Desvendando Data Science em Sistemas Distribuídos com Apache Spark

Fundamentos e aplicações do Apache Spark em Data Science

Apresentação

Alexandre, Data Engineer na DigiFarmz Smart Agriculture, está concluindo sua especialização em Big Data no Instituto Infnet. Especializado em Data Lakehouse, Data Warehouse e Machine Learning com Spark, possui certificações como Microsoft Certified: Azure Data Fundamentals. Reconhecido por sua versatilidade e comprometimento com a excelência em projetos de engenharia de dados.

Alexandre Castro
Data Engineer

alexandre.dcastro@al.infnet.edu.br

linkedin.com/in/alexandremcastro/

Apresentação

Letícia, graduada em Engenharia de Produção, é Analytics Engineer na DigiFarmz Smart Agriculture, especializada em coletar, tratar e analisar dados com Python, SQL, Power BI e Tableau. Com experiência em gestão de produtos e operações, contribui para o crescimento sustentável de negócios através da análise de dados multidisciplinares.

Letícia Moura
Analytics Engineer

leticiamwork@gmail.com

linkedin.com/in/mouralet/

Apresentação

Matheus é Tech Lead de Ciência de Dados na DigiFarmz Smart Agriculture, com doutorado em Computação pela UFRGS. Especialista em desenvolvimento de produtos de dados e implementação de soluções de machine learning em ambientes empresariais, destacou-se com prêmios acadêmicos, incluindo o Aluno Destaque da SBC. Compartilha seu conhecimento ministrando minicursos em eventos como ERAD/RS, WSCAD e no LNCC.

Matheus Serpa
Data Lead

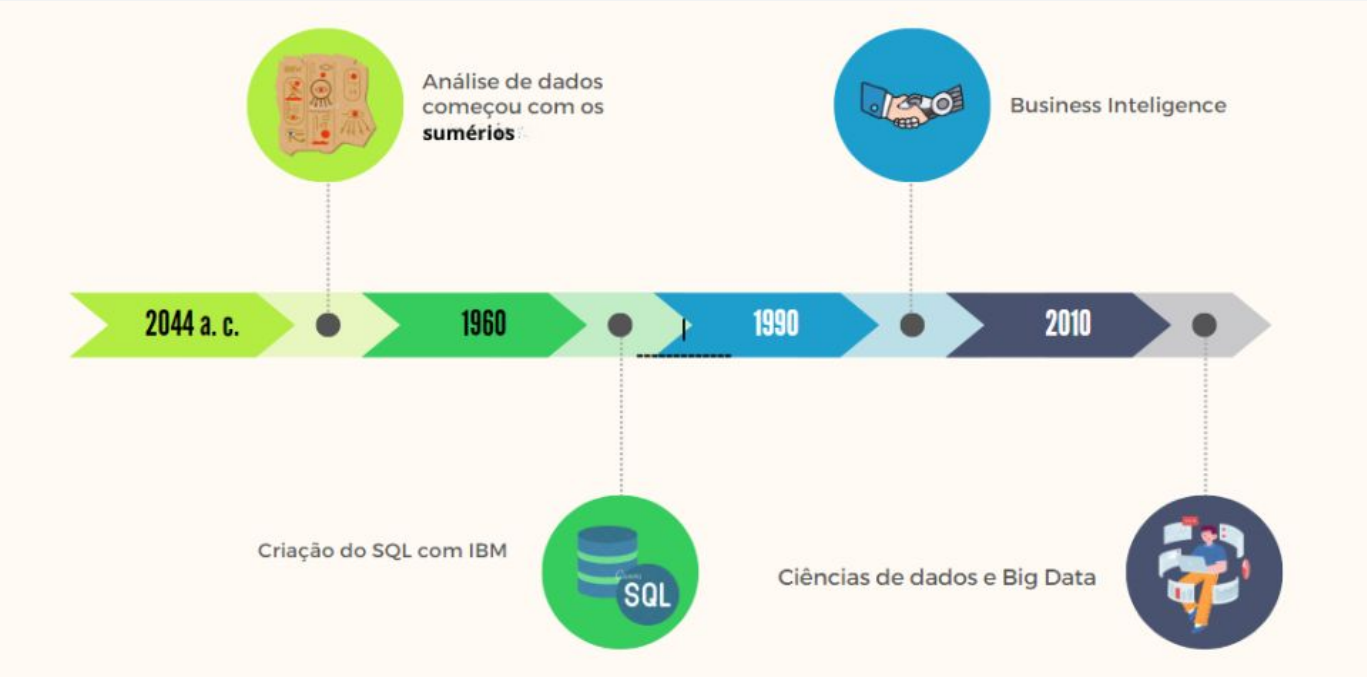
msserpa@inf.ufrgs.br

linkedin.com/in/matheusserpa/

Introdução à ciência de dados

O que é ciência de dados?

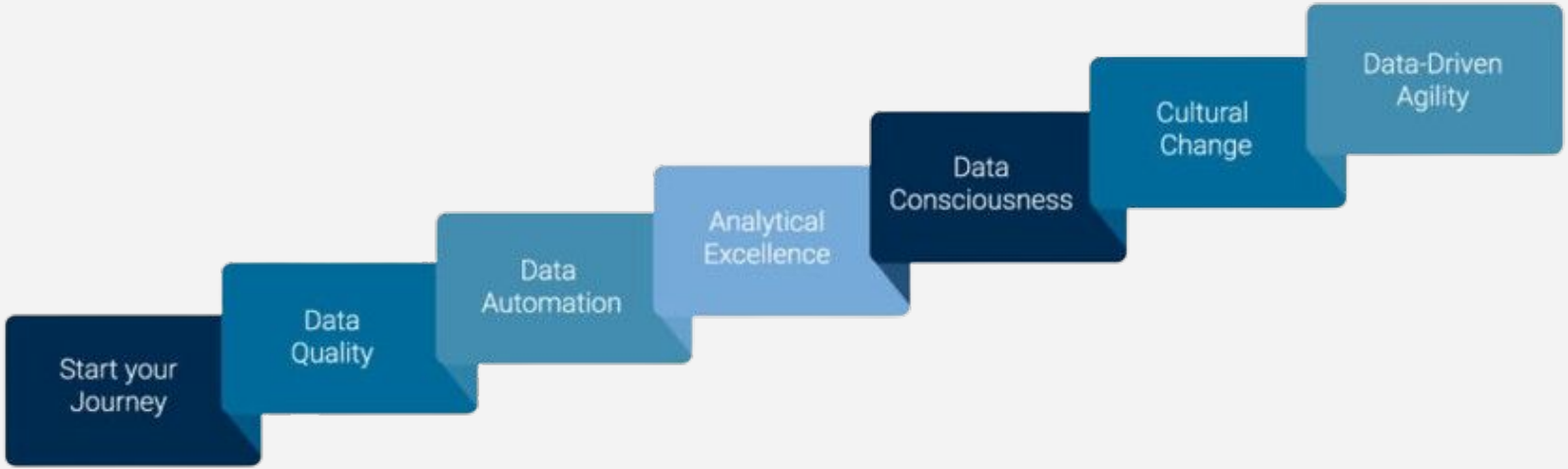
História da ciência de dados



Fonte: História da ciência e análise de dados

O que é ciência de dados?

