, preencha formulário solicitado. No final desse processo, você receberá as credencias (chaves) necessárias para ter acesso ao ***Twitter HTTP Client***.

► 2. Criando conexão com o *Twitter HTTP Client*

Nesta etapa, vamos criar uma classe em Python para receber os *tweets* da API do Twitter e enviar para o Apache Spark *Streaming*.

• **twitter_app.py**

```
import socket
import sys
import requests
import requests_oauthlib
import json
```

```

# Iniciando as variaveis com as credencias obtidos, após o preenchimento do
# formulário no Twitter Apps
ACCESS_TOKEN_TWITTER = '<ACCESS_TOKEN>'
ACCESS_SECRET_TWITTER = '<ACCESS_SECRET>'
CONSUMER_KEY_TWITTER = '<CONSUMER_KEY>'
CONSUMER_SECRET_TWITTER = '<CONSUMER_SECRET>'

# Conectando no Twitter
autorizacao = requests_oauthlib.OAuth1(CONSUMER_KEY_TWITTER, CONSUMER_
SECRET_TWITTER, ACCESS_TOKEN_TWITTER, ACCESS_SECRET_TWITTER)

# Função para filtrar as hastags '#'
def get_tweets():
    url = 'https://stream.twitter.com/1.1/statuses/filter.json'
    query_data = [('language', 'pt'), ('locations', '-130,-20,100,50'),('track','#')]
    query_url = url + '?' + '&'.join([str(t[0]) + '=' + str(t[1]) for t in query_data])
    response = requests.get(query_url, auth=autorizacao, stream=True)
    return response

# Função para enviar os dados filtrados para o Apache Spark Streaming
def envia_tweets_apache_spark_streaming(http_resp, tcp_connection):
    for linha in http_resp.iter_lines():
        try:
            texto_completo_tweet = json.loads(linha)
            tweet_texto = texto_completo_tweet['text']
            print("Tweet Recebido: " + tweet_texto)
            print ("-----")
            tcp_connection.send(tweet_texto + '\n')
        except:
            e = sys.exc_info()[0]
            print("Erro inesperado: %s" % e)

TCP_IP = «localhost»
TCP_PORT = 9009

conn = None
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

# Esperando a conexão com o TCP connection
conn, addr = s.accept()
resp = get_tweets()
envia_tweets_apache_spark_streaming (resp, conn)

```

► 3. Configurando a aplicação para conectar com o Apache Spark *Streaming*

Nessa etapa, vamos fazer a conexão com o Apache Spark *Streaming* para o processamento em tempo real (*streaming*); para os *tweets* recebidos, extrairá os *hashtags* e calculará quantas *hashtags* foram mencionadas.

