Construir um conjunto de modelos preditivos, para prever o CCS (Concrete Compressive Strength) do concreto a partir das variáveis de entrada (ingredientes) oferecidos para produção do concreto. Automatizar o processo para futuros experimentos da equipe, e escolher o modelo mais eficiente para a produção.

In [1]:
```
# Findspark
import findspark
findspark.init()
```

In [2]:
```
# Imports
import pyspark
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.stat import Correlation
from pyspark.ml.regression import *
from pyspark.ml.evaluation import *
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

Ambiente Spark

In [3]:
```
# Spark Context
sc = SparkContext(appName = "MLConcreto")
sc.setLogLevel("ERROR")
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/02/10 21:11:18 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
```

In [4]:
```
# Session
spark = SparkSession.builder.getOrCreate()
```

In [5]:
```
spark
```

Out[5]:

SparkSession - in-memory

SparkContext

Spark UI (http://192.168.1.4:4040)

Version

v3.5.0

Master

local[*]

AppName

MLConcreto

Carga de Dados

In [6]:
```
# Carregando os dados
dados = spark.read.csv('dados/dataset.csv', inferSchema = True, header = True)
```

In [7]:
```
dados.count()
```

Out[7]:
```
1030
```

In [8]:
```
dados.show(5)
```

```
+------+-----+------+-----+---------------+---------------+-------------+---+-----+
|cement| slag|flyash|water|superplasticizer|coarseaggregate|fineaggregate|age|csMPa|
+------+-----+------+-----+---------------+---------------+-------------+---+-----+
| 540.0|  0.0|   0.0|162.0|            2.5|         1040.0|        676.0| 28|79.99|
| 540.0|  0.0|   0.0|162.0|            2.5|         1055.0|        676.0| 28|61.89|
| 332.5|142.5|   0.0|228.0|            0.0|          932.0|        594.0|270|40.27|
| 332.5|142.5|   0.0|228.0|            0.0|          932.0|        594.0|365|41.05|
| 198.6|132.4|   0.0|192.0|            0.0|          978.4|        825.5|360| 44.3|
+------+-----+------+-----+---------------+---------------+-------------+---+-----+
only showing top 5 rows
```

In [9]:
```
# Formato do Pandas
dados.limit(10).toPandas()
```

Out[9]:

	cement	slag	flyash	water	superplasticizer	coarseaggregate	fineaggregate	age	csMPa
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30
5	266.0	114.0	0.0	228.0	0.0	932.0	670.0	90	47.03
6	380.0	95.0	0.0	228.0	0.0	932.0	594.0	365	43.70
7	380.0	95.0	0.0	228.0	0.0	932.0	594.0	28	36.45
8	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85
9	475.0	0.0	0.0	228.0	0.0	932.0	594.0	28	39.29

In [10]:
```
# Schema
dados.printSchema()
```

```
root
 |-- cement: double (nullable = true)
 |-- slag: double (nullable = true)
 |-- flyash: double (nullable = true)
 |-- water: double (nullable = true)
 |-- superplasticizer: double (nullable = true)
 |-- coarseaggregate: double (nullable = true)
 |-- fineaggregate: double (nullable = true)
 |-- age: integer (nullable = true)
 |-- csMPa: double (nullable = true)
```

Automação da Preparação de Dados

O MLlib exige que as colunas de entrada do dataframe sejam vetorizadas. Os dados já estão sem valores ausentes ou nulos, logo esse tratamento não será necessário. Criaremos uma função Python que irá automatizar o processo de todas as tarefas necessárias para preparação dos dados, incluindo a vetorização.
A função será um fluxo de trabalho que vai receber as variáveis de entrada, de saída, a necessidade ou não de tratar outliers, bem como a necessidade ou não de padronizar os dados.

In [11]:
```
# Função para preparação dos dados
def func_modulo_prep_dados(df,
                           variaveis_entrada,
                           variavel_saida,
                           tratar_outliers = True,
                           padronizar_dados = True):