Evolução da ciência de dados



Fonte: Becoming a Data-Driven Company

O que é ciência de dados?

Processo de desenvolvimento de produtos de dados

1. Concepção da Ideia
2. Aquisição e Preparação de Dados
3. Exploração e Análise de Dados
4. Desenvolvimento e Avaliação de Modelos
5. Implantação e Monitoramento
6. Iteração e Melhoria Contínua

O que é ciência de dados?

Exemplo de processo de desenvolvimento de produtos de dados

Fonte: O ciclo de análise de dados - um roteiro para resolver problemas

O que é ciência de dados?

Profissionais de dados

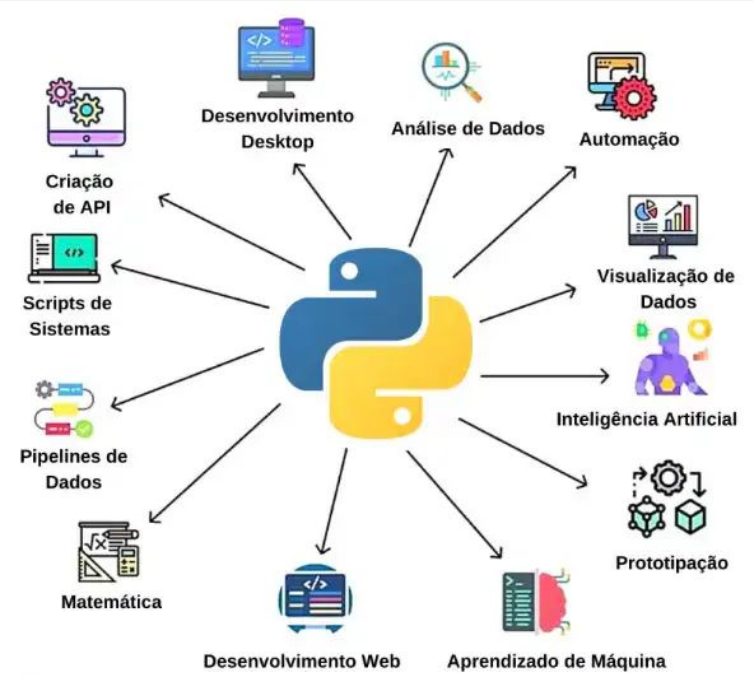
Cientista de dados	Engenheiro de dados	Analista de dados	Engenheiro de analytics
Cria modelos e análises em cima dos dados. Possui conhecimento estatístico e de programação. É uma necessidade de grandes empresas ou em projetos específicos. **Python/R**	É responsável pela manutenção da infraestrutura. Desenvolve código. Não tem conhecimento específico sobre o domínio. Preocupa-se com a **manutenção do código**	É responsável por tirar insights dos dados (ex: Por que estamos perdendo mercado na região X?) Faz análises de negócio. Possui conhecimento de negócio. **Excel/SQL/BI**	Extrai e transforma os dados para análise. Desenvolve o Data Warehouse. Possui conhecimento de negócio e programação. Interage com os analistas e engenheiros de dados. **SQL/DBT/BI**

Fonte: Cientista de dados: quem é, o que faz, quanto ganha?

Fundamentos do Python

Introdução ao Python

Por que Python?



Fonte: Por que aprender Python em 2023?

Introdução ao Python

Fundamentos da linguagem

Python é uma linguagem de programação de alto nível, de código aberto, conhecida por sua simplicidade, facilidade de aprendizado e ampla comunidade ativa.

No contexto da Ciência de Dados, Python é uma ferramenta poderosa devido à sua vasta gama de bibliotecas especializadas, como:

- Pandas
- NumPy
- Matplotlib
- Scikit-learn

Introdução ao Python

Bibliotecas mais usadas em ciência de dados

Pandas: É uma biblioteca essencial para manipulação e análise de dados em Python. Ela oferece estruturas de dados flexíveis e eficientes, como Data Frames e Séries, que permitem a limpeza, transformação e agregação de dados de forma rápida e intuitiva.

Exemplo prático: Carregar um conjunto de dados, explorar suas características e realizar operações de filtragem e agrupamento.

Introdução ao Python

Bibliotecas mais usadas em ciência de dados

NumPy: É fundamental para computação numérica em Python. Ele fornece arrays multidimensionais eficientes e operações matemáticas de alto desempenho, sendo amplamente utilizado em operações de álgebra linear, estatísticas e processamento de sinais.

Exemplo prático: Criar e manipular arrays NumPy para realizar cálculos matemáticos e estatísticos.

Introdução ao Python

Bibliotecas mais usadas em ciência de dados

Matplotlib: Matplotlib é uma biblioteca para criação de gráficos estáticos em Python. Ela oferece uma ampla variedade de tipos de gráficos e opções de personalização para visualizar dados de forma eficaz.

Exemplo prático: Plotar gráficos de linhas, barras, dispersão e histogramas para explorar e comunicar padrões nos dados.

Introdução ao Python

Bibliotecas mais usadas em ciência de dados

Seaborn: É uma biblioteca de visualização de dados baseada no Matplotlib, que facilita a criação de gráficos estatísticos atrativos e informativos. Ele oferece funções de alto nível para plotar visualizações complexas com facilidade.

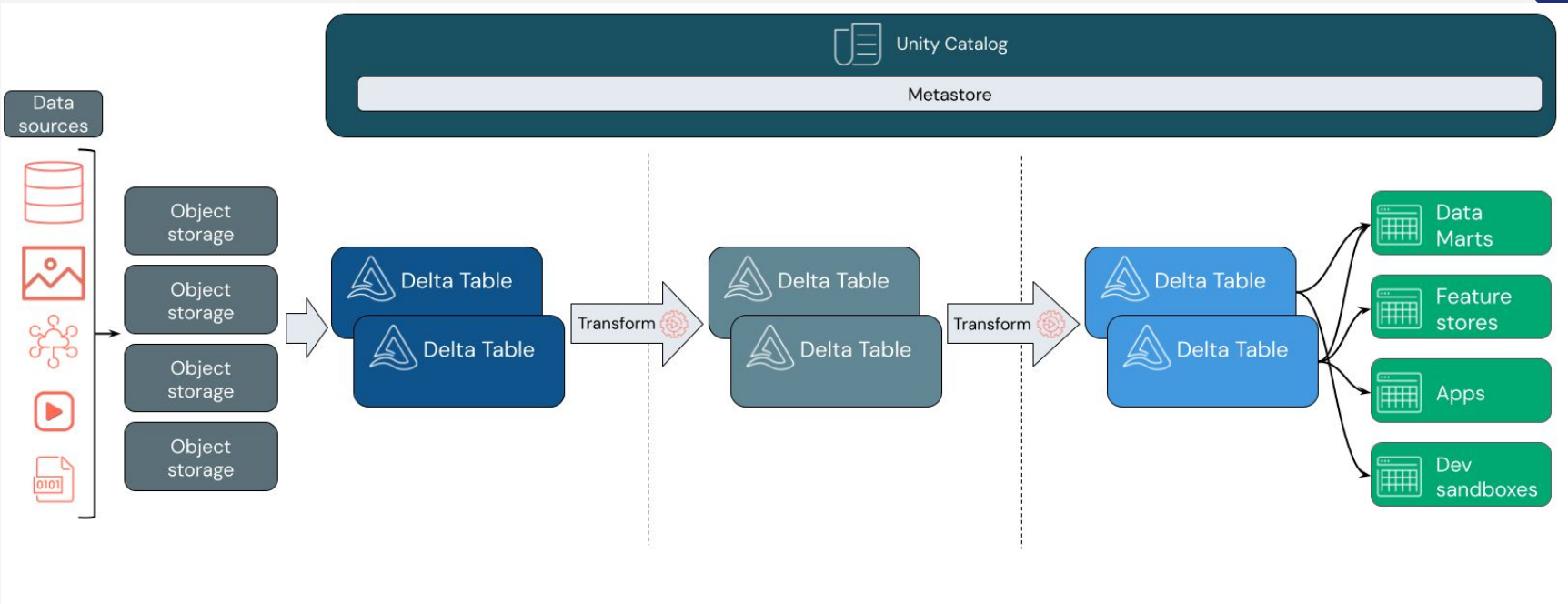
Exemplo prático: Criar gráficos de dispersão com linhas de regressão e mapas de calor para analisar relações entre variáveis e padrões nos dados.