- **main.py**

```
import socket
import sys
import requests
import requests_oauthlib
import json
from pyspark import SparkConf, SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import Row, SqlContext

conf = SparkConf()
conf.setAppName("<Nome_da_aplicacao>")
sc = SparkContext(conf=conf)
porta = 9009

dataStream = ssc.socketTextStream("localhost", porta)

# Fazendo o split cada tweet e armazenando cada palavras num vetor
palavras = Datastream.flatMap(lambda line: line.split(" "))

# filtrando as palavras para obter apenas hashtags e depois salvar cada #hashtag
em um dicionário
hashtags = palavras.filter(lambda w: '#' in w).map(lambda x: (x, 1))

# contabilizando cada hashtag
tags_totals = hashtags.updateStateByKey(aggregate_tags_count)
tags_totals.foreachRDD(process_RDD)

# Iniciando o processo de streaming
ssc.start()
ssc.awaitTermination()
```



```

def aggregate_tags_count (valores, total):
    return sum(valores) + (total or 0)

def get_sql_context_instance (spark_context):
    if ('sqlContextSingletonInstance' not in globals ())
        globals()[ 'sqlContextSingletonInstance' ] = SqlContext (spark_context)
    return globals()[ 'sqlContextSingletonInstance' ]

def process_RDD (time, RDD):
    try:
        sql_context= Get_sql_context_instance(RDD.Context)
        row_rdd = RDD.map(lambda w: Row(hashtag = w [ 0 ], hashtag_count = w [ 1 ]))
        # Criando um dataframe
        hashtags_df = sql_context.createDataFrame(row_rdd)

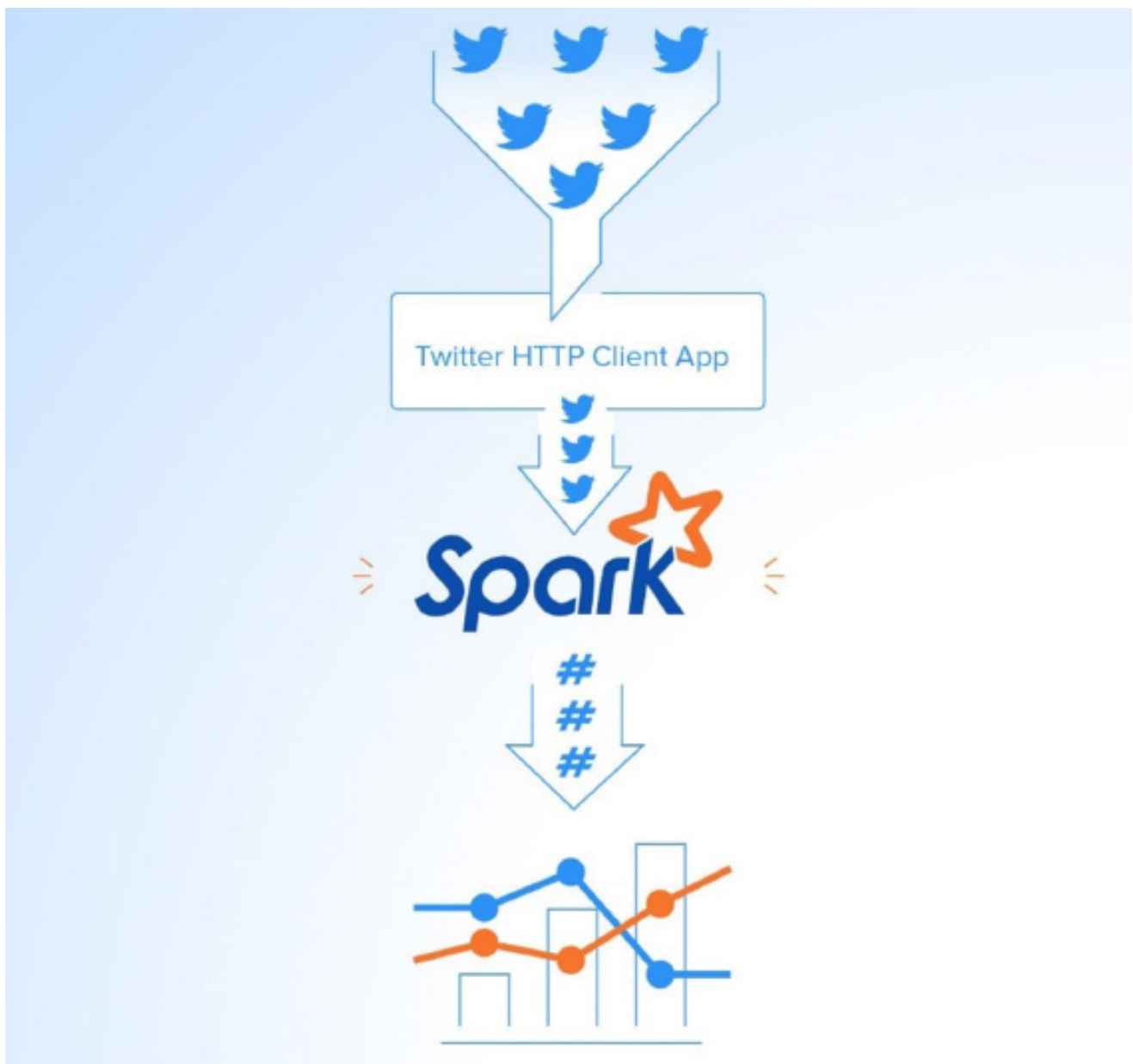
        # Registrando o dataframe na tabela temporaria chamada hashtags
        hashtags_df.registerTempTable( "hashtags" )

        # Filtrando e exibindo as 15 hashtags mais contabilizada usando SQL
        hashtag_counts_df = Sql_context.sql( "select hashtag, hashtags from  hashtag_
count order by hashtag_count desc limit 15" )

        hashtag_counts_df.show()
    except :
        and = sys.exc_info()[ 0 ]
        print ( "Erro inesperado: %s " %e)

