# **MARAMARATONA BEHIND THE CODE 2020**

## **DESAFIO 2: PARTE 1**

### **Introdução**

# **Em projetos de ciência de dados visando a construção de modelos de *machine learning*, ou aprendizado estatístico, é muito incomum que os dados iniciais estejam já no formato ideal para a construção de modelos. São necessários vários passos intermediários de pré-processamento de dados, como por exemplo a codificação de variáveis categóricas, normalização de variáveis numéricas, tratamento de dados faltantes, etc. A biblioteca scikit-learn -- uma das mais populares bibliotecas de código-aberto para *machine learning* no mundo -- possui diversas funções já integradas para a realização das transformações de dados mais utilizadas. Entretanto, em um fluxo comum de um modelo de aprendizado de máquina, é necessária a aplicação dessas transformações pelo menos duas vezes: a primeira vez para "treinar" o modelo, e depois novamente quando novos dados forem enviados como entrada para serem classificados por este modelo.**

# **Para facilitar o trabalho com esse tipo de fluxo, o scikit-learn possui também uma ferramenta chamada Pipeline, que nada mais é do que uma lista ordenada de transformações que devem ser aplicadas nos dados. Para auxiliar no desenvolvimento e no gerenciamento de todo o ciclo-de-vida dessas aplicações, alem do uso de Pipelines, as equipes de cientistas de dados podem utilizar em conjunto o Watson Machine Learning, que possui dezenas de ferramentas para treinar, gerenciar, hospedar e avaliar modelos baseados em aprendizado de máquina. Além disso, o Watson Machine Learning é capaz de encapsular pipelines e modelos em uma API pronta para uso e integração com outras aplicações.**

# **Durante o desafio 2, você participante irá aprender a construir uma Pipeline para um modelo de classificação e hospedá-lo como uma API com o auxílio do Watson Machine Learning. Uma vez hospedado, você poderá integrar o modelo criado com outras aplicações, como assistentes virtuais e muito mais. Neste notebook, será apresentado um exemplo funcional de criação de um modelo e de uma pipeline no scikit-learn (que você poderá utilizar como template para a sua solução!).**

### **Trabalhando com Pipelines do scikit-learn**

# **In [ ]:**

# ***# Primeiro, realizamos a instalação do scikit-learn versão 0.20.0 no Kernel deste notebook:***

# **!pip install scikit-learn==0.20.0 --upgrade**

# **In [ ]:**

# ***# Em seguida iremos importar diversas bibliotecas que serão utilizadas:***

# **​**

# ***# Pacote para trabalhar com JSON***

# **import json**

# **​**

# ***# Pacote para realizar requisições HTTP***

# **import requests**

# **​**

# ***# Pacote para exploração e análise de dados***

# **import pandas as pd**

# **​**

# ***# Pacote com métodos numéricos e representações matriciais***

# **import numpy as np**

# **​**

# ***# Pacote para construção de modelo baseado na técnica Gradient Boosting***

# **import xgboost as xgb**

# **​**

# ***# Pacotes do scikit-learn para pré-processamento de dados***

# ***# "SimpleImputer" é uma transformação para preencher valores faltantes em conjuntos de dados***

# **from sklearn.impute import SimpleImputer**

# **​**

# ***# Pacotes do scikit-learn para treinamento de modelos e construção de pipelines***

# ***# Método para separação de conjunto de dados em amostras de treino e teste***

# **from sklearn.model\_selection import train\_test\_split**

# ***# Método para criação de modelos baseados em árvores de decisão***

# **from sklearn.tree import DecisionTreeClassifier**

# ***# Classe para a criação de uma pipeline de machine-learning***

# **from sklearn.pipeline import Pipeline**

# **​**

# ***# Pacotes do scikit-learn para avaliação de modelos***

# ***# Métodos para validação cruzada do modelo criado***

# **from sklearn.model\_selection import KFold, cross\_validate**

### **Importando um .csv de seu projeto no IBM Cloud Pak for Data para o Kernel deste notebook**

# **Primeiro iremos importar o dataset fornecido para o desafio, que já está incluso neste projeto!**

# **Você pode realizar a importação dos dados de um arquivo .csv diretamente para o Kernel do notebook como um DataFrame da biblioteca Pandas, muito utilizada para a manipulação de dados em Python.**

# **Para realizar a importação, basta selecionar a próxima célula e seguir as instruções na imagem abaixo:**

# **alt text**

# **Após a seleção da opção "Insert to code", a célula abaixo será preenchida com o código necessário para importação e leitura dos dados no arquivo .csv como um DataFrame Pandas.**

# **In [ ]:**

# **​**

# **<< INSIRA O DATASET COMO UM PANDAS DATAFRAME NESTA CÉLULA! >>>**

# **​**

# **Temos 15 colunas presentes no dataset fornecido, sendo dezessete delas variáveis características (dados de entrada) e um delas uma variável-alvo (que queremos que o nosso modelo seja capaz de prever).**

# **As variáveis características são:**

# **MATRICULA - número de matrícula do estudante**

# **NOME - nome completo do estudante**

# **REPROVACOES\_DE - número de reprovações na disciplina de ``Direito Empresarial``**

# **REPROVACOES\_EM - número de reprovações na disciplina de ``Empreendedorismo``**

# **REPROVACOES\_MF - número de reprovações na disciplina de ``Matemática Financeira``**

# **REPROVACOES\_GO - número de reprovações na disciplina de ``Gestão Operacional``**

# **NOTA\_DE - média simples das notas do aluno na disciplina de ``Direito Empresarial`` (0-10)**

# **NOTA\_EM - média simples das notas do aluno na disciplina de ``Empreendedorismo`` (0-10)**

# **NOTA\_MF - média simples das notas do aluno na disciplina de ``Matemática Financeira`` (0-10)**

# **NOTA\_GO - média simples das notas do aluno na disciplina de ``Gestão Operacional`` (0-10)**

# **INGLES - variável binária que indica se o estudante tem conhecimento em língua inglesa (0 -> sim ou 1 -> não).**

# **H\_AULA\_PRES - horas de estudo presencial realizadas pelo estudante**

# **TAREFAS\_ONLINE - número de tarefas online entregues pelo estudante**

# **FALTAS - número de faltas acumuladas do estudante (todas disciplinas)**

# **A variável-alvo é:**

# **PERFIL - uma \*string\* que indica uma de cinco possibilidades:**

# **"EXCELENTE" - Estudante não necessita de mentoria**

# **"MUITO BOM" - Estudante não necessita de mentoria**

# **"HUMANAS" - Estudante necessita de mentoria exclusivamente em matérias com conteúdo de ciências humanas**

# **"EXATAS" - Estudante necessita de mentoria apenas em disciplinas com conteúdo de ciências exatas**

# **"DIFICULDADE" - Estudante necessita de mentoria em duas ou mais disciplinas**

# **Com um modelo capaz de classificar um estudante em uma dessas categorias, podemos automatizar parte da mentoria estudantil através de assistentes virtuais, que serão capazes de recomendar práticas de estudo e conteúdo personalizado com base nas necessidades de cada aluno.**

### **Explorando os dados fornecidos**

# **Podemos continuar a exploração dos dados fornecidos com a função info():**

# **In [ ]:**

# **df\_data\_1.info()**

# **É notado que existem variáveis do tipo float64 (números "decimais"), variáveis do tipo int64 (números inteiros) e do tipo object (nesse caso são *strings*, ou texto).**