Conhecendo o Databricks

Ambiente de desenvolvimento em nuvem

Conhecendo o Databricks

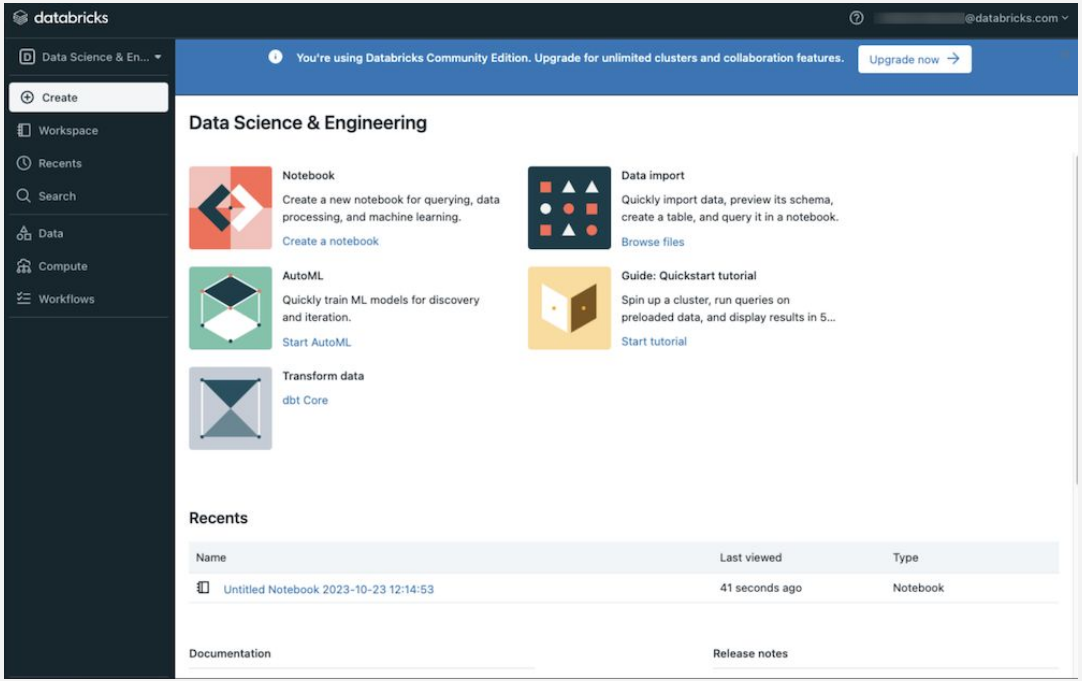
O que é Databricks?



Fonte: Databricks

Conhecendo o Databricks

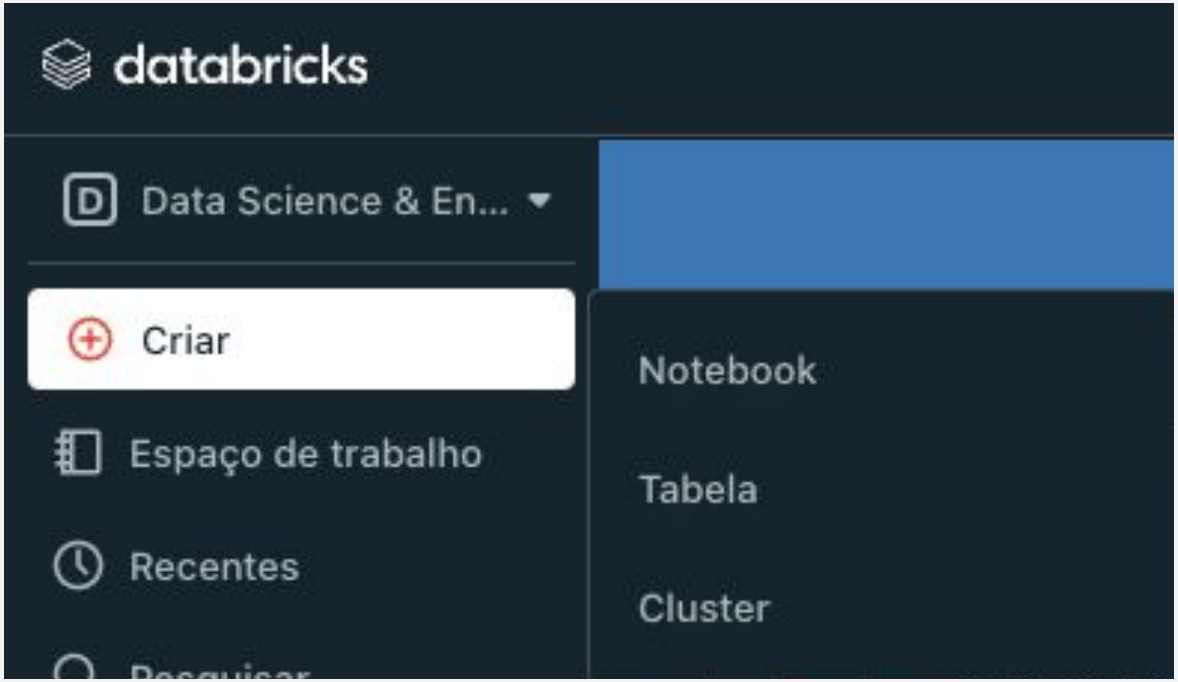
Databricks Community



Link: https://community.cloud.databricks.com/login.html

Conhecendo o Databricks

Conhecendo a interface



Conhecendo o Databricks

Notebooks

1. Business understanding

Uma equipe de análise de dados de uma empresa de telecomunicações precisa entender a probabilidade de churn (rotatividade de clientes) nos próximos meses, utilizando um conjunto de dados históricos como referência.

Com base nos dados de clientes, incluindo informações como gênero, idade, se são casados, têm dependentes, tempo de permanência (tenure), tipo de serviço de telefone e internet, entre outros, a equipe pretende desenvolver um modelo preditivo.

Este modelo será essencial para prever se os clientes têm maior probabilidade de cancelar seus serviços nos próximos meses. Essa previsão é crucial para que a empresa possa tomar medidas proativas, como oferecer promoções ou melhorias nos serviços, para reter esses clientes e reduzir a taxa de churn.