```

Figura 4 – Identificando e contabilizando as *hashtags* do Twitter usando Apache Spark *Streaming* e Python



Fonte: adaptada de Hanée (2017).

ASSIMILE



Embora o processamento em fluxo (*streaming*) e o processamento em lote (*batch*) pareçam processos distintos, na prática eles geralmente precisam trabalhar juntos. Por exemplo, os aplicativos de *streaming* geralmente

precisam associar dados de entrada a um conjunto de dados gravados por um trabalho em *batch*, e a saída de trabalhos de *streaming* geralmente é de arquivos ou tabelas que são consultados em trabalhos em *batch* (CHAMBERS, 2018).

TEORIA EM PRÁTICA



Análise de dados em tempo real (*Real Time Analytics*) é um termo usado para se referir a análises de dados que podem ser acessadas e processadas quando entram em um sistema ou aplicações. Em geral, o termo análise é usado para definir padrões de dados que fornecem significado (valor), onde os analistas coletam informações valiosas a partir dos dados gerados em tempo real. Um exemplo clássico de análise de dados em tempo real são os logs gerados pelas aplicações. Esses arquivos de *logs* contêm muitas informações que podem ajudar na análise de erros que correm durante a execução das aplicações. Diante desse cenário, você poderá utilizar a API Apache Spark *Streaming* para criar algoritmos que possam fazer leituras de arquivos de logs de servidores de aplicações (JBoss, Wildfly, Tomcat) para filtrar todos os erros inesperados durante a execução dos programas.

VERIFICAÇÃO DE LEITURA



1. A API Apache Spark *Streaming* é um *framework* que permite manipular dados em *streaming* (tempo real) e algumas de suas características são:

- a. Tem suporte para as linguagens de programação Java, Scala, Python e R.
 - b. Não tem suporte para processar dados em lote (*batch*).
 - c. PySpark é a biblioteca do Python do Apache Spark e requer a instalação do Python 3 ou superior.
 - d. O Apache Spark *Streaming* fornece uma API para manipular fluxos de dados (*streaming*) que correspondem à API *Resilient Distributed Dataset (RDD)*. Os objetos RDDs são imutáveis.
 - e. O Apache *Streaming* API tem suporte apenas para as várias fontes de dados com Apache Kafka e Elasticsearch.
2. Sobre o Apache Spark *Streaming* API, analise as seguintes afirmações:
- I. O processamento de fluxo (*streaming*) é um requisito essencial em muitas aplicações que utilizam Big Data e, em 2018, o Apache Spark incorporou a biblioteca Spark *Streaming* e sua API DStreams.
 - II. Tem suporte para as linguagens de programação Java, Scala, Python e R.
 - III. Tem suporte para processamento de dados em lote (*batch*) e em fluxo (*streaming*).
- a. Todas as afirmações são verdadeiras.
 - b. Apenas a afirmação I é verdadeira.

- c. Apenas a afirmação III é verdadeira.
 - d. Apenas as afirmações I e III são verdadeiras.
 - e. Todas as afirmações são falsas.
3. Quais as principais características dos *Resilient Distributed Dataset (RDD)*?
- a. Imutabilidade, particionamento, tolerante a falhas e persistência.
 - b. Imutabilidade, particionamento, tolerante a falhas e performance.
 - c. Não imutabilidade, particionamento, tolerante a falhas e performance.
 - d. Não imutabilidade, particionamento, tolerante a falhas e alta latência.
 - e. Imutabilidade, particionamento, tolerante a falhas e baixa latência.