    # Vamos gerar um novo dataframe, renomeando a variável de saída exigido pelo Spark.
    novo_df = df.withColumnRenamed(variavel_saida, 'label')

    # Converter a variável alvo para float
    if str(novo_df.schema['label'].dataType) != 'IntegerType':
        novo_df = novo_df.withColumn("label", novo_df["label"].cast(FloatType()))

    # Listas de controle para as variáveis
    variaveis_numericas = []
    variaveis_categoricas = []

    # Havendo variáveis string converte para numérico
    for coluna in variaveis_entrada:

        # Verifica se é string
        if str(novo_df.schema[coluna].dataType) == 'StringType':

            # Define a variável com um sufixo para tratamento mais tarde
            novo_nome_coluna = coluna + "_num"

            # Adicionamos à lista de variáveis categóricas
            variaveis_categoricas.append(novo_nome_coluna)

        else:

            # Se não for string adiciona a lista numerica
            variaveis_numericas.append(coluna)

            # Coloca os dados no dataframe de variáveis indexadas
            df_indexed = novo_df


    # Se o dataframe tiver dados string, aplica indexação
    if len(variaveis_categoricas) != 0:
```

```
    for coluna in novo_df:

        # Para variável string, cria, treina e aplica o indexador
        if str(novo_df.schema[coluna].dataType) == 'StringType':

            # Cria o indexador
            indexer = StringIndexer(inputCol = coluna, outputCol = coluna + "_num")

            # Treina e aplica o indexador
            df_indexed = indexer.fit(novo_df).transform(novo_df)
else:

    # Não havendo categórica, coloca os dados no dataframe de variáveis indexadas
    df_indexed = novo_df



# Tratamento de outliers
if tratar_outliers == True:
    print("\nAplicando o tratamento de outliers...")

    # Dicionário
    d = {}

    # Dicionário de quartis das variáveis do dataframe indexado
    for col in variaveis_numericas:
        d[col] = df_indexed.approxQuantile(col,[0.01, 0.99], 0.25)

    # Aplica a transformação a partir da distribuição de cada variável
    for col in variaveis_numericas:

        # Extração da assimetria dos dados
        skew = df_indexed.agg(skewness(df_indexed[col])).collect()
        skew = skew[0][0]

        # Transformação de log + 1 se a assimetria for positiva
        if skew > 1:
            indexed = df_indexed.withColumn(col, log(when(df[col] < d[col][0], d[col][0])\
            .when(df_indexed[col] > d[col][1], d[col][1])\
            .otherwise(df_indexed[col] ) + 1).alias(col))
            print("\nA variável " + col + " foi tratada para assimetria positiva com skew =", skew)
```

```
        # Transformação exponencial se a assimetria for negativa
        elif skew < -1:
            indexed = df_indexed.withColumn(col, \
            exp(when(df[col] < d[col][0], d[col][0])\
            .when(df_indexed[col] > d[col][1], d[col][1])\
            .otherwise(df_indexed[col] )).alias(col))
            print("\nA variável " + col + " foi tratada para assimetria negativa com skew =", skew)



# Vetorização para Spark

# Lista final de atributos concatenando variáveis
lista_atributos = variaveis_numericas + variaveis_categoricas

# Cria o vetorizador para os atributos
vetorizador = VectorAssembler(inputCols = lista_atributos, outputCol = 'features')

# Aplica o vetorizador ao conjunto de dados
dados_vetorizados = vetorizador.transform(df_indexed).select('features', 'label')



# Padronização dos dados
if padronizar_dados == True:
    print("\nPadronizando o conjunto de dados para o intervalo entre 0 a 1...")

    # Scaler
    scaler = MinMaxScaler(inputCol = "features", outputCol = "scaledFeatures")

    # Padronizador, e globalizando variável para uso fora da funç.
    global scalerModel
    scalerModel = scaler.fit(dados_vetorizados)

    # Padroniza as variáveis para o intervalo [min, max]
    dados_padronizados = scalerModel.transform(dados_vetorizados)

    # Gera os dados finais
    dados_finais = dados_padronizados.select('label', 'scaledFeatures')

    # Renomeia as colunas como requerido pelo Spark
```

```
        dados_finais = dados_finais.withColumnRenamed('scaledFeatures', 'features')

        print("\nProcesso Concluído!")