# **Como a maioria dos algoritmos de aprendizado estatístico supervisionado só aceita valores numéricos como entrada, é necessário então o pré-processamento das variáveis do tipo "object" antes de usar esse dataset como entrada para o treinamento de um modelo. Também é notado que existem valores faltantes em várias colunas. Esses valores faltantes também devem ser tratados antes de serem construídos modelos com esse conjunto de dados base.**

# **A função describe() gera várias informações sobre as variáveis numéricas que também podem ser úteis:**

# **In [ ]:**

# **df\_data\_1.describe()**

### **Visualizações**

# **Para visualizar o dataset fornecido, podemos utilizar as bibliotecas matplotlib e seaborn:**

# **In [ ]:**

# **import matplotlib.pyplot as plt**

# **import seaborn as sns**

# **%matplotlib inline**

# **In [ ]:**

# **fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))**

# **​**

# **sns.countplot(ax=axes[0], x='REPROVACOES\_DE', data=df\_data\_1)**

# **sns.countplot(ax=axes[1], x='REPROVACOES\_EM', data=df\_data\_1)**

# **sns.countplot(ax=axes[2], x='REPROVACOES\_MF', data=df\_data\_1)**

# **sns.countplot(ax=axes[3], x='REPROVACOES\_GO', data=df\_data\_1)**

# **In [ ]:**

# **fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))**

# **​**

# **sns.distplot(df\_data\_1['NOTA\_DE'], ax=axes[0])**

# **sns.distplot(df\_data\_1['NOTA\_EM'], ax=axes[1])**

# **sns.distplot(df\_data\_1['NOTA\_MF'], ax=axes[2])**

# **sns.distplot(df\_data\_1['NOTA\_GO'].dropna(), ax=axes[3])**

# **In [ ]:**

# **fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))**

# **​**

# **sns.countplot(ax=axes[0], x='INGLES', data=df\_data\_1)**

# **sns.countplot(ax=axes[1], x='FALTAS', data=df\_data\_1)**

# **sns.countplot(ax=axes[2], x='H\_AULA\_PRES', data=df\_data\_1)**

# **sns.countplot(ax=axes[3], x='TAREFAS\_ONLINE', data=df\_data\_1)**

# **In [ ]:**

# **fig = plt.plot()**

# **sns.countplot(x='PERFIL', data=df\_data\_1)**

### **Realizando o pré-processamento dos dados**

# **Para o pré-processamento dos dados serão apresentadas duas transformações básicas neste notebook, demonstrando a construção de uma Pipeline com um modelo funcional. Esta Pipeline funcional fornecida deverá ser melhorada pelo participante para que o modelo final alcance a maior acurácia possível, garantindo uma pontuação maior no desafio. Essa melhoria pode ser feita apenas no pré-processamento dos dados, na escolha de um algoritmo para treinamento de modelo diferente, ou até mesmo na alteração do *framework* usado (entretanto só será fornecido um exemplo pronto de integração do Watson Machine Learning com o *scikit-learn*).**

# **A primeira transformação (passo na nossa Pipeline) será a exclusão da coluna "NOME" do nosso dataset, que além de não ser uma variável numérica, também não é uma variável relacionada ao desempenho dos estudantes nas disciplinas. Existem funções prontas no scikit-learn para a realização dessa transformação, entretanto nosso exemplo irá demonstrar como criar uma transformação personalizada do zero no scikit-learn. Se desejado, o participante poderá utilizar esse exemplo para criar outras transformações e adicioná-las à Pipeline final :)**

#### **Transformação 1: excluindo colunas do dataset**

# **Para a criação de uma transformação de dados personalizada no scikit-learn, é necessária basicamente a criação de uma classe com os métodos transform e fit. No método transform será executada a lógica da nossa transformação.**

# **Na próxima célula é apresentado o código completo de uma transformação DropColumns para a remoção de colunas de um DataFrame pandas.**

# **In [ ]:**

# **from sklearn.base import BaseEstimator, TransformerMixin**

# **​**

# **​**

# ***# All sklearn Transforms must have the `transform` and `fit` methods***

# **class DropColumns(BaseEstimator, TransformerMixin):**

# **def \_\_init\_\_(self, columns):**

# **self.columns = columns**

# **​**

# **def fit(self, X, y=None):**

# **return self**

# 

# **def transform(self, X):**

# ***# Primeiro realizamos a cópia do dataframe 'X' de entrada***

# **data = X.copy()**

# ***# Retornamos um novo dataframe sem as colunas indesejadas***

# **return data.drop(labels=self.columns, axis='columns')**

# **Para aplicar essa transformação em um DataFrame pandas, basta instanciar um objeto *DropColumns* e chamar o método transform().**

# **In [ ]:**

# ***# Instanciando uma transformação DropColumns***

# **rm\_columns = DropColumns(**

# **columns=["NOME"] *# Essa transformação recebe como parâmetro uma lista com os nomes das colunas indesejadas***

# **)**

# **​**

# **print(rm\_columns)**

# **In [ ]:**

# ***# Visualizando as colunas do dataset original***

# **print("Colunas do dataset original: \n")**

# **print(df\_data\_1.columns)**

# **In [ ]:**

# ***# Aplicando a transformação ``DropColumns`` ao conjunto de dados base***

# **rm\_columns.fit(X=df\_data\_1)**

# **​**

# ***# Reconstruindo um DataFrame Pandas com o resultado da transformação***

# **df\_data\_2 = pd.DataFrame.from\_records(**

# **data=rm\_columns.transform(**

# **X=df\_data\_1**

# **),**

# **)**

# **In [ ]:**

# ***# Visualizando as colunas do dataset transformado***

# **print("Colunas do dataset após a transformação ``DropColumns``: \n")**

# **print(df\_data\_2.columns)**

# **Nota-se que a coluna "NOME" foi removida e nosso dataset agora poossui apenas 17 colunas.**

#### **Transformação 2: tratando dados faltantes**

# **Para tratar os dados faltantes em nosso conjunto de dados, iremos agora utilizar uma transformação pronta da biblioteca scikit-learn, chamada SimpleImputer.**

# **Essa transformação permite diversas estratégias para o tratamento de dados faltantes. A documentação oficial pode ser encontrada em:** [**https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html**](https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html)

# **Neste exemplo iremos simplesmente transformar todos os valores faltantes em zero.**

# **In [ ]:**

# ***# Criação de um objeto ``SimpleImputer``***

# **si = SimpleImputer(**

# **missing\_values=np.nan, *# os valores faltantes são do tipo ``np.nan`` (padrão Pandas)***

# **strategy='constant', *# a estratégia escolhida é a alteração do valor faltante por uma constante***

# **fill\_value=0, *# a constante que será usada para preenchimento dos valores faltantes é um int64=0.***

# **verbose=0,**

# **copy=True**

# **)**

# **In [ ]:**

# ***# Visualizando os dados faltantes do dataset após a primeira transformação (df\_data\_2)***

# **print("Valores nulos antes da transformação SimpleImputer: \n\n{}\n".format(df\_data\_2.isnull().sum(axis = 0)))**

# **In [ ]:**

# ***# Aplicamos o SimpleImputer ``si`` ao conjunto de dados df\_data\_2 (resultado da primeira transformação)***

# **si.fit(X=df\_data\_2)**

# **​**

# ***# Reconstrução de um novo DataFrame Pandas com o conjunto imputado (df\_data\_3)***

# **df\_data\_3 = pd.DataFrame.from\_records(**

# **data=si.transform(**

# **X=df\_data\_2**

# **), *# o resultado SimpleImputer.transform(<<pandas dataframe>>) é lista de listas***

# **columns=df\_data\_2.columns *# as colunas originais devem ser conservadas nessa transformação***

# **)**

# **In [ ]:**

# ***# Visualizando os dados faltantes do dataset após a segunda transformação (SimpleImputer) (df\_data\_3)***

# **print("Valores nulos no dataset após a transformação SimpleImputer: \n\n{}\n".format(df\_data\_3.isnull().sum(axis = 0)))**