Referências bibliográficas

ASHOK N. **Apache Spark**. Mitois Technologies, 2019.

CARBONE, P; KATSIFODIMOS, A.; EWEN, S.; MARKL, V.; HARIDI, S.; TZOUMAS, K. Apache flink: Stream and batch processing in a single engine. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, 2015.

CHAMBERS, B.; ZAHARIA, M. **Spark: The Definitive Guide**: Big Data Processing Made Simple. San Francisco: O'Reilly Media, 2018.

DITTRICH, J.; QUIANÉ-RUIZ, J. Efficient big data processing in Hadoop MapReduce. **Proceedings of the VLDB Endowment**, v. 5, n. 12, p. 2014-2015, 2012.

FERRUCCI, F.; SALZA, P.; SARRO, F. Using Hadoop MapReduce for parallel genetic algorithms: a comparison of the global, grid and island models. **Evolutionary computation**, v. 26, n. 4, p. 535-567, 2018.

GARG, N. **Apache Kafka**. Packt Publishing Ltd, 2013.

GLUSHKOVA, D.; JOVANOVIĆ, P.; ABELLÓ, A. Mapreduce performance model for Hadoop 2. X. **Information Systems**, v. 79, p. 32-43, 2019.

GORMLEY, C.; TONG, Z. **Elasticsearch: the definitive guide: a distributed real-time search and analytics engine**. O'Reilly Media, Inc., 2015.

HANEE', M. **Apache Spark Streaming Tutorial: Identifying Twitter Trending Hashtags**. Toptal Project, 2017.

IQBAL, M. H.; SOOMRO, T. R. Big data analysis: Apache storm perspective. **International journal of computer trends and technology**, 2015.

KARAU, H.; KONWINSKI, A.; WENDELL, P.; ZAHARIA, M. **Learning spark: lightning-fast big data analysis**. O'Reilly Media, Inc, 2015.

MCDONALD, C.; DOWNARD, I. **Getting Started with Apache Spark from Inception to Production**, MapR Technologies - USA, 2018.

NOGHABI, S. A.; PARAMASIVAM, K.; PAN, Y.; RAMESH, N.; BRINGHURST, J.; GUPTA, I.; CAMPBELL, R. H. **Samza: stateful scalable stream processing at LinkedIn**. Proceedings of the VLDB Endowment, 2017.

RAMÍREZ-GALLEGO, S.; FERNÁNDEZ, A.; García, S.; Chen, M.; Herrera, F. **Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce**. Information Fusion, 2018.


SWATEE C. **RDDs in PySpark – Building Blocks of PySpark**. Edureka -Training & Certification, 2019.

WALKER R. **How to write Apache Spark data to ElasticSearch using Python**. Machine Learning & Big Data Blog, 2018.

Gabarito

Questão 1 – Resposta: D

Resolução: A API Apache Spark *Streaming* não tem suporte para a linguagem R. Ela tem suporte tanto para processamento em lote (*batch*), quanto para o processamento de dados em fluxo (*streaming*). A PySpark é a biblioteca do Python do Apache Spark e



requer a instalação do Python 2.6 ou superior e não 3.x ou superior. O Apache *Streaming* API tem suporte para fazer integração com diversas fontes de dados e não apenas com o Elasticsearch e o Apache Kafka.

Questão 2 – Resposta: C

Resolução: O processamento de dados em *streaming* é um requisito essencial em muitas aplicações que utilizam *Big Data* e, em 2012, o Apache Spark incorporou a biblioteca *Spark Streaming*, não em 2018. O Apache *Spark Streaming* não tem suporte para a linguagem de programação R.

Questão 3 – Resposta: A

Resolução: As principais características dos *Resilient Distributed Dataset (RDDs)* são: Imutabilidade, particionamento, tolerante a falhas e persistência.



Bons estudos!

