# Machine Learning para prever o preço de fechamento da ação

In [0]:
```
import os
from pyspark.sql import SparkSession
import json
from pyspark.ml.regression import LinearRegression
from pyspark.mllib.regression import LinearRegressionWithSGD
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.window import Window
from pyspark.sql.functions import *
```

## OBS.:

O motivo abaixo para copiar o arquivo é que o Spark (no contexto da Databricks) não consegue acessar diretamente os caminhos do DBFS de maneira convencional, como faria com um arquivo local no sistema de arquivos. Ao copiar o arquivo para o sistema de arquivos local, você garante que o Spark consiga acessar o arquivo corretamente.

In [0]:
```
# Copiar o arquivo do DBFS para o sistema de arquivos local
local_path = "/tmp/estudos_448118_b6a96208faf3.json"
dbutils.fs.cp("dbfs:/tmp/estudos_448118_b6a96208faf3.json", "file:" + local_path)

# Configurar credenciais no Python
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = local_path

# Criar sessão Spark
spark = SparkSession.builder.appName("SparkML_GCS").getOrCreate()

# Configurar credenciais no Spark
spark.conf.set("fs.gs.auth.service.account.enable", "true")
spark.conf.set("google.cloud.auth.service.account.json.keyfile", local_path)

# Caminho do arquivo CSV no GCS
bucket_path = "gs://dados_input/api-acoes/fechamento_gerdau.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# Carregar o CSV no Spark
df_spark = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(bucket_path)

df_spark.show(10)
```

```
+----------+-----+-----+-----+-----+--------+
| timestamp| open| high|  low|close|  volume|
+----------+-----+-----+-----+-----+--------+
|2025-02-25|16.25|16.57|16.13|16.32| 8154600|
|2025-02-24|16.17| 16.5|15.93|16.27|14514900|
|2025-02-21| 16.6|16.77|16.09|16.22|18343500|
|2025-02-20|17.43|17.58|16.43|16.43|26719700|
```

```
|2025-02-19|17.32| 17.5|17.17|17.36| 8894400|
|2025-02-18|17.46|17.64|17.27|17.44|11477300|
|2025-02-17|17.55|17.71|17.36|17.37| 8584700|
|2025-02-14|17.58|17.72|17.34|17.58| 6849900|
|2025-02-13| 17.3| 17.5|17.13|17.45| 5875700|
|2025-02-12|17.54|17.66|17.22|17.37|14205900|
+----------+-----+-----+-----+-----+--------+
only showing top 10 rows
```

In [0]:

```
df_spark.printSchema()
```

```
root
 |-- timestamp: date (nullable = true)
 |-- open: double (nullable = true)
 |-- high: double (nullable = true)
 |-- low: double (nullable = true)
 |-- close: double (nullable = true)
 |-- volume: integer (nullable = true)
```

In [0]:

```
df_spark = df_spark.withColumnRenamed("timestamp", "date")
df_spark.show(5)
```

```
+----------+-----+-----+-----+-----+--------+
|      date| open| high|  low|close|  volume|
+----------+-----+-----+-----+-----+--------+
|2025-02-25|16.25|16.57|16.13|16.32| 8154600|
|2025-02-24|16.17| 16.5|15.93|16.27|14514900|
|2025-02-21| 16.6|16.77|16.09|16.22|18343500|
|2025-02-20|17.43|17.58|16.43|16.43|26719700|
|2025-02-19|17.32| 17.5|17.17|17.36| 8894400|
+----------+-----+-----+-----+-----+--------+
only showing top 5 rows
```

In [0]:

```
df_spark = df_spark.withColumn("MMA-10d",round(avg("close").over(Window.orderBy(col(
df_spark.show(5)
```

```
+----------+-----+-----+-----+-----+--------+-------+
|      date| open| high|  low|close|  volume|MMA-10d|
+----------+-----+-----+-----+-----+--------+-------+
|2025-02-25|16.25|16.57|16.13|16.32| 8154600|  16.32|
|2025-02-24|16.17| 16.5|15.93|16.27|14514900|   16.3|
|2025-02-21| 16.6|16.77|16.09|16.22|18343500|  16.27|
|2025-02-20|17.43|17.58|16.43|16.43|26719700|  16.31|
|2025-02-19|17.32| 17.5|17.17|17.36| 8894400|  16.52|
+----------+-----+-----+-----+-----+--------+-------+
only showing top 5 rows
```