Ao analisar o conjunto de dados, a equipe observará padrões e correlações entre os diferentes atributos dos clientes e a variável de churn. Isso permitirá que eles identifiquem os principais fatores que influenciam a decisão dos clientes de cancelar seus serviços.

Com base nessas descobertas, a empresa poderá implementar estratégias personalizadas de retenção de clientes, visando aumentar a satisfação e fidelidade do cliente, além de otimizar os recursos investidos em marketing e atendimento ao cliente.

1. Data Acquisition

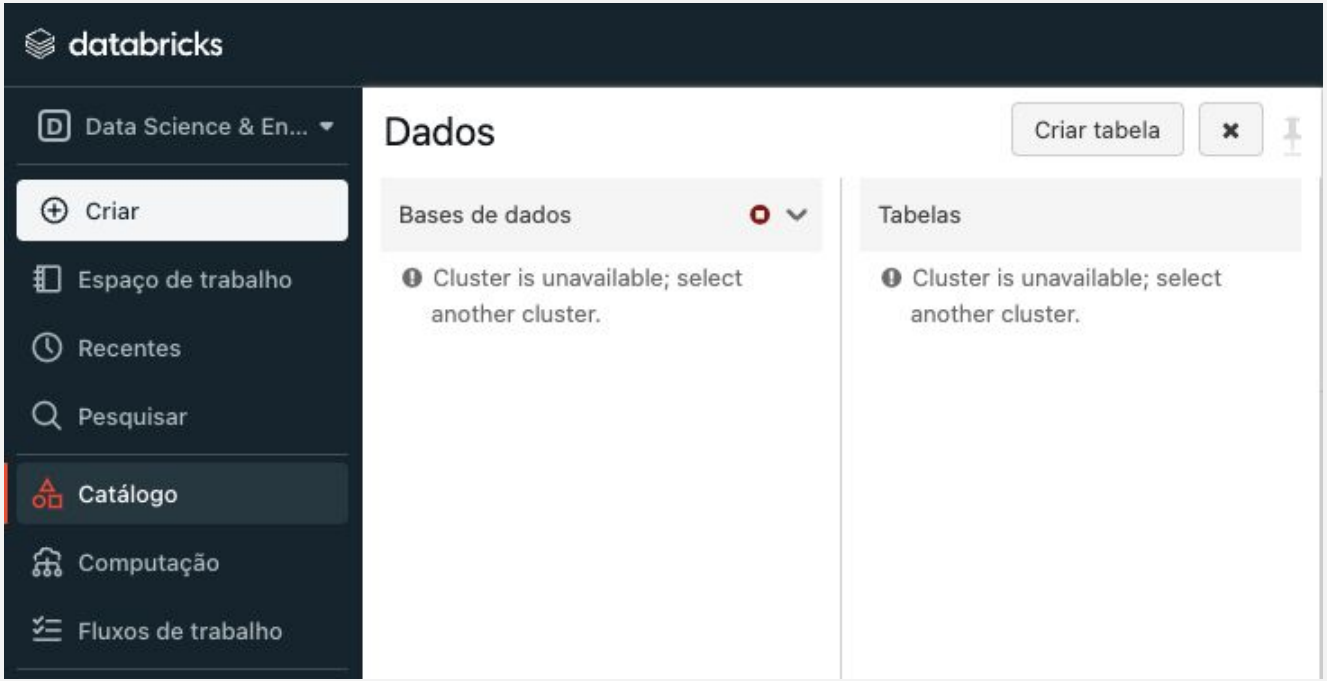
```
▶ 9
dbutils.fs.cp("https://raw.githubusercontent.com/msserpa/ml-datasets/main/rotatividade_de_clientes.csv", "/FileStore/rotatividade_de_clientes.csv")
```

1.1 Loading Data

```
▶ 11
df_raw = spark.read.csv("/FileStore/rotatividade_de_clientes.csv", header=True, inferSchema=True)
```

