



**Fecomércio
Sesc**

Big Data

Prof. Marco Mialaret

Maio

2024



Big Data



Onde me encontrar:

<https://www.linkedin.com/in/marco-mialaret-junior/>

e

<https://github.com/MatmJr>

Trabalhando com o PySpark

Big Data

PySpark é uma API em Python para executar o Spark e foi lançado para oferecer suporte à colaboração entre Apache Spark e Python. O PySpark também oferece suporte à interface do Apache Spark com conjuntos de dados distribuídos resilientes (RDDs) na linguagem de programação Python.

Big Data

Usaremos o conjunto de dados relacionado a campanhas de marketing direto (chamadas telefônicas) de uma instituição bancária. O objetivo da classificação é prever se o cliente irá realizar (Sim/Não) um depósito a prazo. O conjunto de dados pode ser baixado no link

<https://archive.ics.uci.edu/dataset/222/bank+marketing>

Big Data

```
import gdown
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession

# URL do Google Drive
url = 'https://drive.google.com/uc?id=19M95VTZyZ5HzBxlIALMr1Qyo5TbMc1Jy'

# Baixando o arquivo
output = 'file.csv'
gdown.download(url, output, quiet=False)

# Inicializando o SparkSession
spark = SparkSession.builder.appName('ml-bank').getOrCreate()

# Lendo o arquivo CSV baixado
df = spark.read.csv(output, header=True, inferSchema=True)
df.printSchema()
```

Big Data

```
|-- age: integer (nullable = true)
|-- job: string (nullable = true)
|-- marital: string (nullable = true)
|-- education: string (nullable = true)
|-- default: string (nullable = true)
|-- balance: integer (nullable = true)
|-- housing: string (nullable = true)
|-- loan: string (nullable = true)
|-- contact: string (nullable = true)
|-- day: integer (nullable = true)
|-- month: string (nullable = true)
|-- duration: integer (nullable = true)
|-- campaign: integer (nullable = true)
|-- pdays: integer (nullable = true)
|-- previous: integer (nullable = true)
|-- poutcome: string (nullable = true)
|-- deposit: string (nullable = true)
```

Big Data

Para visualizar os dados

`df.show(5)`

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes

Big Data

Mas podemos converter em um DataFrame do PySpark para um DataFrame do Pandas. O método `df.take(5)` obtém as primeiras 5 linhas do DataFrame do PySpark `df`, e a função `pd.DataFrame(..., columns=df.columns)` converte essas 5 linhas em um DataFrame do Pandas, preservando os nomes das colunas originais do DataFrame do PySpark.

Big Data

Para visualizar os dados

```
pd.DataFrame(df.take(5), columns=df.columns)
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes

Big Data

O código a seguir agrupa os dados pela coluna 'deposit', contabilizando quantos clientes realizaram (Yes) e quantos não realizaram ('No') um depósito a prazo. Em seguida, converte o resultado para um DataFrame do Pandas, facilitando a manipulação e visualização dos dados. O resultado final é uma tabela que mostra a contagem de registros para cada valor na coluna 'deposit'.

Big Data

```
df.groupby('deposit').count().toPandas()
```

	deposit	count
0	no	5873
1	yes	5289

Big Data

Primeiro, vamos uma lista chamada ``numeric_features``, que contém os nomes das colunas do DataFrame ``df`` que têm o tipo de dado inteiro. Em seguida, selecionamos essas colunas do DataFrame e aplicamos o método ``describe()`` para obter estatísticas descritivas, como média, desvio padrão, valores mínimo e máximo. Por fim, convertemos o resultado para um DataFrame do Pandas e transpomos a tabela para facilitar a visualização dessas estatísticas.

Big Data

```
numeric_features = [t[0] for t in df.dtypes if t[1] == 'int']  
df.select(numeric_features).describe().toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
age	11162	41.231947679627304	11.913369192215518	18	95
balance	11162	1528.5385235620856	3225.413325946149	-6847	81204
duration	11162	371.99381831213043	347.12838571630687	2	3881
campaign	11162	2.508421429851281	2.7220771816614824	1	63
pdays	11162	51.33040673714388	108.75828197197717	-1	854
previous	11162	0.8325568894463358	2.292007218670508	0	58

Big Data

```
numeric_features = [t[0] for t in df.dtypes if t[1] == 'int' or t[1] == 'double']  
numeric_df = df.select(numeric_features).toPandas()  
numeric_df.corr()
```