In [0]:

```
# Dividir os dados cronologicamente no Pandas
df_pandas = df_spark.toPandas()

train_size = int(len(df_pandas) * 0.2)
train_data_pandas = df_pandas[train_size:]  # 80% dos dados
display(train_data_pandas)

test_data_pandas = df_pandas[:train_size]   # 20% dos dados mais recentes
display(test_data_pandas)
```

| date | open | high | low | close | volume | MMA-10d |
|---|---|---|---|---|---|---|
| 2025-01-28 | 17.66 | 17.88 | 17.55 | 17.67 | 7886500 | 17.3 |
| 2025-01-27 | 17.53 | 17.78 | 17.46 | 17.75 | 8151600 | 17.31 |
| 2025-01-24 | 17.47 | 17.63 | 17.39 | 17.56 | 8424500 | 17.4 |
| 2025-01-23 | 17.62 | 17.69 | 17.36 | 17.46 | 8861100 | 17.44 |
| 2025-01-22 | 17.99 | 17.99 | 17.47 | 17.6 | 11092400 | 17.51 |
| 2025-01-21 | 17.6 | 17.93 | 17.5 | 17.91 | 10082400 | 17.59 |
| 2025-01-20 | 17.29 | 17.61 | 17.11 | 17.53 | 6081600 | 17.59 |
| 2025-01-17 | 17.4 | 17.5 | 17.17 | 17.37 | 12824800 | 17.61 |
| 2025-01-16 | 17.61 | 17.64 | 16.99 | 17.26 | 13266400 | 17.56 |
| 2025-01-15 | 17.22 | 17.68 | 17.22 | 17.65 | 11043700 | 17.58 |

| date | open | high | low | close | volume | MMA-10d |
|---|---|---|---|---|---|---|
| 2025-02-25 | 16.25 | 16.57 | 16.13 | 16.32 | 8154600 | 16.32 |
| 2025-02-24 | 16.17 | 16.5 | 15.93 | 16.27 | 14514900 | 16.3 |
| 2025-02-21 | 16.6 | 16.77 | 16.09 | 16.22 | 18343500 | 16.27 |
| 2025-02-20 | 17.43 | 17.58 | 16.43 | 16.43 | 26719700 | 16.31 |
| 2025-02-19 | 17.32 | 17.5 | 17.17 | 17.36 | 8894400 | 16.52 |
| 2025-02-18 | 17.46 | 17.64 | 17.27 | 17.44 | 11477300 | 16.67 |
| 2025-02-17 | 17.55 | 17.71 | 17.36 | 17.37 | 8584700 | 16.77 |
| 2025-02-14 | 17.58 | 17.72 | 17.34 | 17.58 | 6849900 | 16.87 |
| 2025-02-13 | 17.3 | 17.5 | 17.13 | 17.45 | 5875700 | 16.94 |
| 2025-02-12 | 17.54 | 17.66 | 17.22 | 17.37 | 14205900 | 16.98 |

In [0]:
```
# Converter de volta para Spark DataFrame
train_data_spark = spark.createDataFrame(train_data_pandas)
test_data_spark = spark.createDataFrame(test_data_pandas)
```

In [0]:
```
# Definir features (X) e target (y)
features = ['open', 'high', 'low', 'volume', 'MMA-10d']
assembler = VectorAssembler(inputCols=features, outputCol='features')
```

In [0]:
```
# Aplicar o VectorAssembler aos dados de treino e teste
train_data_spark = assembler.transform(train_data_spark)
test_data_spark = assembler.transform(test_data_spark)
```

In [0]:
```
# Criar e treinar o modelo (Linear Regression com Spark ML)
lr = LinearRegression(featuresCol='features', labelCol='close')
model_lr = lr.fit(train_data_spark)
```

In [0]:
```
# Fazer previsões
predictions_lr = model_lr.transform(test_data_spark)
```

```
# Arredondar a coluna de previsão
predictions_lr = predictions_lr.withColumn("prediction", round("prediction", 2))

# Exibir as previsões
display(predictions_lr, 10)
```

| date | open | high | low | close | volume | MMA-10d | features | prediction |
|---|---|---|---|---|---|---|---|---|
| 2025-02-25 | 16.25 | 16.57 | 16.13 | 16.32 | 8154600 | 16.32 | Map(vectorType -> dense, length -> 5, values -> List(16.25, 16.57, 16.13, 8154600.0, 16.32)) | 16.38 |
| 2025-02-24 | 16.17 | 16.5 | 15.93 | 16.27 | 14514900 | 16.3 | Map(vectorType -> dense, length -> 5, values -> List(16.17, 16.5, 15.93, 1.45149E7, 16.3)) | 16.23 |
| 2025-02-21 | 16.6 | 16.77 | 16.09 | 16.22 | 18343500 | 16.27 | Map(vectorType -> dense, length -> 5, values -> List(16.6, 16.77, 16.09, 1.83435E7, 16.27)) | 16.29 |
| 2025- | 17.43 | 17.58 | 16.43 | 16.43 | 26719700 | 16.31 | Map(vectorType -> dense, length -> 5, values -> | 16.67 |