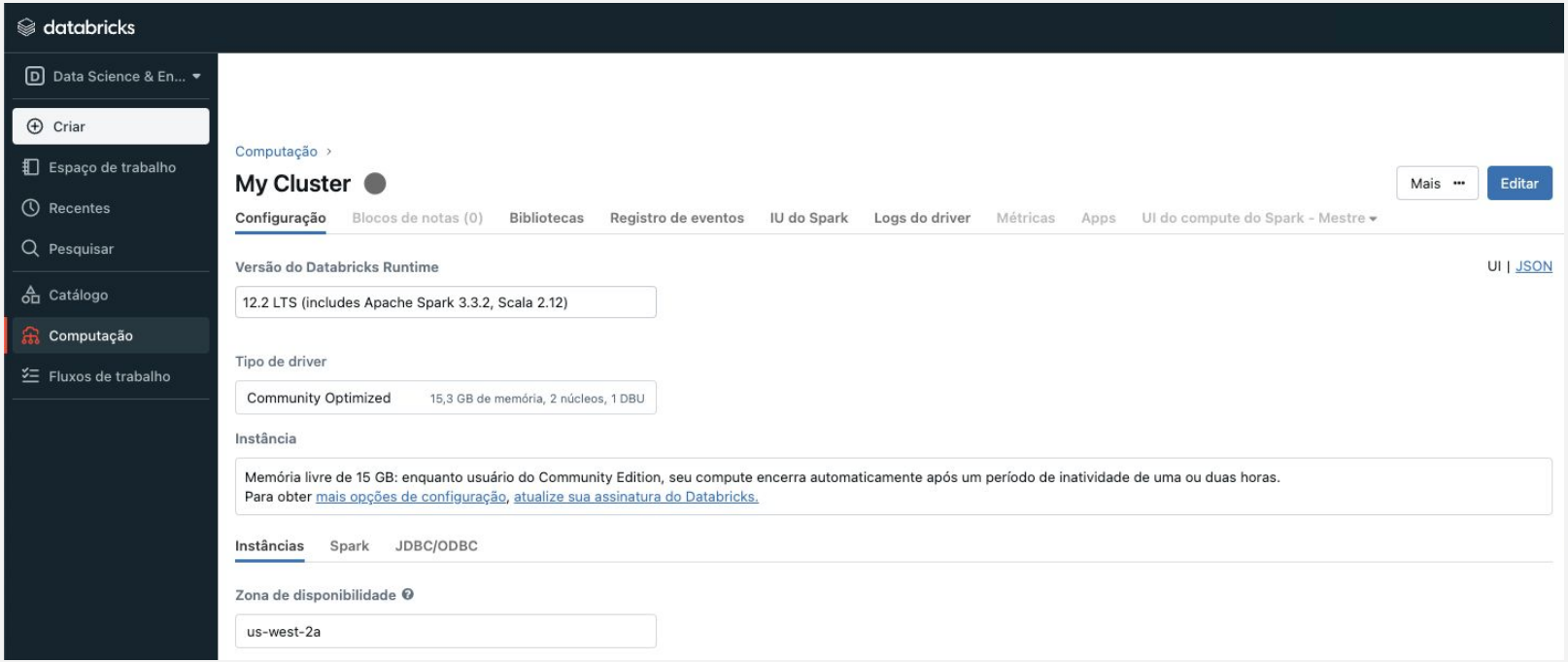
Conhecendo o Databricks

Catálogo



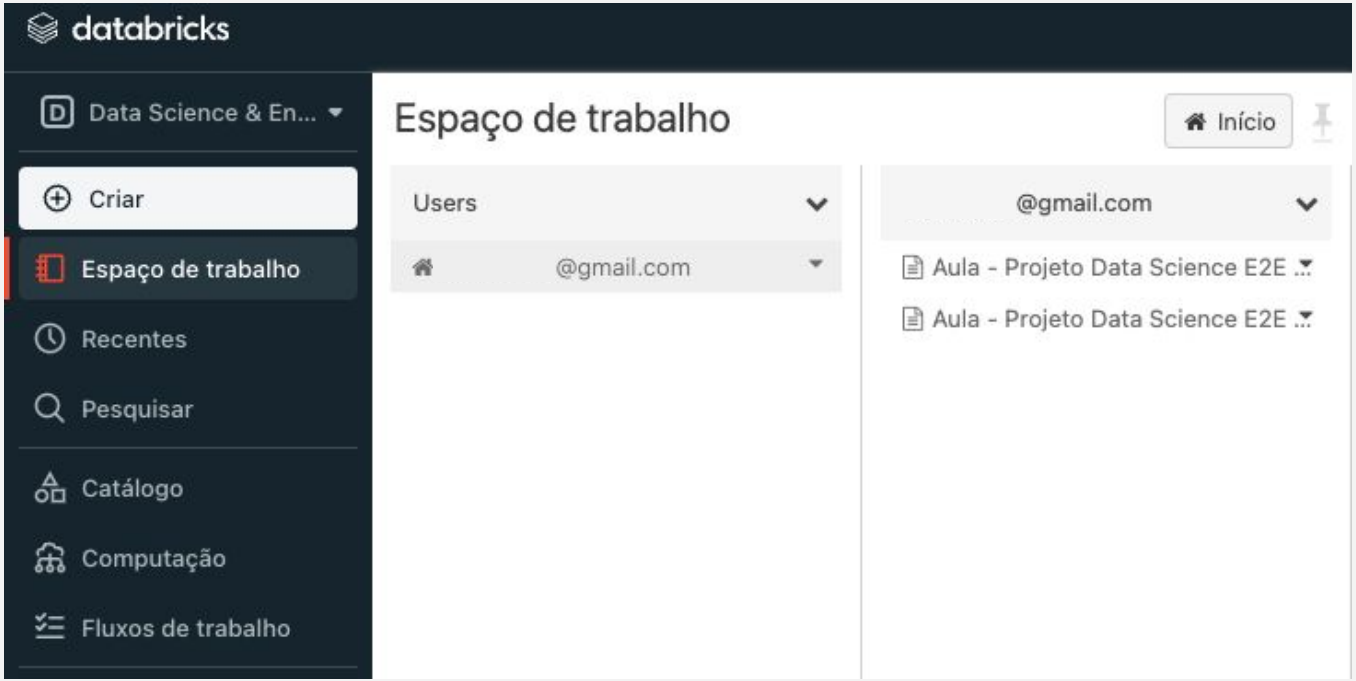
Conhecendo o Databricks

Cluster no Databricks

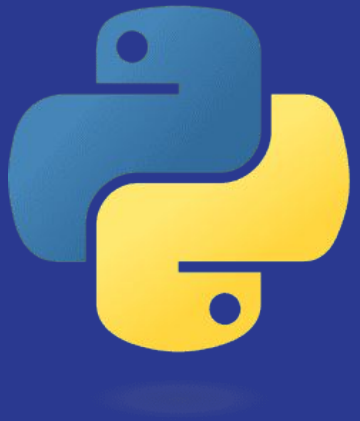


Conhecendo o Databricks

Conhecendo a interface



Prática com o Python



Prática com o Python

Sintaxe básica do Python

```
print("Hello, world!")

"""
Comentários
Múltiplos
"""

x = 5        # Integer Type
y = "John"   # String Type
z = 5.0      # Double Type

# Checando o tipo de dado
print(dtype(x))
```

Prática com o Python

Sintaxe básica do Python

```
# Variáveis permitidas
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

# Variáveis ilegais
2myvar = "John"
my-var = "John"
my var = "John"
```

Prática com o Python

Sintaxe básica do Python

```
# Atribuindo três variáveis a três valores
x, y, z = "Orange", "Banana", "Cherry"

# Lista
lista = ["apple", "banana", "cherry"]

# Tupla
tupla = ("apple", "banana", "cherry")

# Dicionários
dicionario = {'fruits': 'apple', 'banana', 'cherry')
```

Prática com o Python

Sintaxe básica do Python

```
# Condicionais
a = 1
b = 10

if b > a:
  print("b é maior que a")

# Função
def my_function(first_name, last_name):
  print(first_name + " " + last_name)

my_function("Matheus", "Serpa") # Matheus Serpa
```

Prática com o Python

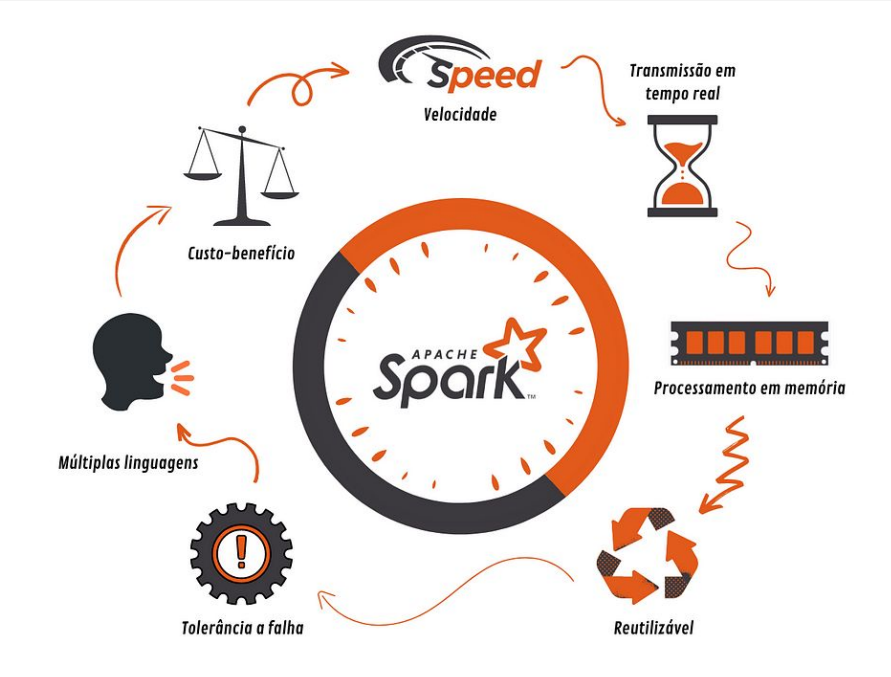
Sintaxe básica do Python

```
# For loop
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)

for x in "banana":
  print(x)

# While Loop
i = 1
while i < 6:
  print(i)
  i += 1
```