    # Não havendo necessidade
    else:
        print("\nOs dados não serão padronizados.")
        dados_finais = dados_vetorizados

    return dados_finais
```

Preparação dos Dados

In [12]:
```
# Variáveis de entrada
variaveis_entrada = dados.columns[:-1]
```

In [13]:
```
# Target
variavel_saida = dados.columns[-1]
```

In [14]:
```
# Aplica a função
dados_finais = func_modulo_prep_dados(dados, variaveis_entrada, variavel_saida)
```

```
Aplicando o tratamento de outliers...

A variável age foi tratada para assimetria positiva (direita) com skew = 3.2644145354168086

Padronizando o conjunto de dados para o intervalo entre 0 a 1...

Processo Concluído!
```

In [15]:
```
# Visualiza
dados_finais.show(5, truncate = False)
```

```
+-----+----------------------------------------------------------------------------------------------------
----------------------+
|label|features
|
+-----+----------------------------------------------------------------------------------------------------
----------------------+
|79.99|[1.0,0.0,0.0,0.3210862619808307,0.07763975155279502,0.6947674418604651,0.20572002007024587,0.0741
7582417582418]           |
|61.89|[1.0,0.0,0.0,0.3210862619808307,0.07763975155279502,0.7383720930232558,0.20572002007024587,0.0741
7582417582418]           |
|40.27|[0.526255707762557,0.3964941569282137,0.0,0.8482428115015974,0.0,0.3808139534883721,0.0,0.7390109
89010989]                |
|41.05|[0.526255707762557,0.3964941569282137,0.0,0.8482428115015974,0.0,0.3808139534883721,0.0,1.0]
|
|44.3 |[0.22054794520547943,0.3683917640511965,0.0,0.560702875399361,0.0,0.5156976744186046,0.5807827395
8856,0.9862637362637363]|
+-----+----------------------------------------------------------------------------------------------------
----------------------+
only showing top 5 rows
```

Verificando Correlação

In [16]:
```
# Extrai a correlação com coef. de Pearson
coeficientes_corr = Correlation.corr(dados_finais, 'features', 'pearson').collect()[0][0]
```

In [17]:
```
# Convertendo em array
array_corr = coeficientes_corr.toArray()
```

In [18]:
```
array_corr
```

Out[18]:
```
array([[ 1.        , -0.27521591, -0.39746734, -0.08158675,  0.09238617,
        -0.10934899, -0.22271785,  0.08194602],
       [-0.27521591,  1.        , -0.3235799 ,  0.10725203,  0.04327042,
        -0.28399861, -0.28160267, -0.04424602],
       [-0.39746734, -0.3235799 ,  1.        , -0.25698402,  0.37750315,
        -0.00996083,  0.07910849, -0.15437052],
       [-0.08158675,  0.10725203, -0.25698402,  1.        , -0.65753291,
        -0.1822936 , -0.45066117,  0.27761822],
       [ 0.09238617,  0.04327042,  0.37750315, -0.65753291,  1.        ,
        -0.26599915,  0.22269123, -0.19270003],
       [-0.10934899, -0.28399861, -0.00996083, -0.1822936 , -0.26599915,
         1.        , -0.17848096, -0.00301588],
       [-0.22271785, -0.28160267,  0.07910849, -0.45066117,  0.22269123,
        -0.17848096,  1.        , -0.1560947 ],
       [ 0.08194602, -0.04424602, -0.15437052,  0.27761822, -0.19270003,
        -0.00301588, -0.1560947 ,  1.        ]])
```

In [19]:
```
# Correlação entre os atributos e a variável alvo
for item in array_corr:
    print(item[7])
```