## Utilizando algumas métricas para avaliar nosso modelo

**(RMSE) Root Mean Squared Error - Quanto mais próximo de zero, melhor.**

- Um RMSE pequeno indica que as previsões estão bem próximas dos valores reais.
- Isso sugere que o modelo está fazendo previsões bastante precisas.

In [0]:
```
# Avaliar modelo (Spark ML)
evaluator = RegressionEvaluator(labelCol='close', predictionCol='prediction', metric
rmse_lr = evaluator.evaluate(predictions_lr)
print(f'RMSE: {rmse_lr:.2f}')
```

```
RMSE: 0.10
```

- Como o preço médio de fechamento está em torno de 18-20, um erro de 0.10 representa um erro percentual muito pequeno (cerca de 0.5% do valor médio).

**$R^2$ (R-quadrado), quanto mais próximo de 1, melhor!**

O $R^2$ é uma métrica usada para avaliar a qualidade de um modelo de regressão. Ele indica a proporção da variabilidade dos dados que é explicada pelo modelo.

- $R^2$ = 1: O modelo explica 100% da variabilidade dos dados. Ele é perfeito.
- $R^2$ = 0: O modelo não consegue explicar nenhuma variabilidade dos dados, ou seja, o modelo não é melhor do que simplesmente usar a média dos valores reais.
- $R^2$ negativo: Significa que o modelo está se saindo pior do que uma simples média dos dados, o que indica que ele está fazendo previsões ruins.

In [0]:
```
r2_evaluator = RegressionEvaluator(labelCol='close', predictionCol='prediction', met
r2 = r2_evaluator.evaluate(predictions_lr)
print(f'R²: {r2:.2f}')
```

```
R²: 0.96
```

**Erro percentual**

- Calcula a relaçao da predição com relação ao valor real de fechamento.
- Quanto mais baixo, melhor.

In [0]:

```
# Calcular erro percentual para cada linha
predictions_lr = predictions_lr.withColumn('erro_percentual', round(abs(col('predict
predictions_lr.select('date', 'close', 'prediction', 'erro_percentual').show(10)

# Calcular a média do erro percentual
avg_erro_percentual = predictions_lr.agg({'erro_percentual': 'avg'}).collect()[0][0]
print(f'Média do Erro Percentual: {avg_erro_percentual:.2f}%')
```

```
+----------+-----+----------+---------------+
|      date|close|prediction|erro_percentual|
+----------+-----+----------+---------------+
|2025-02-25|16.32|     16.38|           0.37|
|2025-02-24|16.27|     16.23|           0.25|
|2025-02-21|16.22|     16.29|           0.43|
|2025-02-20|16.43|     16.67|           1.46|
|2025-02-19|17.36|      17.3|           0.35|
|2025-02-18|17.44|     17.41|           0.17|
|2025-02-17|17.37|     17.48|           0.63|
|2025-02-14|17.58|     17.45|           0.74|
|2025-02-13|17.45|     17.28|           0.97|
|2025-02-12|17.37|     17.35|           0.12|
+----------+-----+----------+---------------+
only showing top 10 rows

Média do Erro Percentual: 0.43%
```

# Considerações finais sobre o modelo

As métricas RMSE (0,10), $R^2$ (0,96) e o erro percentual (0,43%) indicam que o modelo apresenta um bom desempenho na previsão do preço de fechamento (close). O valor de $R^2$ sugere uma forte correlação entre as previsões e os valores reais, enquanto o RMSE e o erro percentual indicam que as previsões estão próximas dos valores reais, com um pequeno desvio.

É recomendável realizar testes com mais variáveis (features), como as máximas e mínimas semanais, além de outros indicadores técnicos. Esses testes podem fornecer uma visão mais completa do comportamento do modelo e possibilitar melhorias nas métricas de avaliação.

Esses ajustes ajudam a tornar mais claro o impacto das métricas e reforçam a ideia de que adicionar mais variáveis pode melhorar o modelo.