Apache Spark em ciência de dados

Apache Spark em ciência de dados

Computação distribuída com Apache Spark



Fonte: Medium

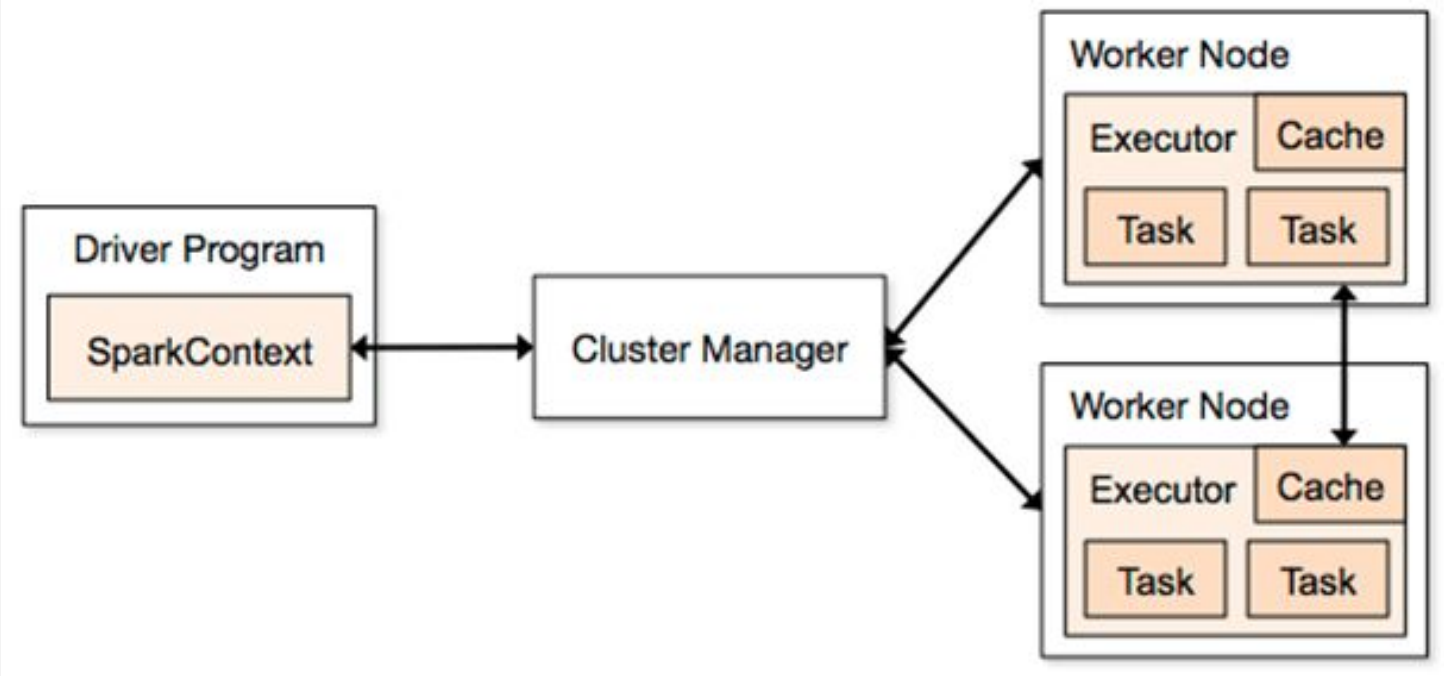
Apache Spark em ciência de dados

Computação distribuída com Apache Spark

O Apache Spark é uma poderosa ferramenta de processamento de Big Data que oferece uma alternativa rápida e flexível ao modelo de processamento de dados em lotes do Apache Hadoop. Surgiu em resposta à necessidade de lidar com a crescente demanda por análise de dados em tempo real e processamento de grandes volumes de dados de forma mais eficiente.

Apache Spark em ciência de dados

Computação distribuída com Apache Spark



Apache Spark em ciência de dados

Computação distribuída com Apache Spark

Relevância no Processamento de Big Data: O Spark se destaca por sua capacidade de processamento em memória, que permite realizar operações de análise de dados de forma significativamente mais rápida do que o modelo tradicional de disco do Hadoop.

Acessibilidade: Além disso, oferece suporte a uma variedade de linguagens de programação, incluindo Scala, Java, Python e R, tornando-o acessível a uma ampla gama de desenvolvedores e cientistas de dados. Sua arquitetura distribuída e tolerante a falhas o torna ideal para lidar com conjuntos de dados massivos e aplicativos de análise de dados em larga escala.

Apache Spark em ciência de dados

História do Apache Spark

Apache Hadoop: Os Primórdios do Big Data: Desenvolvido a partir do conceito de MapReduce do Google, o Hadoop foi uma das primeiras soluções para lidar com grandes volumes de dados, oferecendo processamento distribuído em larga escala.

Apache Spark: A Evolução do Processamento de Big Data: Começando em 2009, o Spark introduziu o processamento em memória, proporcionando ganhos significativos de desempenho em comparação com o modelo de processamento em disco do Hadoop, tornando-o mais adequado para análise de dados em tempo real e processamento iterativo.

Complementaridade e Integração: Embora o Spark tenha trazido inovação, muitas implementações são executadas em cima do Hadoop Distributed File System (HDFS), mantendo a compatibilidade e permitindo que as organizações aproveitem os benefícios do Spark enquanto ainda mantém seus investimentos existentes em infraestrutura Hadoop.

Apache Spark em ciência de dados

Aplicações do Apache Spark em ciência de dados

Processamento de Grandes Volumes de Dados: O Spark é amplamente utilizado em projetos de ciência de dados para lidar com grandes volumes de dados de forma eficiente. Ele permite a análise de conjuntos de dados massivos, incluindo dados estruturados, semi-estruturados e não estruturados, de diversas fontes, como logs de servidores, registros de transações e dados de sensores.

Análise em Tempo Real: Uma das principais vantagens do Spark é sua capacidade de processamento em tempo real. Ele é usado em casos de uso que exigem análise de dados em tempo real, como detecção de fraudes, análise de sentimentos em redes sociais e monitoramento de sistemas de IoT (Internet das Coisas).

Apache Spark em ciência de dados

Aplicações do Apache Spark em ciência de dados

Machine Learning Distribuído: O Spark oferece uma biblioteca de machine learning escalável e distribuída, conhecida como MLlib, que permite a construção e treinamento de modelos de machine learning em grandes conjuntos de dados. Ele é usado em uma variedade de aplicações de machine learning, incluindo classificação, regressão, clustering e recomendação.

Processamento de Gráficos e Análise de Grafos: Além disso, o Spark GraphX é uma extensão do Spark que permite o processamento de grafos em larga escala. Ele é usado em casos de uso que envolvem análise de redes sociais, detecção de comunidades, análise de conexões e muito mais.