# **Nota-se que não temos mais nenhum valor faltante no nosso conjunto de dados :)**

# **Vale salientar que nem sempre a alteração dos valores faltantes por 0 é a melhor estratégia. O participante é incentivado a estudar e implementar estratégias diferentes de tratamento dos valores faltantes para aprimorar seu modelo e melhorar sua pontuação final.**

### **Treinando um modelo de classificação**

# **Finalizado o pré-processamento, já temos o conjunto de dados no formato necessário para o treinamento do nosso modelo:**

# **In [ ]:**

# **df\_data\_3.head()**

# **No exemplo fornecido, iremos utilizar todas as colunas, exceto a coluna LABELS como *features* (variáveis de entrada).**

# **A variável LABELS será a variável-alvo do modelo, conforme descrito no enunciado do desafio.**

#### **Definindo as features do modelo**

# **In [ ]:**

# ***# Definição das colunas que serão features (nota-se que a coluna NOME não está presente)***

# **features = [**

# **"MATRICULA", 'REPROVACOES\_DE', 'REPROVACOES\_EM', "REPROVACOES\_MF", "REPROVACOES\_GO",**

# **"NOTA\_DE", "NOTA\_EM", "NOTA\_MF", "NOTA\_GO",**

# **"INGLES", "H\_AULA\_PRES", "TAREFAS\_ONLINE", "FALTAS",**

# **]**

# **​**

# ***# Definição da variável-alvo***

# **target = ["PERFIL"]**

# **​**

# ***# Preparação dos argumentos para os métodos da biblioteca ``scikit-learn``***

# **X = df\_data\_3[features]**

# **y = df\_data\_3[target]**

# **O conjunto de entrada (X):**

# **In [ ]:**

# **X.head()**

# **As variáveis-alvo correspondentes (y):**

# **In [ ]:**

# **y.head()**

#### **Separando o dataset em um conjunto de treino e um conjunto de teste**

# **Iremos separar o dataset fornecido em dois grupos: um para treinar nosso modelo, e outro para testarmos o resultado através de um teste cego. A separação do dataset pode ser feita facilmente com o método *train\_test\_split()* do scikit-learn:**

# **In [ ]:**

# ***# Separação dos dados em um conjunto de treino e um conjunto de teste***

# **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=337)**

#### **Criando um modelo baseado em árvores de decisão**

# **No exemplo fornecido iremos criar um classificador baseado em árvores de decisão.**

# **O primeiro passo é basicamente instanciar um objeto *DecisionTreeClassifier()* da biblioteca scikit-learn.**

# **In [ ]:**

# ***# Criação da árvore de decisão com a biblioteca ``scikit-learn``:***

# **dtc\_model = DecisionTreeClassifier() *# O modelo será criado com os parâmetros padrões da biblioteca***

# **Material teórico sobre árvores de decisão na documentação oficial do scikit-learn:** [**https://scikit-learn.org/stable/modules/tree.html**](https://scikit-learn.org/stable/modules/tree.html)

# **Um guia para iniciantes no mundo do machine learning:** [**https://developer.ibm.com/br/articles/cc-beginner-guide-machine-learning-ai-cognitive/**](https://developer.ibm.com/br/articles/cc-beginner-guide-machine-learning-ai-cognitive/)

#### **Execução do evento de treino de uma árvore de decisão**

# **In [ ]:**

# ***# Treino do modelo (é chamado o método \*fit()\* com os conjuntos de treino)***

# **dtc\_model.fit(**

# **X\_train,**

# **y\_train**

# **)**

#### **Execução de predições e avaliação do modelo criado**

# **In [ ]:**

# ***# Realização de teste cego no modelo criado***

# **y\_pred = dtc\_MARATONA BEHIND THE CODE 2020**

## **DESAFIO 2: PARTE 1**

### **Introdução**

# **Em projetos de ciência de dados visando a construção de modelos de *machine learning*, ou aprendizado estatístico, é muito incomum que os dados iniciais estejam já no formato ideal para a construção de modelos. São necessários vários passos intermediários de pré-processamento de dados, como por exemplo a codificação de variáveis categóricas, normalização de variáveis numéricas, tratamento de dados faltantes, etc. A biblioteca scikit-learn -- uma das mais populares bibliotecas de código-aberto para *machine learning* no mundo -- possui diversas funções já integradas para a realização das transformações de dados mais utilizadas. Entretanto, em um fluxo comum de um modelo de aprendizado de máquina, é necessária a aplicação dessas transformações pelo menos duas vezes: a primeira vez para "treinar" o modelo, e depois novamente quando novos dados forem enviados como entrada para serem classificados por este modelo.**

# **Para facilitar o trabalho com esse tipo de fluxo, o scikit-learn possui também uma ferramenta chamada Pipeline, que nada mais é do que uma lista ordenada de transformações que devem ser aplicadas nos dados. Para auxiliar no desenvolvimento e no gerenciamento de todo o ciclo-de-vida dessas aplicações, alem do uso de Pipelines, as equipes de cientistas de dados podem utilizar em conjunto o Watson Machine Learning, que possui dezenas de ferramentas para treinar, gerenciar, hospedar e avaliar modelos baseados em aprendizado de máquina. Além disso, o Watson Machine Learning é capaz de encapsular pipelines e modelos em uma API pronta para uso e integração com outras aplicações.**

# **Durante o desafio 2, você participante irá aprender a construir uma Pipeline para um modelo de classificação e hospedá-lo como uma API com o auxílio do Watson Machine Learning. Uma vez hospedado, você poderá integrar o modelo criado com outras aplicações, como assistentes virtuais e muito mais. Neste notebook, será apresentado um exemplo funcional de criação de um modelo e de uma pipeline no scikit-learn (que você poderá utilizar como template para a sua solução!).**

### **Trabalhando com Pipelines do scikit-learn**

# **In [ ]:**

# ***# Primeiro, realizamos a instalação do scikit-learn versão 0.20.0 no Kernel deste notebook:***

# **!pip install scikit-learn==0.20.0 --upgrade**

# **In [ ]:**

# ***# Em seguida iremos importar diversas bibliotecas que serão utilizadas:***

# **​**

# ***# Pacote para trabalhar com JSON***

# **import json**

# **​**

# ***# Pacote para realizar requisições HTTP***

# **import requests**

# **​**

# ***# Pacote para exploração e análise de dados***

# **import pandas as pd**

# **​**

# ***# Pacote com métodos numéricos e representações matriciais***

# **import numpy as np**

# **​**

# ***# Pacote para construção de modelo baseado na técnica Gradient Boosting***

# **import xgboost as xgb**

# **​**

# ***# Pacotes do scikit-learn para pré-processamento de dados***

# ***# "SimpleImputer" é uma transformação para preencher valores faltantes em conjuntos de dados***

# **from sklearn.impute import SimpleImputer**

# **​**

# ***# Pacotes do scikit-learn para treinamento de modelos e construção de pipelines***

# ***# Método para separação de conjunto de dados em amostras de treino e teste***

# **from sklearn.model\_selection import train\_test\_split**

# ***# Método para criação de modelos baseados em árvores de decisão***

# **from sklearn.tree import DecisionTreeClassifier**

# ***# Classe para a criação de uma pipeline de machine-learning***

# **from sklearn.pipeline import Pipeline**

# **​**

# ***# Pacotes do scikit-learn para avaliação de modelos***

# ***# Métodos para validação cruzada do modelo criado***

# **from sklearn.model\_selection import KFold, cross\_validate**

### **Importando um .csv de seu projeto no IBM Cloud Pak for Data para o Kernel deste notebook**

# **Primeiro iremos importar o dataset fornecido para o desafio, que já está incluso neste projeto!**

# **Você pode realizar a importação dos dados de um arquivo .csv diretamente para o Kernel do notebook como um DataFrame da biblioteca Pandas, muito utilizada para a manipulação de dados em Python.**

