

\_PRÉ-PROCESSAMENTO: ONEHOTENCODER

*#transforma cada valor único em uma coluna categórica em um novo conjunto de colunas binárias, onde cada coluna representa a presença ou ausência desse valor em uma linha de dados.*



```
> from sklearn.preprocessing import OneHotEncoder

> colunas_categoricas = ['Migração', 'Sexo','Estrangeiro', 'Necessidades educacionais especiais', 'Devedor','Taxas de matrícula em dia', 'Bolsista', 'Período','Estado civil', 'Curso', 'Qualificação prévia']

#Selecionando apenas as colunas categóricas do dataframe
> df_categorico = df[colunas_categoricas]

#Inicializando o OneHotEncoder
> encoder = OneHotEncoder(drop='if_binary')

# Ajustando e transformando os dados, criando um novo dataframe com as colunas codificadas
> df_encoded = pd.DataFrame(encoder.fit_transform(df_categorico).toarray(), columns=encoder.get_feature_names_out(colunas_categoricas))

#Combinando as colunas codificadas com as colunas não codificadas do dataframe original
> df_final = pd.concat([df.drop(colunas_categoricas, axis=1), df_encoded], axis=1)
```

\_DIVISÃO DOS DADOS

```
#Realiza a divisão dos dados entre treino, validação e teste de forma estratificada
> X, X_teste, y, y_teste = train_test_split(X, y, test_size=0.15, stratify=y, random_state=0)

> X_treino, X_val, y_treino, y_val = train_test_split(X, y, stratify=y, random_state=0)
```

CONJUNTO DE DADOS



\_CRIANDO UM MODELO

```
#Importa o algoritmo Random Forest Classifier
> from sklearn.tree import RandomForestClassifier

#Inicializa um modelo de árvore de decisão com profundidade máxima de 10
> modelo = RandomForestClassifier(max_depth = 10)
```

 clique nos ícones para **acessar** as respectivas **documentações**

## \_AJUSTANDO UM MODELO

*#Ajusta um modelo usando dados de treinamento*



```
> modelo.fit(X_treino, y_treino)
```

## \_AVALIANDO UM MODELO

*#Avalia a taxa de acerto do modelo usando dados de validação*



```
> modelo.score(X_val, y_val)
```

## \_FAZENDO PREVISÕES

*# Fazendo previsões do modelo a partir dos dados de validação*



```
> y_pred = modelo.predict(X_val)
```

## \_MATRIZ DE CONFUSÃO

*#Importa a função para criação da matriz de confusão*



```
> from sklearn.metrics import confusion_matrix
```

*#Importa a função para criação da visualização da matriz de confusão*



```
> from sklearn.metrics import ConfusionMatrixDisplay
```

*#Gerando uma matriz de confusão*

```
> matriz_confusao = confusion_matrix(y_val, y_pred)
```