Prática Apache Spark Data Frames

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Iniciar Spark Session
spark = SparkSession \
    .builder \
    .appName("Exemplo PySpark") \
    .config("spark.configuração.opção", "valor") \
    .getOrCreate()

# Exemplos de configurações
.config("spark.executor.memory", "2g")
.config("spark.executor.cores", "4")
.config("spark.task.maxFailures", "4")
.config("spark.driver.maxResultSize", "1g")
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Leitura com base em um arquivo CSV.
df = spark.read.csv("file.csv", encoding='UTF-8', inferSchema=True)

# Leitura com base em um arquivo JSON.
df = spark.read.json(path="file.json", header=True)

# Leitura com base em um arquivo Parquet.
df = spark.read.parquet("file.parquet", schema=['col1', 'col2'])

# Leitura com base em um arquivo Avro.
df = spark.read.load("file.avro", format="avro")
```

Introdução ao Spark Data Frames

Parâmetros mais utilizados para leitura de arquivos

path: O caminho para o arquivo ou diretório que contém os dados a serem lidos.

schema: Especifica explicitamente o esquema dos dados a serem lidos. Isso é útil quando o Spark não pode inferir automaticamente o esquema dos dados com precisão.

header: Especifica se a primeira linha do arquivo contém os nomes das colunas (True) ou não (False).

inferSchema: Especifica se o Spark deve tentar inferir automaticamente o esquema dos dados (True) ou não (False).

Introdução ao Spark Data Frames

Parâmetros mais utilizados para leitura de arquivos

delimiter: Especifica o caractere usado para delimitar campos em arquivos de texto (por exemplo, CSV). O padrão é a vírgula (,).

quote: Especifica o caractere usado para citar campos em arquivos de texto.

escape: Especifica o caractere de escape em arquivos de texto.

ignoreLeadingWhiteSpace: Especifica se os espaços em branco iniciais devem ser ignorados.

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Removendo coluna
df = df.drop('col')

# Selecionado e visualizando a coluna
df.select('col').show()

# O resultado da seleção também pode ser armazenado em um Data Frame
df = df.select('col')
df.show()
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Unindo Data Frames
df_c = df_a.union(df_b)

# Juntando Data Frames
df_c = df_a.join(df_b, col_a == col_b, 'left')

df_c = df_a.join(df_b, col_a == col_b, 'right')
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Criando Data Frame
from pyspark.sql import Row

df = spark.createDataFrame([
    Row(col1=1, col2=2., col3='string1'),
    Row(col1=2, col2=3., col3='string2'),
    Row(col1=4, col2=5., col3='string3')])

# Alternativa para criar manualmente um Data Frame
data = [[14, "Tom"], [23, "Alice"], [16, "Bob"], [16, "Bob"]]
schema = ['Idade', 'Nome']

df = spark.createDataFrame(data, schema)
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Mostra o resultado
df.show()

# Mostra o Data Frame sendo n o número de linhas que retornará e truncate o tamanho do retorno dos dados
df.show(n=2, truncate=3)

# Mostra as primeiras 'n' linhas (exclusivo do Databricks)
df.display()

# Recupera os nomes de todas as colunas do Data Frame como uma lista
df.columns
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Mostra o schema no formato de árvore
df.printSchema()

# Calcula estatísticas básicas para colunas numéricas e de cadeia
df.describe('col_numerica').show()

# Retorna todos os registos como uma lista de strings
df.collect()

# Devolve as últimas linhas numéricas como uma lista de strings
df.tail(1)
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Mostra o número de linhas no DataFrame
df.count()

# Mostra todos os nomes de colunas e seus tipos de dados em uma lista
df.dtypes

# Mostra o primeiro registro do Data Frame
df.first()

# Realiza uma cópia do Dataframe
df_new = df.alias('df_novo')
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Renomeando as colunas
df_renamed = df.withColumnRenamed('coluna_atual', 'coluna_nova')

# Adicionando novas colunas
df_with_occupation = df.withColumn(
    "Profissão",
     when(col("Nome") == "Tom", "Arquiteto")
    .when(col("Nome") == "Alice", "Advogada")
    .otherwise("Não informado")
)

# Alterando o tipo de dado
df_new_col = df.withColumn("nome_coluna", col("nome_coluna").cast("double"))
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Alterando texto para maiúsculo
df_upper_case = df.withColumn('Pais', upper(df.Pais))

# Alterando texto para minúsculas
df_lower_case = df.withColumn('Pais', lower(df.Profissao))

# Concatena várias colunas de entrada em uma única coluna.
df.select(lit(5).alias('height')).withColumn('spark_user', lit(True)).take(1)

# Cria uma coluna com valores literais
df = df.select(col("EmpId"), col("Salary").lit("1").alias("valor_lit"))
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Alterando o tipo de dado
df = df.withColumn("nome_coluna", col("nome_coluna").cast("double"))

# Faz uma cópia do Data frame
df_novo = df.alias('df_novo')

# Visualizar coluna com dados ordenados (ascendente ou descendente)
df.sort("nome_coluna", ascending=False).show()

# Visualizar dados agrupados
df.groupBy("nome_coluna").show()
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Filtrando dados ausentes
df = df.filter(df.coluna.isNull())

# Filtrando dados não ausentes (where alternativa à filter)
df = df.where(df.coluna.isNotNull())

# Preenche os valores nulos de uma coluna
df = df.fillna(0, ["coluna"])
df = df.na.fill(0, ["coluna"])

# Retorna a quantidade de linhas que contêm valores distintos
df.distinct().count()
```

Introdução ao Spark Data Frames

Comandos mais utilizados

```
# Remove registros duplicados
df = df.dropDuplicates()
df = df.drop_duplicates()

# Remove registros nulos
df = df.dropna()

# Gravando a um Data Frame
df.write \
  .format("csv") \
  .mode("overwrite") \
  .saveAsTable("nome_tabela")
```

Introdução ao Spark Data Frames

Spark SQL

```
# Criando uma view temporária
df.createTempView("df_sql")
df.createOrReplaceTempView("df_sql")

# Criando uma view temporária global
df.createGlobalTempView("df_sql")
df.createOrReplaceGlobalTempView("df_sql")

# Consultando através do SQL com o Spark
spark.sql("SELECT * FROM df_sql").show()
df = spark.sql("SELECT * FROM df_sql")
```

Machine Learning com Apache Spark (MLlib)

Visão geral

O MLlib é uma biblioteca de machine learning escalável e distribuída, integrada ao Apache Spark, que oferece uma ampla gama de algoritmos e ferramentas para construção, treinamento e avaliação de modelos de machine learning em grandes conjuntos de dados.

Sua integração com o Spark permite aproveitar a capacidade de processamento em memória e distribuído do Spark para treinar modelos em paralelo e lidar com conjuntos de dados massivos de forma eficiente.

Machine Learning com Apache Spark (MLlib)

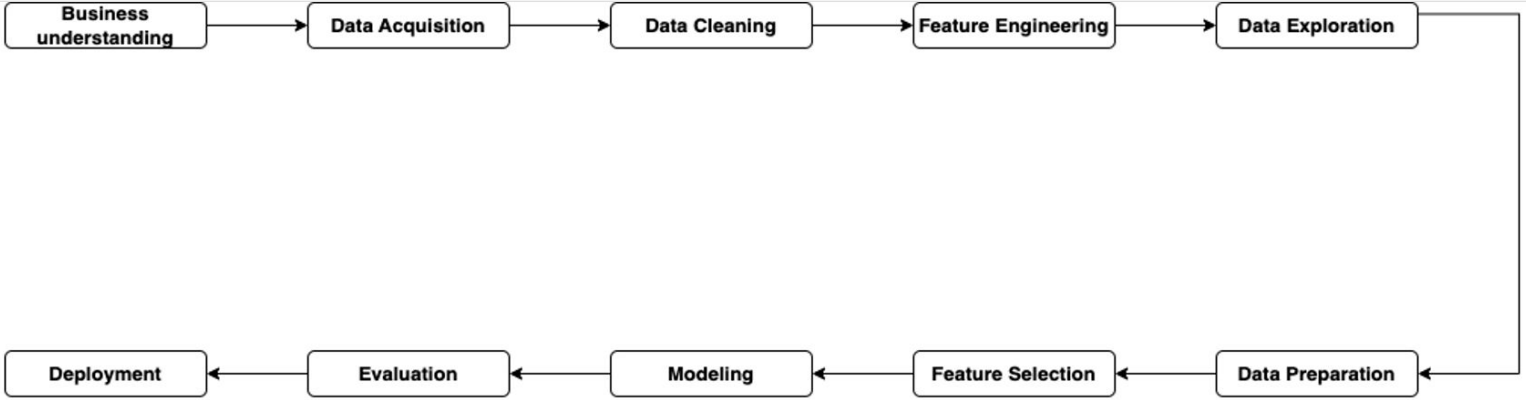
Aplicações práticas

O MLlib é aplicado em uma variedade de problemas de machine learning, incluindo classificação, regressão, clustering, recomendação e muito mais.