# **Para realizar a importação, basta selecionar a próxima célula e seguir as instruções na imagem abaixo:**

# **alt text**

# **Após a seleção da opção "Insert to code", a célula abaixo será preenchida com o código necessário para importação e leitura dos dados no arquivo .csv como um DataFrame Pandas.**

# **In [ ]:**

# **​**

# **<< INSIRA O DATASET COMO UM PANDAS DATAFRAME NESTA CÉLULA! >>>**

# **​**

# **Temos 15 colunas presentes no dataset fornecido, sendo dezessete delas variáveis características (dados de entrada) e um delas uma variável-alvo (que queremos que o nosso modelo seja capaz de prever).**

# **As variáveis características são:**

# **MATRICULA - número de matrícula do estudante**

# **NOME - nome completo do estudante**

# **REPROVACOES\_DE - número de reprovações na disciplina de ``Direito Empresarial``**

# **REPROVACOES\_EM - número de reprovações na disciplina de ``Empreendedorismo``**

# **REPROVACOES\_MF - número de reprovações na disciplina de ``Matemática Financeira``**

# **REPROVACOES\_GO - número de reprovações na disciplina de ``Gestão Operacional``**

# **NOTA\_DE - média simples das notas do aluno na disciplina de ``Direito Empresarial`` (0-10)**

# **NOTA\_EM - média simples das notas do aluno na disciplina de ``Empreendedorismo`` (0-10)**

# **NOTA\_MF - média simples das notas do aluno na disciplina de ``Matemática Financeira`` (0-10)**

# **NOTA\_GO - média simples das notas do aluno na disciplina de ``Gestão Operacional`` (0-10)**

# **INGLES - variável binária que indica se o estudante tem conhecimento em língua inglesa (0 -> sim ou 1 -> não).**

# **H\_AULA\_PRES - horas de estudo presencial realizadas pelo estudante**

# **TAREFAS\_ONLINE - número de tarefas online entregues pelo estudante**

# **FALTAS - número de faltas acumuladas do estudante (todas disciplinas)**

# **A variável-alvo é:**

# **PERFIL - uma \*string\* que indica uma de cinco possibilidades:**

# **"EXCELENTE" - Estudante não necessita de mentoria**

# **"MUITO BOM" - Estudante não necessita de mentoria**

# **"HUMANAS" - Estudante necessita de mentoria exclusivamente em matérias com conteúdo de ciências humanas**

# **"EXATAS" - Estudante necessita de mentoria apenas em disciplinas com conteúdo de ciências exatas**

# **"DIFICULDADE" - Estudante necessita de mentoria em duas ou mais disciplinas**

# **Com um modelo capaz de classificar um estudante em uma dessas categorias, podemos automatizar parte da mentoria estudantil através de assistentes virtuais, que serão capazes de recomendar práticas de estudo e conteúdo personalizado com base nas necessidades de cada aluno.**

### **Explorando os dados fornecidos**

# **Podemos continuar a exploração dos dados fornecidos com a função info():**

# **In [ ]:**

# **df\_data\_1.info()**

# **É notado que existem variáveis do tipo float64 (números "decimais"), variáveis do tipo int64 (números inteiros) e do tipo object (nesse caso são *strings*, ou texto).**

# **Como a maioria dos algoritmos de aprendizado estatístico supervisionado só aceita valores numéricos como entrada, é necessário então o pré-processamento das variáveis do tipo "object" antes de usar esse dataset como entrada para o treinamento de um modelo. Também é notado que existem valores faltantes em várias colunas. Esses valores faltantes também devem ser tratados antes de serem construídos modelos com esse conjunto de dados base.**

# **A função describe() gera várias informações sobre as variáveis numéricas que também podem ser úteis:**

# **In [ ]:**

# **df\_data\_1.describe()**

### **Visualizações**

# **Para visualizar o dataset fornecido, podemos utilizar as bibliotecas matplotlib e seaborn:**

# **In [ ]:**

# **import matplotlib.pyplot as plt**

# **import seaborn as sns**

# **%matplotlib inline**

# **In [ ]:**

# **fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))**

# **​**

# **sns.countplot(ax=axes[0], x='REPROVACOES\_DE', data=df\_data\_1)**

# **sns.countplot(ax=axes[1], x='REPROVACOES\_EM', data=df\_data\_1)**

# **sns.countplot(ax=axes[2], x='REPROVACOES\_MF', data=df\_data\_1)**

# **sns.countplot(ax=axes[3], x='REPROVACOES\_GO', data=df\_data\_1)**

# **In [ ]:**

# **fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))**

# **​**

# **sns.distplot(df\_data\_1['NOTA\_DE'], ax=axes[0])**

# **sns.distplot(df\_data\_1['NOTA\_EM'], ax=axes[1])**

# **sns.distplot(df\_data\_1['NOTA\_MF'], ax=axes[2])**

# **sns.distplot(df\_data\_1['NOTA\_GO'].dropna(), ax=axes[3])**

# **In [ ]:**

# **fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(28, 4))**

# **​**

# **sns.countplot(ax=axes[0], x='INGLES', data=df\_data\_1)**

# **sns.countplot(ax=axes[1], x='FALTAS', data=df\_data\_1)**

# **sns.countplot(ax=axes[2], x='H\_AULA\_PRES', data=df\_data\_1)**

# **sns.countplot(ax=axes[3], x='TAREFAS\_ONLINE', data=df\_data\_1)**

# **In [ ]:**

# **fig = plt.plot()**

# **sns.countplot(x='PERFIL', data=df\_data\_1)**

### **Realizando o pré-processamento dos dados**

# **Para o pré-processamento dos dados serão apresentadas duas transformações básicas neste notebook, demonstrando a construção de uma Pipeline com um modelo funcional. Esta Pipeline funcional fornecida deverá ser melhorada pelo participante para que o modelo final alcance a maior acurácia possível, garantindo uma pontuação maior no desafio. Essa melhoria pode ser feita apenas no pré-processamento dos dados, na escolha de um algoritmo para treinamento de modelo diferente, ou até mesmo na alteração do *framework* usado (entretanto só será fornecido um exemplo pronto de integração do Watson Machine Learning com o *scikit-learn*).**

# **A primeira transformação (passo na nossa Pipeline) será a exclusão da coluna "NOME" do nosso dataset, que além de não ser uma variável numérica, também não é uma variável relacionada ao desempenho dos estudantes nas disciplinas. Existem funções prontas no scikit-learn para a realização dessa transformação, entretanto nosso exemplo irá demonstrar como criar uma transformação personalizada do zero no scikit-learn. Se desejado, o participante poderá utilizar esse exemplo para criar outras transformações e adicioná-las à Pipeline final :)**

#### **Transformação 1: excluindo colunas do dataset**

# **Para a criação de uma transformação de dados personalizada no scikit-learn, é necessária basicamente a criação de uma classe com os métodos transform e fit. No método transform será executada a lógica da nossa transformação.**