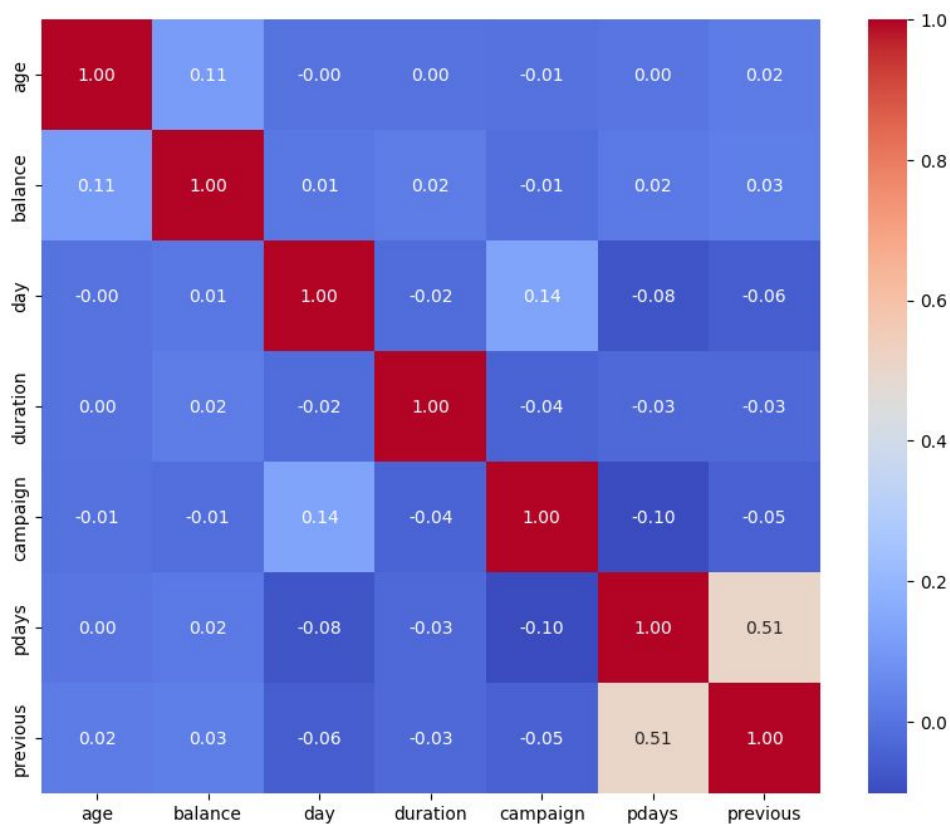
	age	balance	day	duration	campaign	pdays	previous
age	1.000000	0.112300	-0.000762	0.000189	-0.005278	0.002774	0.020169
balance	0.112300	1.000000	0.010467	0.022436	-0.013894	0.017411	0.030805
day	-0.000762	0.010467	1.000000	-0.018511	0.137007	-0.077232	-0.058981
duration	0.000189	0.022436	-0.018511	1.000000	-0.041557	-0.027392	-0.026716
campaign	-0.005278	-0.013894	0.137007	-0.041557	1.000000	-0.102726	-0.049699
pdays	0.002774	0.017411	-0.077232	-0.027392	-0.102726	1.000000	0.507272
previous	0.020169	0.030805	-0.058981	-0.026716	-0.049699	0.507272	1.000000

Big Data

A principal conclusão da matriz de correlação apresentada é que a maioria das variáveis numéricas não possuem fortes correlações entre si. No entanto, há uma correlação moderadamente forte entre as variáveis `pdays` e `previous` (0.507272). Isso indica que há uma relação significativa entre o número de dias desde que um cliente foi contatado por uma campanha anterior (`pdays`) e o número de contatos realizados antes dessa campanha (`previous`).