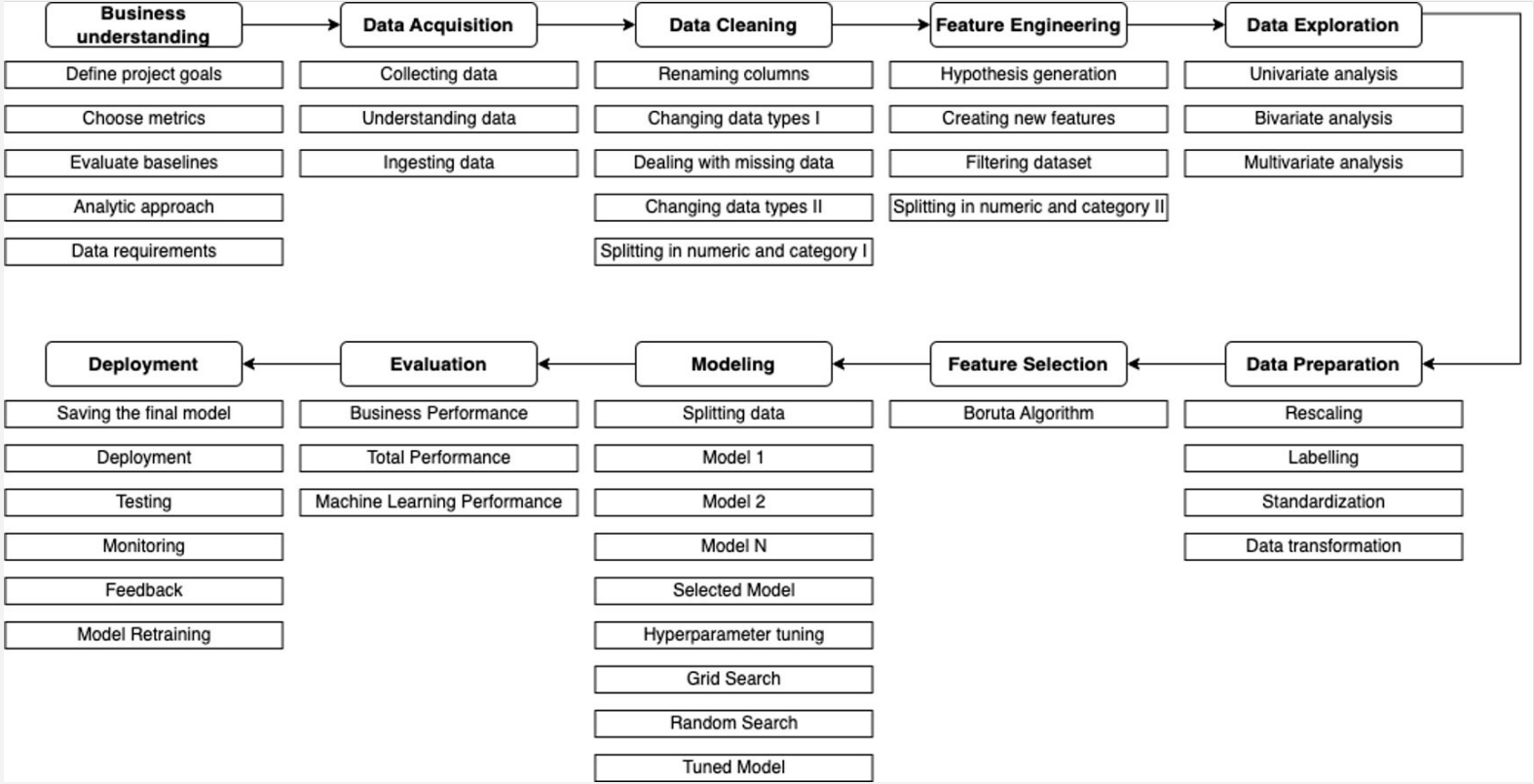
Exemplos práticos incluem a previsão de vendas com base em dados históricos, a detecção de fraudes em transações financeiras, a segmentação de clientes para campanhas de marketing direcionadas e a filtragem colaborativa para recomendação de produtos ou conteúdo.

A flexibilidade e escalabilidade do MLlib o tornam uma escolha popular para projetos de machine learning em larga escala.

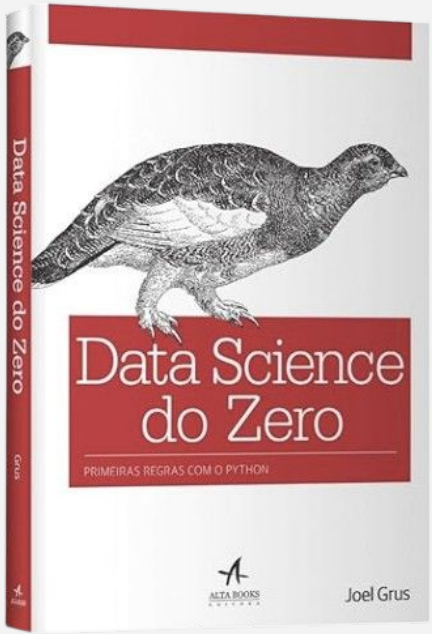
Projeto E2E com Apache Spark

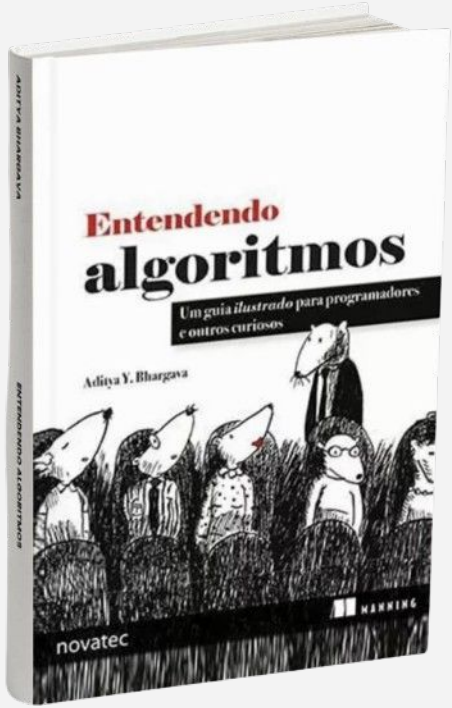


Projeto E2E com Apache Spark



Sugestões de leitura





Material Prático

Minicurso - Data Science.ipynb

Data Science E2E (Regressão).ipynb