# **Na próxima célula é apresentado o código completo de uma transformação DropColumns para a remoção de colunas de um DataFrame pandas.**

# **In [ ]:**

# **from sklearn.base import BaseEstimator, TransformerMixin**

# **​**

# **​**

# ***# All sklearn Transforms must have the `transform` and `fit` methods***

# **class DropColumns(BaseEstimator, TransformerMixin):**

# **def \_\_init\_\_(self, columns):**

# **self.columns = columns**

# **​**

# **def fit(self, X, y=None):**

# **return self**

# 

# **def transform(self, X):**

# ***# Primeiro realizamos a cópia do dataframe 'X' de entrada***

# **data = X.copy()**

# ***# Retornamos um novo dataframe sem as colunas indesejadas***

# **return data.drop(labels=self.columns, axis='columns')**

# **Para aplicar essa transformação em um DataFrame pandas, basta instanciar um objeto *DropColumns* e chamar o método transform().**

# **In [ ]:**

# ***# Instanciando uma transformação DropColumns***

# **rm\_columns = DropColumns(**

# **columns=["NOME"] *# Essa transformação recebe como parâmetro uma lista com os nomes das colunas indesejadas***

# **)**

# **​**

# **print(rm\_columns)**

# **In [ ]:**

# ***# Visualizando as colunas do dataset original***

# **print("Colunas do dataset original: \n")**

# **print(df\_data\_1.columns)**

# **In [ ]:**

# ***# Aplicando a transformação ``DropColumns`` ao conjunto de dados base***

# **rm\_columns.fit(X=df\_data\_1)**

# **​**

# ***# Reconstruindo um DataFrame Pandas com o resultado da transformação***

# **df\_data\_2 = pd.DataFrame.from\_records(**

# **data=rm\_columns.transform(**

# **X=df\_data\_1**

# **),**

# **)**

# **In [ ]:**

# ***# Visualizando as colunas do dataset transformado***

# **print("Colunas do dataset após a transformação ``DropColumns``: \n")**

# **print(df\_data\_2.columns)**

# **Nota-se que a coluna "NOME" foi removida e nosso dataset agora poossui apenas 17 colunas.**

#### **Transformação 2: tratando dados faltantes**

# **Para tratar os dados faltantes em nosso conjunto de dados, iremos agora utilizar uma transformação pronta da biblioteca scikit-learn, chamada SimpleImputer.**

# **Essa transformação permite diversas estratégias para o tratamento de dados faltantes. A documentação oficial pode ser encontrada em:** [**https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html**](https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html)

# **Neste exemplo iremos simplesmente transformar todos os valores faltantes em zero.**

# **In [ ]:**

# ***# Criação de um objeto ``SimpleImputer``***

# **si = SimpleImputer(**

# **missing\_values=np.nan, *# os valores faltantes são do tipo ``np.nan`` (padrão Pandas)***

# **strategy='constant', *# a estratégia escolhida é a alteração do valor faltante por uma constante***

# **fill\_value=0, *# a constante que será usada para preenchimento dos valores faltantes é um int64=0.***

# **verbose=0,**

# **copy=True**

# **)**

# **In [ ]:**

# ***# Visualizando os dados faltantes do dataset após a primeira transformação (df\_data\_2)***

# **print("Valores nulos antes da transformação SimpleImputer: \n\n{}\n".format(df\_data\_2.isnull().sum(axis = 0)))**

# **In [ ]:**

# ***# Aplicamos o SimpleImputer ``si`` ao conjunto de dados df\_data\_2 (resultado da primeira transformação)***

# **si.fit(X=df\_data\_2)**

# **​**

# ***# Reconstrução de um novo DataFrame Pandas com o conjunto imputado (df\_data\_3)***

# **df\_data\_3 = pd.DataFrame.from\_records(**

# **data=si.transform(**

# **X=df\_data\_2**

# **), *# o resultado SimpleImputer.transform(<<pandas dataframe>>) é lista de listas***

# **columns=df\_data\_2.columns *# as colunas originais devem ser conservadas nessa transformação***

# **)**

# **In [ ]:**

# ***# Visualizando os dados faltantes do dataset após a segunda transformação (SimpleImputer) (df\_data\_3)***

# **print("Valores nulos no dataset após a transformação SimpleImputer: \n\n{}\n".format(df\_data\_3.isnull().sum(axis = 0)))**

# **Nota-se que não temos mais nenhum valor faltante no nosso conjunto de dados :)**

# **Vale salientar que nem sempre a alteração dos valores faltantes por 0 é a melhor estratégia. O participante é incentivado a estudar e implementar estratégias diferentes de tratamento dos valores faltantes para aprimorar seu modelo e melhorar sua pontuação final.**

### **Treinando um modelo de classificação**

# **Finalizado o pré-processamento, já temos o conjunto de dados no formato necessário para o treinamento do nosso modelo:**

# **In [ ]:**

# **df\_data\_3.head()**

# **No exemplo fornecido, iremos utilizar todas as colunas, exceto a coluna LABELS como *features* (variáveis de entrada).**

# **A variável LABELS será a variável-alvo do modelo, conforme descrito no enunciado do desafio.**

#### **Definindo as features do modelo**

# **In [ ]:**

# ***# Definição das colunas que serão features (nota-se que a coluna NOME não está presente)***

# **features = [**

# **"MATRICULA", 'REPROVACOES\_DE', 'REPROVACOES\_EM', "REPROVACOES\_MF", "REPROVACOES\_GO",**

# **"NOTA\_DE", "NOTA\_EM", "NOTA\_MF", "NOTA\_GO",**

# **"INGLES", "H\_AULA\_PRES", "TAREFAS\_ONLINE", "FALTAS",**

# **]**

# **​**

# ***# Definição da variável-alvo***

# **target = ["PERFIL"]**

# **​**

# ***# Preparação dos argumentos para os métodos da biblioteca ``scikit-learn``***

# **X = df\_data\_3[features]**

# **y = df\_data\_3[target]**

# **O conjunto de entrada (X):**

# **In [ ]:**

# **X.head()**

# **As variáveis-alvo correspondentes (y):**

# **In [ ]:**

# **y.head()**

#### **Separando o dataset em um conjunto de treino e um conjunto de teste**

# **Iremos separar o dataset fornecido em dois grupos: um para treinar nosso modelo, e outro para testarmos o resultado através de um teste cego. A separação do dataset pode ser feita facilmente com o método *train\_test\_split()* do scikit-learn:**

# **In [ ]:**

# ***# Separação dos dados em um conjunto de treino e um conjunto de teste***

# **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=337)**

#### **Criando um modelo baseado em árvores de decisão**

# **No exemplo fornecido iremos criar um classificador baseado em árvores de decisão.**

# **O primeiro passo é basicamente instanciar um objeto *DecisionTreeClassifier()* da biblioteca scikit-learn.**

# **In [ ]:**

# ***# Criação da árvore de decisão com a biblioteca ``scikit-learn``:***

# **dtc\_model = DecisionTreeClassifier() *# O modelo será criado com os parâmetros padrões da biblioteca***

# **Material teórico sobre árvores de decisão na documentação oficial do scikit-learn:** [**https://scikit-learn.org/stable/modules/tree.html**](https://scikit-learn.org/stable/modules/tree.html)

# **Um guia para iniciantes no mundo do machine learning:** [**https://developer.ibm.com/br/articles/cc-beginner-guide-machine-learning-ai-cognitive/**](https://developer.ibm.com/br/articles/cc-beginner-guide-machine-learning-ai-cognitive/)