```
0.08194602387182176
-0.044246019304454175
-0.15437051606792915
0.27761822152100296
-0.19270002804347258
-0.0030158803467436645
-0.15609470264758615
1.0
```

A partir da correlação da variável alvo com as variáveis de entrada, podemos perceber que nem todas as variáveis apresentam uma boa correlação com a target. Mesmo assim, conversando com a equipe, nossa escolha será levar todas as variáveis a diante para o modelo, a fim de que possa possivelmente compreender melhor o padrão dos dados.

Machine Learning

Módulo de AutoML

Vamos criar uma função para automatizar o uso de diversos algoritmos, multiplas vezes cada um destes. Nossa função irá criar, treinar e avaliar cada um dos algorítimos com diferentes combinações de hiperparâmetros. E então escolheremos o melhor modelo de cada modelo, e dentre todos os modelos, o que apresentar melhor performance.
A função recebe algoritmo como entrada, verifica o algorítmo e executa o código específico para tal, utiliza validação cruzada, constroi os avaliadores RMSE (erro) e R2 (acurácia), escolhe o melhor modelo, e entra no loop para execução do algorítmo imprimindo as métricas. Então seleciona o melhor modelo de cada algorítmo e faz as previsões, extraindo as métricas de eficiência, e salva no Data Frame de resultados para comparação

In [20]:
```
# Divisão em treino e teste 70/30
dados_treino, dados_teste = dados_finais.randomSplit([0.7,0.3])
```

In [21]:
```
# Lista de algoritmos
regressores = [LinearRegression(),
               DecisionTreeRegressor(),
               RandomForestRegressor(),
               GBTRegressor(),
               IsotonicRegression()]
```

In [41]:
```
# Módulo de Auto ML
def func_modulo_ml(algoritmo_regressao):

    # Obter o tipo do algoritmo e criar a instância do objeto
    def func_tipo_algo(algo_regressao):
        algoritmo = algo_regressao
        tipo_algo = type(algoritmo).__name__
        return tipo_algo

    # Aplica
    tipo_algo = func_tipo_algo(algoritmo_regressao)

    # Para Regressão Linear
    if tipo_algo == "LinearRegression":

        # Primeira versão sem validação cruzada
        modelo = regressor.fit(dados_treino)

        # Métricas do modelo
        print('\033[1m' + "Modelo de Regressão Linear Sem Validação Cruzada:" + '\033[0m')
        print("")

        # Avalia com dados de teste
        resultado_teste = modelo.evaluate(dados_teste)

        # Métricas de erro do modelo com dados de teste
        print("RMSE em Teste: {}".format(resultado_teste.rootMeanSquaredError))
        print("Coeficiente R2 em Teste: {}".format(resultado_teste.r2))
        print("")


        # Segunda versão, usando validação cruzada

        # Grid de hiperparâmetros
        paramGrid = (ParamGridBuilder().addGrid(regressor.regParam, [0.1, 0.01]).build())

        # Cria os avaliadores
        eval_rmse = RegressionEvaluator(metricName = "rmse")
        eval_r2 = RegressionEvaluator(metricName = "r2")

        # Cross Validator
```

```
        crossval = CrossValidator(estimator = regressor,
                                  estimatorParamMaps = paramGrid,
                                  evaluator = eval_rmse,
                                  numFolds = 3)

        print('\033[1m' + "Modelo de Regressão Linear Com Validação Cruzada:" + '\033[0m')
        print("")

        # Treina modelo com validação cruzada
        modelo = crossval.fit(dados_treino)

        # Salva o melhor modelo da versão 2
        global LR_BestModel
        LR_BestModel = modelo.bestModel

        # Previsões com dados de teste
        previsoes = LR_BestModel.transform(dados_teste)

        # Avaliação do melhor modelo
        resultado_teste_rmse = eval_rmse.evaluate(previsoes)
        print('RMSE em Teste:', resultado_teste_rmse)

        resultado_teste_r2 = eval_r2.evaluate(previsoes)
        print('Coeficiente R2 em Teste:', resultado_teste_r2)
        print("")