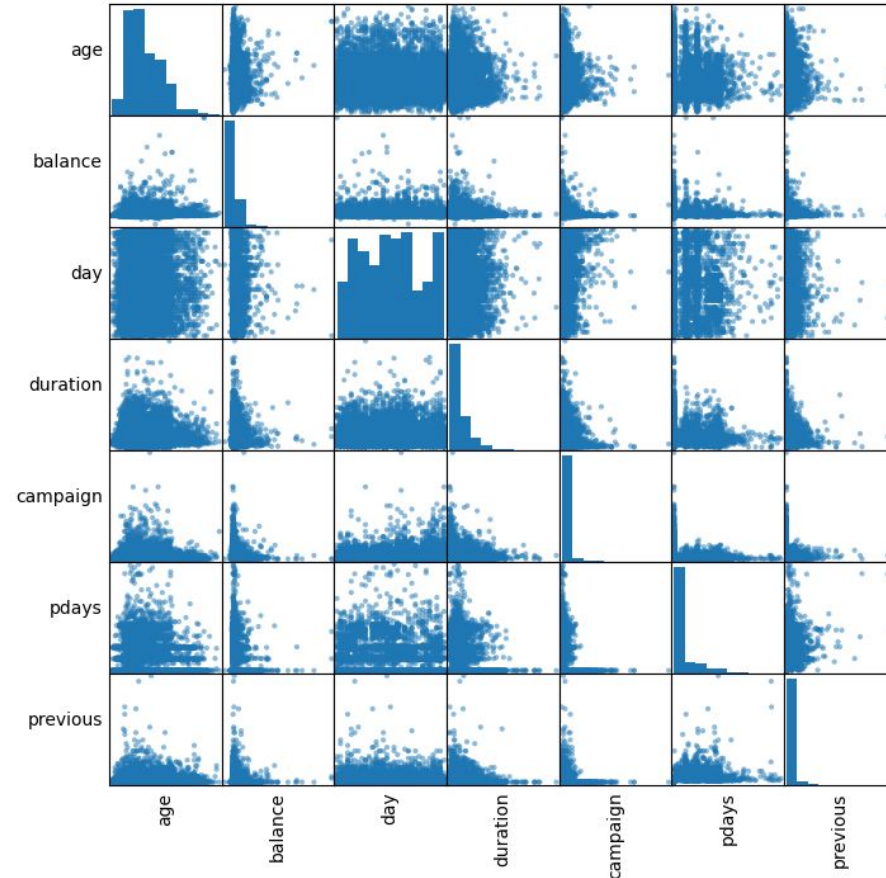
Big Data

```
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_df.corr(), annot=True,
            cmap='coolwarm', fmt='.2f')
plt.show()
```



Big Data

```
numeric_data = df.select(numeric_features).toPandas()  
axs = pd.plotting.scatter_matrix(numeric_data, figsize=(8, 8))
```



Big Data

Vamos realizar a indexação e codificação de colunas categóricas de um DataFrame do PySpark. Primeiro, vamos importar as bibliotecas necessárias e define uma lista de colunas categóricas. Para cada coluna, vamos criar um `StringIndexer` que converte os valores categóricos em índices numéricos. Em seguida, usa um `OneHotEncoderEstimator` para transformar esses índices em vetores binários one-hot.

Big Data



Esses transformadores são armazenados em uma lista chamada `stages`, que será usada em um pipeline para aplicar essas transformações aos dados. O resultado final é um DataFrame com as colunas categóricas convertidas em vetores binários, facilitando a análise e o uso em algoritmos de aprendizado de máquina.

Big Data

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
```

```
# Definir colunas categóricas
```

```
categoricalColumns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'poutcome']
```

```
# Lista para armazenar estágios do pipeline
```

```
stages = []
```

Big Data

```
# Indexar e codificar colunas categóricas
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + 'Index')
    encoder = OneHotEncoder(inputCol=stringIndexer.getOutputCol(), outputCol=categoricalCol + 'classVec')
    stages += [stringIndexer, encoder]

# Indexar a coluna de etiquetas
label_stringIdx = StringIndexer(inputCol='deposit', outputCol='label')
stages += [label_stringIdx]
```

Big Data

Usamos o StringIndexer novamente para codificar nossas etiquetas em índices de etiquetas. Em seguida, usamos o VectorAssembler para combinar todas as colunas de características em uma única coluna de vetor.

Big Data

Colunas numéricas

```
numericCols = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']
```

Montar o vetor de características

```
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
```

```
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
```

```
stages += [assembler]
```

Big Data

Pipeline

Usamos o Pipeline para encadear múltiplos Transformadores e Estimadores juntos para especificar nosso fluxo de trabalho de aprendizado de máquina. Os estágios de um Pipeline são especificados como um array ordenado.

Big Data