#### **Execução do evento de treino de uma árvore de decisão**

# **In [ ]:**

# ***# Treino do modelo (é chamado o método \*fit()\* com os conjuntos de treino)***

# **dtc\_model.fit(**

# **X\_train,**

# **y\_train**

# **)**

#### **Execução de predições e avaliação do modelo criado**

# **In [ ]:**

# ***# Realização de teste cego no modelo criado***

# **y\_pred = dtc\_model.predict(X\_test)**

# **In [ ]:**

# **X\_test.head()**

# **In [ ]:**

# **print(y\_pred)**

# **In [ ]:**

# **from sklearn.metrics import accuracy\_score**

# **​**

# ***# Acurácia alcançada pela árvore de decisão***

# **print("Acurácia: {}%".format(100\*round(accuracy\_score(y\_test, y\_pred), 2)))**

# 

# **Neste notebook foi demonstrado como trabalhar com transformações e modelos com a biblioteca scikit-learn. É recomendado que o participante realize seus experimentos editando o código fornecido aqui até que um modelo com acurácia elevada seja alcançado.**

# **Quando você estiver satisfeito com seu modelo, pode passar para a segunda etapa do desafio -- encapsular seu modelo como uma API REST pronta para uso com o Watson Machine Learning!**

# **O notebook para a segunda etapa já se encontra neste projeto, basta acessar a aba ASSETS e inicializá-lo! Não se esqueca de antes desligar o Kernel deste notebook para reduzir o consumo de sua camada grátis do IBM Cloud Pak for Data.**

# **In [ ]:**

# **​**

# **model.predict(X\_test)**

# **In [ ]:**

# **X\_test.head()**

# **In [ ]:**

# **print(y\_pred)**

# **In [ ]:**

# **from sklearn.metrics import accuracy\_score**

# **​**

# ***# Acurácia alcançada pela árvore de decisão***

# **print("Acurácia: {}%".format(100\*round(accuracy\_score(y\_test, y\_pred), 2)))**

# 

# **Neste notebook foi demonstrado como trabalhar com transformações e modelos com a biblioteca scikit-learn. É recomendado que o participante realize seus experimentos editando o código fornecido aqui até que um modelo com acurácia elevada seja alcançado.**

# **Quando você estiver satisfeito com seu modelo, pode passar para a segunda etapa do desafio -- encapsular seu modelo como uma API REST pronta para uso com o Watson Machine Learning!**

# **O notebook para a segunda etapa já se encontra neste projeto, basta acessar a aba ASSETS e inicializá-lo! Não se esqueca de antes desligar o Kernel deste notebook para reduzir o consumo de sua camada grátis do IBM Cloud Pak for Data.**

# **In [ ]:**

# **​**

# **TONA BEHIND THE CODE 2020**

## **DESAFIO 2: PARTE 1**

### **Introdução**

Em projetos de ciência de dados visando a construção de modelos de *machine learning*, ou aprendizado estatístico, é muito incomum que os dados iniciais estejam já no formato ideal para a construção de modelos. São necessários vários passos intermediários de pré-processamento de dados, como por exemplo a codificação de variáveis categóricas, normalização de variáveis numéricas, tratamento de dados faltantes, etc. A biblioteca **scikit-learn** -- uma das mais populares bibliotecas de código-aberto para *machine learning* no mundo -- possui diversas funções já integradas para a realização das transformações de dados mais utilizadas. Entretanto, em um fluxo comum de um modelo de aprendizado de máquina, é necessária a aplicação dessas transformações pelo menos duas vezes: a primeira vez para "treinar" o modelo, e depois novamente quando novos dados forem enviados como entrada para serem classificados por este modelo.

Para facilitar o trabalho com esse tipo de fluxo, o scikit-learn possui também uma ferramenta chamada **Pipeline**, que nada mais é do que uma lista ordenada de transformações que devem ser aplicadas nos dados. Para auxiliar no desenvolvimento e no gerenciamento de todo o ciclo-de-vida dessas aplicações, alem do uso de Pipelines, as equipes de cientistas de dados podem utilizar em conjunto o **Watson Machine Learning**, que possui dezenas de ferramentas para treinar, gerenciar, hospedar e avaliar modelos baseados em aprendizado de máquina. Além disso, o Watson Machine Learning é capaz de encapsular pipelines e modelos em uma API pronta para uso e integração com outras aplicações.

Durante o desafio 2, você participante irá aprender a construir uma **Pipeline** para um modelo de classificação e hospedá-lo como uma API com o auxílio do Watson Machine Learning. Uma vez hospedado, você poderá integrar o modelo criado com outras aplicações, como assistentes virtuais e muito mais. Neste notebook, será apresentado um exemplo funcional de criação de um modelo e de uma pipeline no scikit-learn (que você poderá utilizar como template para a sua solução!).

### **Trabalhando com Pipelines do scikit-learn**

In [ ]:

*# Primeiro, realizamos a instalação do scikit-learn versão 0.20.0 no Kernel deste notebook:*

**!**pip install scikit**-**learn**==**0.20.0 **--**upgrade

In [ ]:

*# Em seguida iremos importar diversas bibliotecas que serão utilizadas:*

​

*# Pacote para trabalhar com JSON*

**import** json

​

*# Pacote para realizar requisições HTTP*

**import** requests

​

*# Pacote para exploração e análise de dados*

**import** pandas **as** pd

​

*# Pacote com métodos numéricos e representações matriciais*

**import** numpy **as** np

​

*# Pacote para construção de modelo baseado na técnica Gradient Boosting*

**import** xgboost **as** xgb

​

*# Pacotes do scikit-learn para pré-processamento de dados*

*# "SimpleImputer" é uma transformação para preencher valores faltantes em conjuntos de dados*

**from** sklearn.impute **import** SimpleImputer

​

*# Pacotes do scikit-learn para treinamento de modelos e construção de pipelines*

*# Método para separação de conjunto de dados em amostras de treino e teste*

**from** sklearn.model\_selection **import** train\_test\_split

*# Método para criação de modelos baseados em árvores de decisão*

**from** sklearn.tree **import** DecisionTreeClassifier

*# Classe para a criação de uma pipeline de machine-learning*

**from** sklearn.pipeline **import** Pipeline

​

*# Pacotes do scikit-learn para avaliação de modelos*

*# Métodos para validação cruzada do modelo criado*

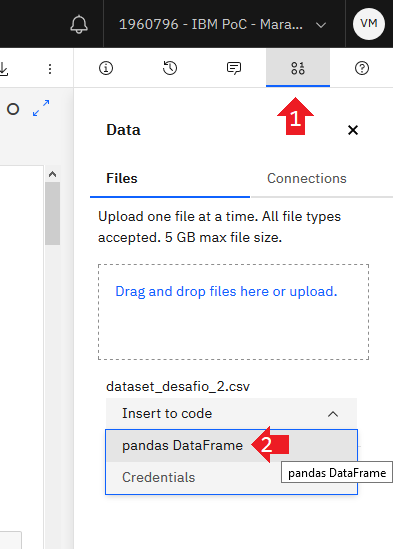
**from** sklearn.model\_selection **import** KFold, cross\_validate

### **Importando um .csv de seu projeto no IBM Cloud Pak for Data para o Kernel deste notebook**

Primeiro iremos importar o dataset fornecido para o desafio, que já está incluso neste projeto!

Você pode realizar a importação dos dados de um arquivo .csv diretamente para o Kernel do notebook como um DataFrame da biblioteca Pandas, muito utilizada para a manipulação de dados em Python.