        # Lista de colunas dataframe de resumo
        columns = ['Regressor', 'Resultado_RMSE', 'Resultado_R2']

        # Formata as métricas e nome do algoritmo
        rmse_str = [str(resultado_teste_rmse)]
        r2_str = [str(resultado_teste_r2)]
        tipo_algo = [tipo_algo]

        # Cria o dataframne de resumo
        df_resultado = spark.createDataFrame(zip(tipo_algo, rmse_str, r2_str), schema = columns)

        # Grava os resultados no dataframe
        df_resultado = df_resultado.withColumn('Resultado_RMSE', df_resultado.Resultado_RMSE.substr(0, 5))
        df_resultado = df_resultado.withColumn('Resultado_R2', df_resultado.Resultado_R2.substr(0, 5))

        return df_resultado
```

```
else:

    # Para Decision Tree
    if tipo_algo in("DecisionTreeRegressor"):
        paramGrid = (ParamGridBuilder().addGrid(regressor.maxBins, [10, 20, 40]).build())

    # Para Random Forest
    if tipo_algo in("RandomForestRegressor"):
        paramGrid = (ParamGridBuilder().addGrid(regressor.numTrees, [5, 10, 20]).build())

    # Para GBT
    if tipo_algo in("GBTRegressor"):
        paramGrid = (ParamGridBuilder() \
                     .addGrid(regressor.maxBins, [10, 20]) \
                     .addGrid(regressor.maxIter, [10, 15])
                     .build())

    # Para Isotonic
    if tipo_algo in("IsotonicRegression"):
        paramGrid = (ParamGridBuilder().addGrid(regressor.isotonic, [True, False]).build())


    # Cria os avaliadores
    eval_rmse = RegressionEvaluator(metricName = "rmse")
    eval_r2 = RegressionEvaluator(metricName = "r2")


    # Prepara o Cross Validator
    crossval = CrossValidator(estimator = regressor,
                              estimatorParamMaps = paramGrid,
                              evaluator = eval_rmse,
                              numFolds = 3)


    # Treina o modelo usando validação cruzada
    modelo = crossval.fit(dados_treino)


    # Extrai o melhor modelo
    BestModel = modelo.bestModel
```

```
# Resumo de cada modelo
# Métricas do modelo
if tipo_algo in("DecisionTreeRegressor"):

    # Variável global
    global DT_BestModel
    DT_BestModel = modelo.bestModel

    # Previsões com dados de teste
    previsoes_DT = DT_BestModel.transform(dados_teste)

    print('\033[1m' + "Modelo Decision Tree Com Validação Cruzada:" + '\033[0m')
    print(" ")

    # Avaliação do modelo
    resultado_teste_rmse = eval_rmse.evaluate(previsoes_DT)
    print('RMSE em Teste:', resultado_teste_rmse)

    resultado_teste_r2 = eval_r2.evaluate(previsoes_DT)
    print('Coeficiente R2 em Teste:', resultado_teste_r2)
    print("")

# Métricas do modelo
if tipo_algo in("RandomForestRegressor"):

    # Variável global
    global RF_BestModel
    RF_BestModel = modelo.bestModel

    # Previsões com dados de teste
    previsoes_RF = RF_BestModel.transform(dados_teste)

    print('\033[1m' + "Modelo RandomForest Com Validação Cruzada:" + '\033[0m')
    print(" ")

    # Avaliação do modelo
    resultado_teste_rmse = eval_rmse.evaluate(previsoes_RF)
    print('RMSE em Teste:', resultado_teste_rmse)

    resultado_teste_r2 = eval_r2.evaluate(previsoes_RF)
    print('Coeficiente R2 em Teste:', resultado_teste_r2)
```

```
        print("")

    # Métricas do modelo
    if tipo_algo in("GBTRegressor"):

        # Variável global
        global GBT_BestModel
        GBT_BestModel = modelo.bestModel

        # Previsões com dados de teste
        previsoes_GBT = GBT_BestModel.transform(dados_teste)

        print('\033[1m' + "Modelo Gradient-Boosted Tree (GBT) Com Validação Cruzada:" + '\033[0m')
        print(" ")