```
# Criar e ajustar o pipeline
pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
```

```
# Selecionar colunas de interesse
selectedCols = ['label', 'features'] + [col for col in df.columns if col not in ['label']]
df = df.select(selectedCols)
df.printSchema()
```


Big Data

```
pd.DataFrame(df.take(5), columns=df.columns)
```

	label	features	age	job	marital	education	default	balance	housing	loan	contact	duration	campaign	pdays	previous	poutcome	deposit
0	1.0	(0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	59	admin.	married	secondary	no	2343	yes	no	unknown	1042	1	-1	0	unknown	yes
1	1.0	(0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	56	admin.	married	secondary	no	45	no	no	unknown	1467	1	-1	0	unknown	yes
2	1.0	(0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	41	technician	married	secondary	no	1270	yes	no	unknown	1389	1	-1	0	unknown	yes
3	1.0	(0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...	55	services	married	secondary	no	2476	yes	no	unknown	579	1	-1	0	unknown	yes
4	1.0	(0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	54	admin.	married	tertiary	no	184	no	no	unknown	673	2	-1	0	unknown	yes

Big Data



Separando o conjunto em treino e teste.

```
train, test = df.randomSplit([0.7, 0.3], seed = 2018)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 7855
Test Dataset Count: 3307
```

Big Data

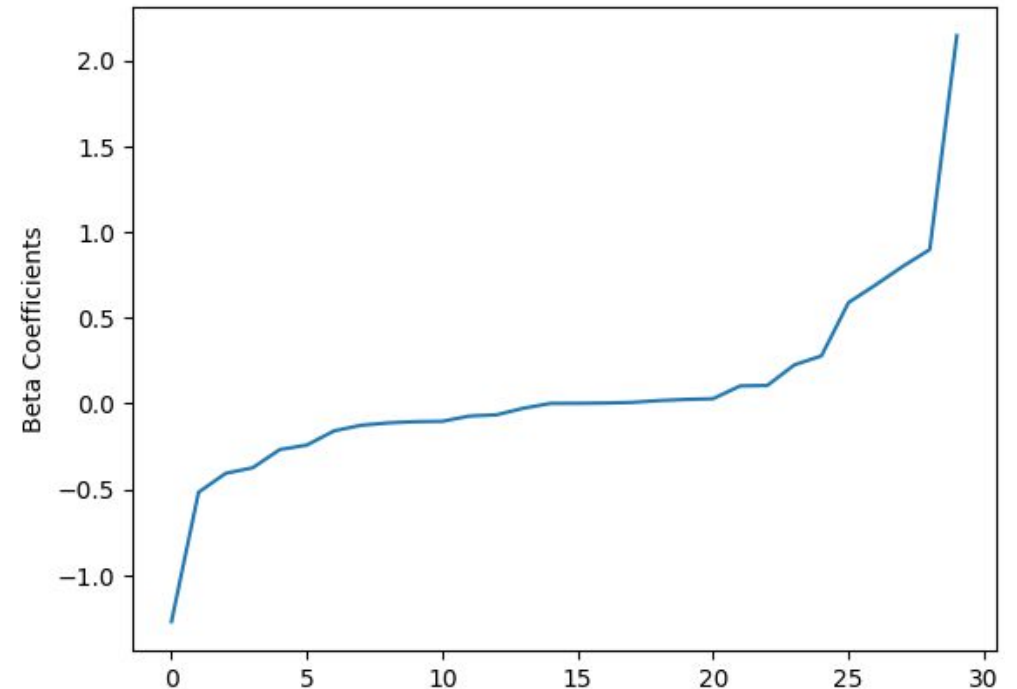


Criando um Modelo de Regressão Logística