Para realizar a importação, basta selecionar a próxima célula e seguir as instruções na imagem abaixo:



Após a seleção da opção **"Insert to code"**, a célula abaixo será preenchida com o código necessário para importação e leitura dos dados no arquivo .csv como um DataFrame Pandas.

In [ ]:

​

**<<** INSIRA O DATASET COMO UM PANDAS DATAFRAME NESTA CÉLULA**!** **>>>**

​

Temos 15 colunas presentes no dataset fornecido, sendo dezessete delas variáveis características (dados de entrada) e um delas uma variável-alvo (que queremos que o nosso modelo seja capaz de prever).

As variáveis características são:

MATRICULA - número de matrícula do estudante

NOME - nome completo do estudante

REPROVACOES\_DE - número de reprovações na disciplina de ``Direito Empresarial``

REPROVACOES\_EM - número de reprovações na disciplina de ``Empreendedorismo``

REPROVACOES\_MF - número de reprovações na disciplina de ``Matemática Financeira``

REPROVACOES\_GO - número de reprovações na disciplina de ``Gestão Operacional``

NOTA\_DE - média simples das notas do aluno na disciplina de ``Direito Empresarial`` (0-10)

NOTA\_EM - média simples das notas do aluno na disciplina de ``Empreendedorismo`` (0-10)

NOTA\_MF - média simples das notas do aluno na disciplina de ``Matemática Financeira`` (0-10)

NOTA\_GO - média simples das notas do aluno na disciplina de ``Gestão Operacional`` (0-10)

INGLES - variável binária que indica se o estudante tem conhecimento em língua inglesa (0 -> sim ou 1 -> não).

H\_AULA\_PRES - horas de estudo presencial realizadas pelo estudante

TAREFAS\_ONLINE - número de tarefas online entregues pelo estudante

FALTAS - número de faltas acumuladas do estudante (todas disciplinas)

A variável-alvo é:

PERFIL - uma \*string\* que indica uma de cinco possibilidades:

"EXCELENTE" - Estudante não necessita de mentoria

"MUITO BOM" - Estudante não necessita de mentoria

"HUMANAS" - Estudante necessita de mentoria exclusivamente em matérias com conteúdo de ciências humanas

"EXATAS" - Estudante necessita de mentoria apenas em disciplinas com conteúdo de ciências exatas

"DIFICULDADE" - Estudante necessita de mentoria em duas ou mais disciplinas

Com um modelo capaz de classificar um estudante em uma dessas categorias, podemos automatizar parte da mentoria estudantil através de assistentes virtuais, que serão capazes de recomendar práticas de estudo e conteúdo personalizado com base nas necessidades de cada aluno.

### **Explorando os dados fornecidos**

Podemos continuar a exploração dos dados fornecidos com a função info():

In [ ]:

df\_data\_1.info()

É notado que existem variáveis do tipo float64 (números "decimais"), variáveis do tipo int64 (números inteiros) e do tipo object (nesse caso são *strings*, ou texto).

Como a maioria dos algoritmos de aprendizado estatístico supervisionado só aceita valores numéricos como entrada, é necessário então o pré-processamento das variáveis do tipo "object" antes de usar esse dataset como entrada para o treinamento de um modelo. Também é notado que existem valores faltantes em várias colunas. Esses valores faltantes também devem ser tratados antes de serem construídos modelos com esse conjunto de dados base.

A função describe() gera várias informações sobre as variáveis numéricas que também podem ser úteis:

In [ ]:

df\_data\_1.describe()

### **Visualizações**

Para visualizar o dataset fornecido, podemos utilizar as bibliotecas matplotlib e seaborn:

In [ ]:

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**%**matplotlib inline

In [ ]:

fig, axes **=** plt.subplots(nrows**=**1, ncols**=**4, figsize**=**(28, 4))

​

sns.countplot(ax**=**axes[0], x**=**'REPROVACOES\_DE', data**=**df\_data\_1)

sns.countplot(ax**=**axes[1], x**=**'REPROVACOES\_EM', data**=**df\_data\_1)

sns.countplot(ax**=**axes[2], x**=**'REPROVACOES\_MF', data**=**df\_data\_1)

sns.countplot(ax**=**axes[3], x**=**'REPROVACOES\_GO', data**=**df\_data\_1)

In [ ]:

fig, axes **=** plt.subplots(nrows**=**1, ncols**=**4, figsize**=**(28, 4))

​

sns.distplot(df\_data\_1['NOTA\_DE'], ax**=**axes[0])

sns.distplot(df\_data\_1['NOTA\_EM'], ax**=**axes[1])

sns.distplot(df\_data\_1['NOTA\_MF'], ax**=**axes[2])

sns.distplot(df\_data\_1['NOTA\_GO'].dropna(), ax**=**axes[3])

In [ ]:

fig, axes **=** plt.subplots(nrows**=**1, ncols**=**4, figsize**=**(28, 4))

​

sns.countplot(ax**=**axes[0], x**=**'INGLES', data**=**df\_data\_1)

sns.countplot(ax**=**axes[1], x**=**'FALTAS', data**=**df\_data\_1)

sns.countplot(ax**=**axes[2], x**=**'H\_AULA\_PRES', data**=**df\_data\_1)

sns.countplot(ax**=**axes[3], x**=**'TAREFAS\_ONLINE', data**=**df\_data\_1)

In [ ]:

fig **=** plt.plot()

sns.countplot(x**=**'PERFIL', data**=**df\_data\_1)

### **Realizando o pré-processamento dos dados**

Para o pré-processamento dos dados serão apresentadas duas transformações básicas neste notebook, demonstrando a construção de uma Pipeline com um modelo funcional. Esta Pipeline funcional fornecida deverá ser melhorada pelo participante para que o modelo final alcance a maior acurácia possível, garantindo uma pontuação maior no desafio. Essa melhoria pode ser feita apenas no pré-processamento dos dados, na escolha de um algoritmo para treinamento de modelo diferente, ou até mesmo na alteração do *framework* usado (entretanto só será fornecido um exemplo pronto de integração do Watson Machine Learning com o *scikit-learn*).

A primeira transformação (passo na nossa Pipeline) será a exclusão da coluna "NOME" do nosso dataset, que além de não ser uma variável numérica, também não é uma variável relacionada ao desempenho dos estudantes nas disciplinas. Existem funções prontas no scikit-learn para a realização dessa transformação, entretanto nosso exemplo irá demonstrar como criar uma transformação personalizada do zero no scikit-learn. Se desejado, o participante poderá utilizar esse exemplo para criar outras transformações e adicioná-las à Pipeline final :)

#### **Transformação 1: excluindo colunas do dataset**

Para a criação de uma transformação de dados personalizada no scikit-learn, é necessária basicamente a criação de uma classe com os métodos transform e fit. No método transform será executada a lógica da nossa transformação.

Na próxima célula é apresentado o código completo de uma transformação DropColumns para a remoção de colunas de um DataFrame pandas.

In [ ]:

**from** sklearn.base **import** BaseEstimator, TransformerMixin

​

​

*# All sklearn Transforms must have the `transform` and `fit` methods*

**class** DropColumns(BaseEstimator, TransformerMixin):

**def** \_\_init\_\_(self, columns):

self.columns **=** columns

​

**def** fit(self, X, y**=None**):

**return** self

**def** transform(self, X):

*# Primeiro realizamos a cópia do dataframe 'X' de entrada*

data **=** X.copy()

*# Retornamos um novo dataframe sem as colunas indesejadas*

**return** data.drop(labels**=**self.columns, axis**=**'columns')

Para aplicar essa transformação em um DataFrame pandas, basta instanciar um objeto *DropColumns* e chamar o método transform().