        # Avaliação do modelo
        resultado_teste_rmse = eval_rmse.evaluate(previsoes_GBT)
        print('RMSE em Teste:', resultado_teste_rmse)

        resultado_teste_r2 = eval_r2.evaluate(previsoes_GBT)
        print('Coeficiente R2 em Teste:', resultado_teste_r2)
        print("")

    # Métricas do modelo
    if tipo_algo in("IsotonicRegression"):

        # Variável global
        global ISO_BestModel
        ISO_BestModel = modelo.bestModel

        # Previsões com dados de teste
        previsoes_ISO = ISO_BestModel.transform(dados_teste)

        print('\033[1m' + "Modelo Isotonic Com Validação Cruzada:" + '\033[0m')
        print(" ")

        # Avaliação do modelo
        resultado_teste_rmse = eval_rmse.evaluate(previsoes_ISO)
        print('RMSE em Teste:', resultado_teste_rmse)

        resultado_teste_r2 = eval_r2.evaluate(previsoes_ISO)
        print('Coeficiente R2 em Teste:', resultado_teste_r2)
```

```
        print("")


    # Lista de colunas
    columns = ['Regressor', 'Resultado_RMSE', 'Resultado_R2']


    # Previsões com dados de teste
    previsoes = modelo.transform(dados_teste)


    # Avalia o modelo
    eval_rmse = RegressionEvaluator(metricName = "rmse")
    rmse = eval_rmse.evaluate(previsoes)
    rmse_str = [str(rmse)]

    eval_r2 = RegressionEvaluator(metricName = "r2")
    r2 = eval_r2.evaluate(previsoes)
    r2_str = [str(r2)]

    tipo_algo = [tipo_algo]


    # Cria o dataframe
    df_resultado = spark.createDataFrame(zip(tipo_algo, rmse_str, r2_str), schema = columns)


    # Grava o resultado no dataframe
    df_resultado = df_resultado.withColumn('Resultado_RMSE', df_resultado.Resultado_RMSE.substr(0, 5))
    df_resultado = df_resultado.withColumn('Resultado_R2', df_resultado.Resultado_R2.substr(0, 5))


    return df_resultado
```

In [23]:
```
# Colunas e valores
colunas = ['Regressor', 'Resultado_RMSE', 'Resultado_R2']
valores = [("N/A", "N/A", "N/A")]
```

In [24]:
```
# Tabela de resumo
df_resultados_treinamento = spark.createDataFrame(valores, colunas)
```

In [25]:
```
# Loop de treinamento
for regressor in regressores:

    # Resultado para cada regressor
    resultado_modelo = func_modulo_ml(regressor)

    # Grava os resultados
    df_resultados_treinamento = df_resultados_treinamento.union(resultado_modelo)
```

Modelo de Regressão Linear Sem Validação Cruzada:

RMSE em Teste: 10.131478111661103
Coeficiente R2 em Teste: 0.6378209342710394

Modelo de Regressão Linear Com Validação Cruzada:

RMSE em Teste: 10.131058302562344
Coeficiente R2 em Teste: 0.6378509482365038

Modelo Decision Tree Com Validação Cruzada:

RMSE em Teste: 9.161036559296653
Coeficiente R2 em Teste: 0.7038805352591408

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.util.SizeEstimator$ (file:/Users/emerson/anaconda
3/lib/python3.11/site-packages/pyspark/jars/spark-core_2.12-3.5.0.jar) to field java.nio.charset.Charse
t.name
WARNING: Please consider reporting this to the maintainers of org.apache.spark.util.SizeEstimator$
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
```

Modelo RandomForest Com Validação Cruzada:

RMSE em Teste: 7.682086955630076
Coeficiente R2 em Teste: 0.791773422660956

Modelo Gradient-Boosted Tree (GBT) Com Validação Cruzada:

RMSE em Teste: 6.954546124882646
Coeficiente R2 em Teste: 0.82934645796339

Modelo Isotonic Com Validação Cruzada:

RMSE em Teste: 14.136438977111792
Coeficiente R2 em Teste: 0.2948885473415652

In [26]:
```
# Retorna as linhas diferentes de N/A
df_resultados_treinamento = df_resultados_treinamento.where("Regressor!='N/A'")
df_resultados_treinamento.show(10, False)
```

```
[Stage 2152:==================================>                    (12 + 4) / 19]

+--------------------+-------------+------------+
|Regressor           |Resultado_RMSE|Resultado_R2|
+--------------------+-------------+------------+
|LinearRegression    |10.13        |0.637       |
|DecisionTreeRegressor|9.161        |0.703       |
|RandomForestRegressor|7.682        |0.791       |
|GBTRegressor        |6.954        |0.829       |
|IsotonicRegression  |14.13        |0.294       |
+--------------------+-------------+------------+
```

A partir das métricas de avaliação, podemos ver que o modelo GBT apresentou melhor performance nas duas métricas avaliativas, obtendo o RMSE (erro, quanto menor melhor) de 6.95 e o R2(acurácia, quanto maior melhor) de 0.829. Logo, será o modelo selecionado para ser levado a frente para a produção.

Fazendo Previsões com o Modelo Treinado

Para realizar as previsões com novos dados, temos que realizar os mesmos processos de tratamento nos novos dados, realizados nos dados utilizados para treinar o modelo, pois é o que o modelo espera a partir da sua construção e sequenciamento de tarefas. Logo, criaremos um data frame com o valor das variáveis de entrada, aplicaremos a normalização de dados na coluna "age", vetorizaremos os dados, padronizaremos os dados e então realizamos as novas previsões utilizando o melhor modelo.

In [27]:
```
# Valores de entrada
values = [(540,0.0,0.0,162,2.5,1040,676,28)]
```

In [28]:
```
# Nomes das colunas
column_names = dados.columns
column_names = column_names[0:8]
```

In [29]:
```
# Associa valores aos nomes de coluna
novos_dados = spark.createDataFrame(values, column_names)
```

In [30]:
```
# Transformação aplicada na coluna age
novos_dados = novos_dados.withColumn("age", log("age") +1)
```

In [31]:
```
# Lista de atributos
lista_atributos = ["cement",
                   "slag",
                   "flyash",
                   "water",
                   "superplasticizer",
                   "coarseaggregate",
                   "fineaggregate",
                   "age"]
```

In [32]:
```
# Vetorizador
assembler = VectorAssembler(inputCols = lista_atributos, outputCol = 'features')
```

In [33]:
```
# Transforma dados em vetor
novos_dados = assembler.transform(novos_dados).select('features')
```

In [34]:
```
# Padroniza os dados
novos_dados_scaled = scalerModel.transform(novos_dados)
```

In [35]:
```
# Seleciona a coluna resultante
novos_dados_final = novos_dados_scaled.select('scaledFeatures')
```

In [36]:
```
# Renomeia a coluna
novos_dados_final = novos_dados_final.withColumnRenamed('scaledFeatures','features')
```

In [37]:
```
# Previsões com novos dados usando melhor modelo
previsoes_novos_dados = GBT_BestModel.transform(novos_dados_final)
```

In [38]:
```
# Resultado
previsoes_novos_dados.show()
```

```
+------------------+----------------+
|            features|      prediction|
+------------------+----------------+
|[1.0,0.0,0.0,0.32...|40.59330248869691|
+------------------+----------------+
```

In [47]:
```
column_names
```

Out[47]:
```
['cement',
 'slag',
 'flyash',
 'water',
 'superplasticizer',
 'coarseaggregate',
 'fineaggregate',
 'age']
```

In [48]:
```
values
```

Out[48]:
```
[(540, 0.0, 0.0, 162, 2.5, 1040, 676, 28)]
```

Segundo a previsão do nosso modelo, o cimento composto pela medida referente aos valores das variáveis de entrada (ingredientes: cement 540, slag 0.0, flyash 0.0, water 162, superplasticizer 2.5, fineaggregate 1040, 676, age 28) terá um valor de 40.593CCS (Concrete Compressive Strength).