```
from pyspark.ml.classification import  
LogisticRegression  
# Treinar o modelo de Regressão Logística  
lr = LogisticRegression(featuresCol='features',  
labelCol='label', maxIter=10)  
lrModel = lr.fit(train)
```

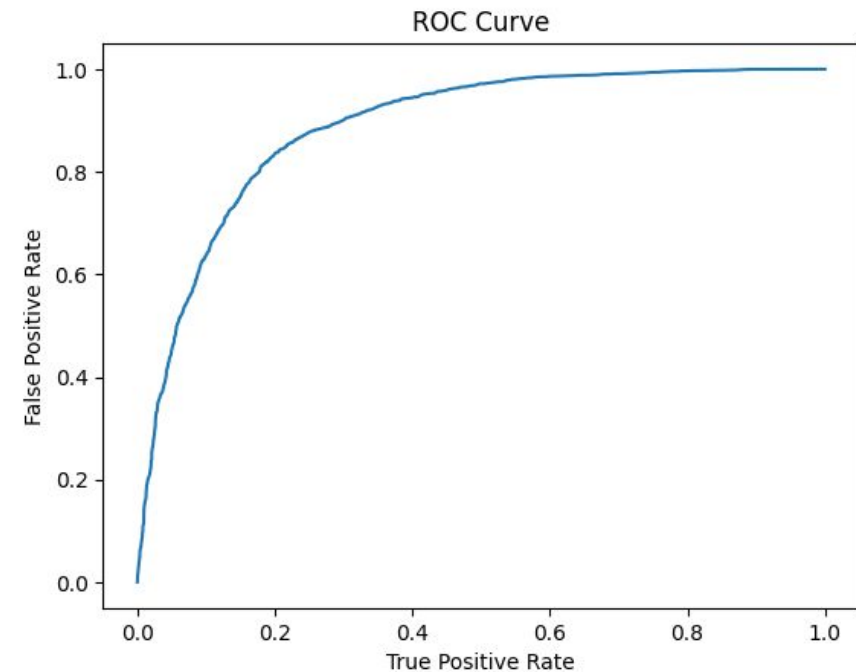
Big Data

```
import matplotlib.pyplot as plt
import numpy as np
beta = np.sort(lrModel.coefficients)
plt.plot(beta)
plt.ylabel('Beta Coefficients')
plt.show()
```



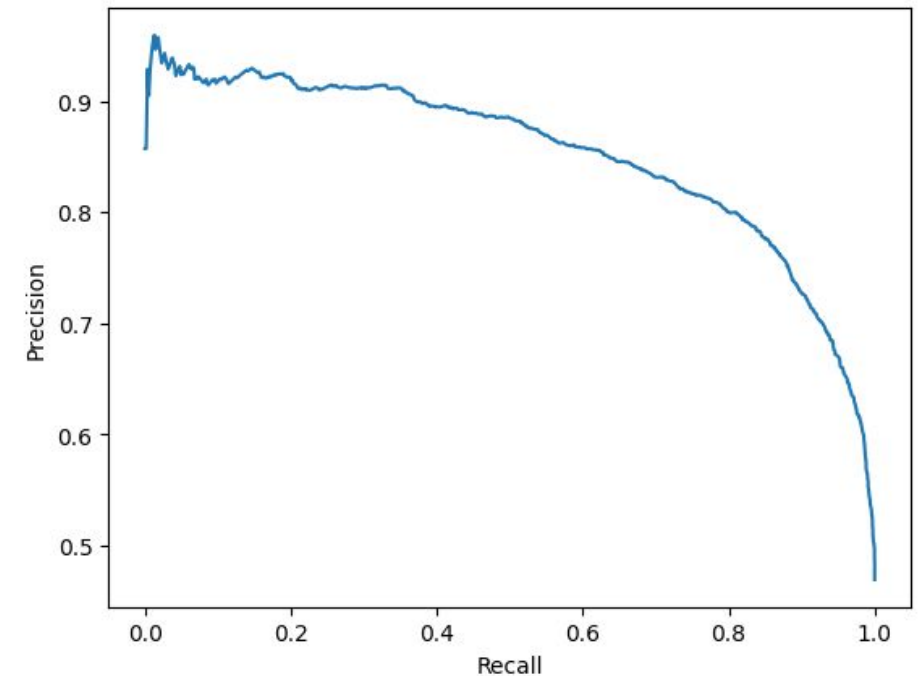
Big Data

```
trainingSummary = lrModel.summary
roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set areaUnderROC: ' +
      str(trainingSummary.areaUnderROC))
```



Big Data

```
pr = trainingSummary.pr.toPandas()
plt.plot(pr['recall'],pr['precision'])
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()
```



Big Data

- Beta Coefficients: Indicam a influência das variáveis independentes no modelo.
- ROC Curve: Avalia a capacidade do modelo de distinguir entre classes.
- Precision x Recall: Avalia o equilíbrio entre precisão e recall, especialmente útil para classes desbalanceadas.

Big Data