In [ ]:

*# Instanciando uma transformação DropColumns*

rm\_columns **=** DropColumns(

columns**=**["NOME"] *# Essa transformação recebe como parâmetro uma lista com os nomes das colunas indesejadas*

)

​

print(rm\_columns)

In [ ]:

*# Visualizando as colunas do dataset original*

print("Colunas do dataset original: \n")

print(df\_data\_1.columns)

In [ ]:

*# Aplicando a transformação ``DropColumns`` ao conjunto de dados base*

rm\_columns.fit(X**=**df\_data\_1)

​

*# Reconstruindo um DataFrame Pandas com o resultado da transformação*

df\_data\_2 **=** pd.DataFrame.from\_records(

data**=**rm\_columns.transform(

X**=**df\_data\_1

),

)

In [ ]:

*# Visualizando as colunas do dataset transformado*

print("Colunas do dataset após a transformação ``DropColumns``: \n")

print(df\_data\_2.columns)

Nota-se que a coluna "NOME" foi removida e nosso dataset agora poossui apenas 17 colunas.

#### **Transformação 2: tratando dados faltantes**

Para tratar os dados faltantes em nosso conjunto de dados, iremos agora utilizar uma transformação pronta da biblioteca scikit-learn, chamada **SimpleImputer**.

Essa transformação permite diversas estratégias para o tratamento de dados faltantes. A documentação oficial pode ser encontrada em: <https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html>

Neste exemplo iremos simplesmente transformar todos os valores faltantes em zero.

In [ ]:

*# Criação de um objeto ``SimpleImputer``*

si **=** SimpleImputer(

missing\_values**=**np.nan, *# os valores faltantes são do tipo ``np.nan`` (padrão Pandas)*

strategy**=**'constant', *# a estratégia escolhida é a alteração do valor faltante por uma constante*

fill\_value**=**0, *# a constante que será usada para preenchimento dos valores faltantes é um int64=0.*

verbose**=**0,

copy**=True**

)

In [ ]:

*# Visualizando os dados faltantes do dataset após a primeira transformação (df\_data\_2)*

print("Valores nulos antes da transformação SimpleImputer: \n\n{}\n".format(df\_data\_2.isnull().sum(axis **=** 0)))

In [ ]:

*# Aplicamos o SimpleImputer ``si`` ao conjunto de dados df\_data\_2 (resultado da primeira transformação)*

si.fit(X**=**df\_data\_2)

​

*# Reconstrução de um novo DataFrame Pandas com o conjunto imputado (df\_data\_3)*

df\_data\_3 **=** pd.DataFrame.from\_records(

data**=**si.transform(

X**=**df\_data\_2

), *# o resultado SimpleImputer.transform(<<pandas dataframe>>) é lista de listas*

columns**=**df\_data\_2.columns *# as colunas originais devem ser conservadas nessa transformação*

)

In [ ]:

*# Visualizando os dados faltantes do dataset após a segunda transformação (SimpleImputer) (df\_data\_3)*

print("Valores nulos no dataset após a transformação SimpleImputer: \n\n{}\n".format(df\_data\_3.isnull().sum(axis **=** 0)))

Nota-se que não temos mais nenhum valor faltante no nosso conjunto de dados :)

Vale salientar que nem sempre a alteração dos valores faltantes por 0 é a melhor estratégia. O participante é incentivado a estudar e implementar estratégias diferentes de tratamento dos valores faltantes para aprimorar seu modelo e melhorar sua pontuação final.

### **Treinando um modelo de classificação**

Finalizado o pré-processamento, já temos o conjunto de dados no formato necessário para o treinamento do nosso modelo:

In [ ]:

df\_data\_3.head()

No exemplo fornecido, iremos utilizar todas as colunas, exceto a coluna **LABELS** como *features* (variáveis de entrada).

A variável **LABELS** será a variável-alvo do modelo, conforme descrito no enunciado do desafio.

#### **Definindo as features do modelo**

In [ ]:

*# Definição das colunas que serão features (nota-se que a coluna NOME não está presente)*

features **=** [

"MATRICULA", 'REPROVACOES\_DE', 'REPROVACOES\_EM', "REPROVACOES\_MF", "REPROVACOES\_GO",

"NOTA\_DE", "NOTA\_EM", "NOTA\_MF", "NOTA\_GO",

"INGLES", "H\_AULA\_PRES", "TAREFAS\_ONLINE", "FALTAS",

]

​

*# Definição da variável-alvo*

target **=** ["PERFIL"]

​

*# Preparação dos argumentos para os métodos da biblioteca ``scikit-learn``*

X **=** df\_data\_3[features]

y **=** df\_data\_3[target]

O conjunto de entrada (X):

In [ ]:

X.head()

As variáveis-alvo correspondentes (y):

In [ ]:

y.head()

#### **Separando o dataset em um conjunto de treino e um conjunto de teste**

Iremos separar o dataset fornecido em dois grupos: um para treinar nosso modelo, e outro para testarmos o resultado através de um teste cego. A separação do dataset pode ser feita facilmente com o método *train\_test\_split()* do scikit-learn:

In [ ]:

*# Separação dos dados em um conjunto de treino e um conjunto de teste*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.3, random\_state**=**337)

#### **Criando um modelo baseado em árvores de decisão**

No exemplo fornecido iremos criar um classificador baseado em **árvores de decisão**.

O primeiro passo é basicamente instanciar um objeto *DecisionTreeClassifier()* da biblioteca scikit-learn.

In [ ]:

*# Criação da árvore de decisão com a biblioteca ``scikit-learn``:*

dtc\_model **=** DecisionTreeClassifier() *# O modelo será criado com os parâmetros padrões da biblioteca*

Material teórico sobre árvores de decisão na documentação oficial do scikit-learn: <https://scikit-learn.org/stable/modules/tree.html>

Um guia para iniciantes no mundo do machine learning: <https://developer.ibm.com/br/articles/cc-beginner-guide-machine-learning-ai-cognitive/>

#### **Execução do evento de treino de uma árvore de decisão**

In [ ]:

*# Treino do modelo (é chamado o método \*fit()\* com os conjuntos de treino)*

dtc\_model.fit(

X\_train,

y\_train

)

#### **Execução de predições e avaliação do modelo criado**

In [ ]:

*# Realização de teste cego no modelo criado*

y\_pred **=** dtc\_model.predict(X\_test)

In [ ]:

X\_test.head()

In [ ]:

print(y\_pred)

In [ ]:

**from** sklearn.metrics **import** accuracy\_score

​

*# Acurácia alcançada pela árvore de decisão*

print("Acurácia: {}%".format(100**\***round(accuracy\_score(y\_test, y\_pred), 2)))

Neste notebook foi demonstrado como trabalhar com transformações e modelos com a biblioteca scikit-learn. É recomendado que o participante realize seus experimentos editando o código fornecido aqui até que um modelo com acurácia elevada seja alcançado.

Quando você estiver satisfeito com seu modelo, pode passar para a segunda etapa do desafio -- encapsular seu modelo como uma API REST pronta para uso com o Watson Machine Learning!

O notebook para a segunda etapa já se encontra neste projeto, basta acessar a aba **ASSETS** e inicializá-lo! Não se esqueca de antes desligar o Kernel deste notebook para reduzir o consumo de sua camada grátis do IBM Cloud Pak for Data.

In [ ]:

​