```
predictions = lrModel.transform(test)
predictions.select('age', 'job', 'label', 'rawPrediction',
'prediction', 'probability', 'deposit').show()
```

age	job	label	rawPrediction	prediction	probability	deposit
33	management	0.0	[1.93084854518128...	0.0	[0.87334331124860...	no
49	management	0.0	[1.92783695472097...	0.0	[0.87300981013433...	no
52	management	0.0	[-0.7737627890181...	1.0	[0.31566570209267...	no
53	management	0.0	[0.94708137344074...	0.0	[0.72052784002179...	no
58	management	0.0	[2.44657764692457...	0.0	[0.92031082026376...	no

Uma outra forma de usar o Pyspark

Big Data

O PySpark também pode ser utilizado no Colab. Vamos começar a explorar esse novo mundo juntos:

Crie um notebook novo

Big Data

Use a primeira célula para instalar a biblioteca do PySpark

```
!pip install pyspark requests
```

Big Data

Vamos iniciar uma sessão Spark

```
from pyspark.sql import SparkSession
```

```
spark = (SparkSession.builder  
        .appName("PySpark App")  
        .getOrCreate())
```

Big Data

Carregando o primeiro dataset

```
import requests
```

```
response=requests.get(  
"https://ddragon.leagueoflegends.com/cdn/12.17.1/data/pt_BR/cham  
pion.json")
```

```
champions=response.json().get("data")  
champions.keys()
```

Big Data

Limpeza dos dados

Antes de começarmos de fato com a análise, é necessário fazermos uma limpeza prévia nos dados. Vamos pegar apenas os que nos interessa, e remover os dicionários dentro de dicionários, deixando um único dicionário para cada campeão com os dados necessários.

Big Data

```
champions=[{'name': value['name'], 'title': value['title'], **value['info'],  
**value['stats']} for key, value in champions.items()]  
champions[2]
```

Big Data



Criando o DataFrame

Agora que os dados dos campeões estão limpos, podemos criar nosso DataFrame usando o Spark.

No entanto, o Spark é bastante específico quanto ao tipo de objeto que aceitamos para criar um DataFrame. Atualmente, nosso objeto "champions" é uma lista de dicionários, que não é compatível com o Spark.

Big Data

Mas existe uma solução! A biblioteca Pandas é muito mais flexível quando se trata de criar um DataFrame. Podemos criar um DataFrame do Pandas a partir do nosso objeto "champions" atual e, em seguida, usar esse DataFrame do Pandas para criar um DataFrame do Spark.

Big Data

```
import pandas as pd

df = spark.createDataFrame(pd.DataFrame(champions))

df.select("name", "title").show(5, False)
```

Big Data



Concatenação de colunas

Para facilitar a visualização dos dados, vamos criar uma nova coluna chamada ``full_name`` que concatena as colunas ``name`` e ``title``. Utilizaremos o método ``withColumn`` para isso. Esse método recebe dois parâmetros: o nome da nova coluna e os dados para populá-la. Usaremos a função ``concat`` para juntar as colunas ``name`` e ``title``, e a função ``lit`` para adicionar uma vírgula e um espaço entre elas.

Big Data

```
from pyspark.sql import functions as F
```

```
df = df.withColumn("full_name", F.concat(df.name, F.lit(", "), df.title))  
df.select("full_name").show(5, False)
```

Big Data

Quem são os campeões mais poderosos de League of Legends?

```
base_columns = ["attackdamage", "armor", "hp", "mp"]
```

```
(df.orderBy(*base_columns, ascending=False)  
.select("full_name", *base_columns)  
.show(5, False)  
)
```

Big Data

Quem são os campeões mais poderosos no nível 10 de League of Legends?

level = 10

```
df2 = df.withColumns({  
    "attackdamage": df.attackdamage+df.attackdamageperlevel*level,  
    "armor": df.armor+df.armorperlevel*level,  
    "hp": df.hp+df.hpperlevel*level,  
    "mp": df.mp+df.mpperlevel*level  
})
```


Big Data

Estatísticas dos níveis de poder

```
(df2.agg({  
    "attackdamage": "mean",  
    "hp": "max",  
    "mp": "max",  
    "armor": "min"  
})  
    .show()  
)
```

Dúvidas?



Marco Mialaret, MSc

Telefone:

81 98160 7018

E-mail:

marcomialaret@gmail.com