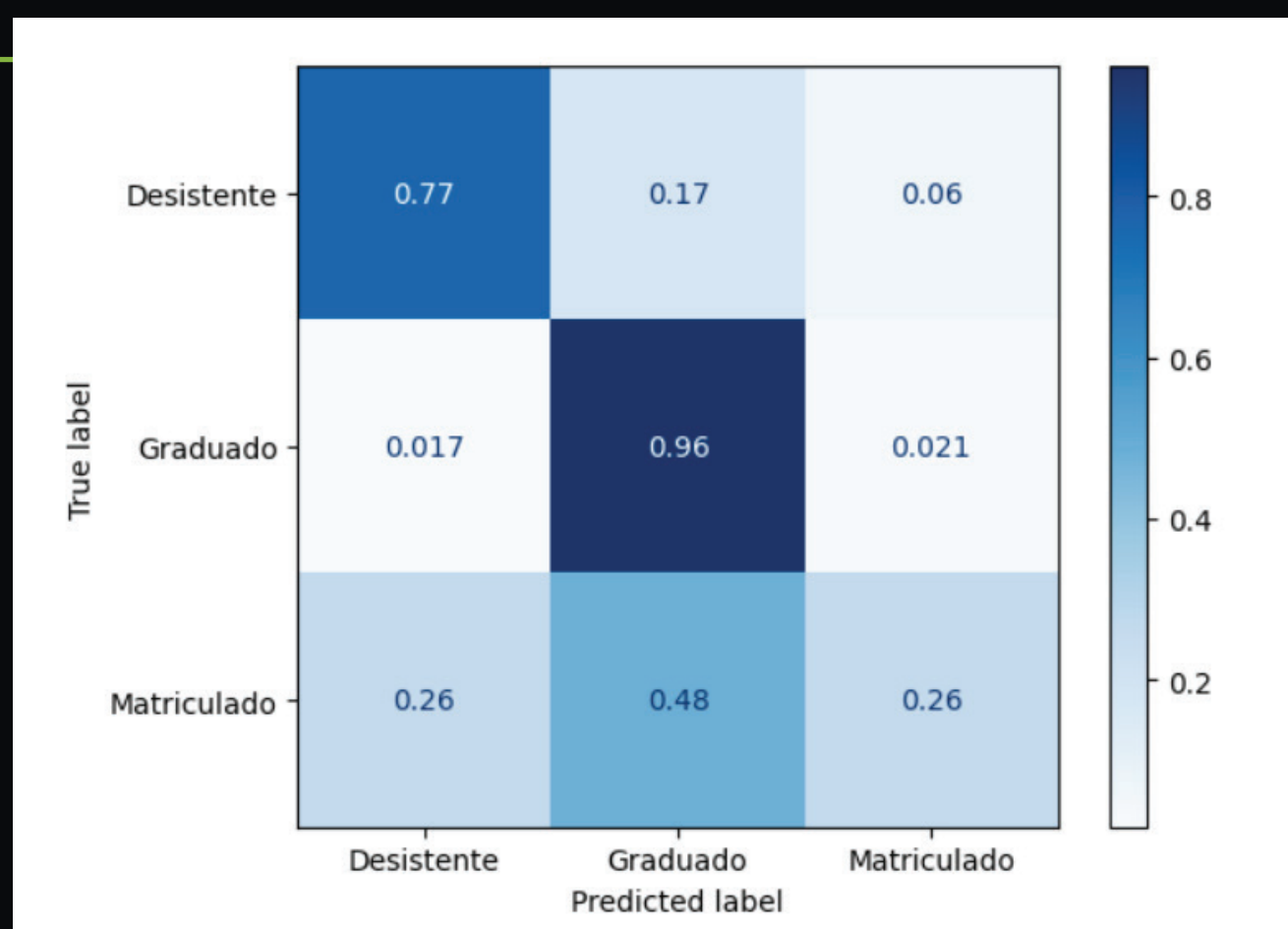
```
> print(matriz_confusao)
```

*#Gerando a visualização a partir de uma matriz de confusão*

```
> ConfusionMatrixDisplay(confusion_matrix = matriz_confusao,  
    display_labels=modelo.classes_ )
```

*#Gerando a visualização da matriz de confusão a partir dos dados reais e previstos, com os valores normalizados e uma paleta de cores em tons de azul*

```
> ConfusionMatrixDisplay.from_predictions(y_val, y_pred, normalize = 'true',  
    cmap = 'Blues');
```



clique nos ícones para **acessar** as respectivas **documentações**

## \_MÉTRICAS DE DESEMPENHO

*# importa a função para gerar o relatório de métricas de classificação* 

```
> from sklearn.metrics import classification_report
```

*#Gera o relatório de métricas de classificação usando os dados reais e previstos*

```
> classification_report(y_val, y_previsto)
```

*# A acurácia mede a proporção de predições corretas (verdadeiros positivos e verdadeiros negativos) em relação ao total de amostras. É uma métrica geral que avalia o desempenho global do modelo, mas pode ser enganosa em problemas de desequilíbrio de classe.*

$$\text{Acurácia} = \frac{\checkmark \text{ VP} + \checkmark \text{ VN}}{\checkmark \text{ VP} + \checkmark \text{ VN} + \otimes \text{ FP} + \otimes \text{ FN}}$$
 


*# A precisão é a proporção de previsões corretas entre todas as previsões positivas feitas pelo modelo. Em outras palavras, ela mede a qualidade das previsões positivas.*

$$\text{Precisão} = \frac{\checkmark \text{ VP}}{\checkmark \text{ VP} + \otimes \text{ FP}}$$
 


*# O recall é a proporção de previsões corretas entre todos os exemplos positivos reais. Ela mede a capacidade do modelo de identificar todos os exemplos positivos.*

$$\text{Recall} = \frac{\checkmark \text{ VP}}{\checkmark \text{ VP} + \otimes \text{ FN}}$$
 

*# O F1-Score é a média harmônica da precisão e o recall, fornecendo um equilíbrio entre essas duas métricas.*

$$\text{F1} = 2 * \frac{\text{precisão} * \text{recall}}{\text{precisão} + \text{recall}}$$
 

## \_BALANCEAMENTO DE DADOS

*#Importa a função de oversampling com SMOTE* 

```
> from imblearn.over_sampling import SMOTE
```

*#Inicializa e faz o oversampling de dados com SMOTE*

```
> oversample = SMOTE(random_state=0)
> X_balanceado, y_balanceado = oversample.fit_resample(X_treino, y_treino)
```

 clique nos ícones para **acessar** as respectivas **documentações**

## \_PIPELINE DE DADOS

*#Importa a função de criação de pipelines* 

```
> from imblearn.pipeline import Pipeline as imbpipeline
```

*#Cria um pipeline de oversampling com SMOTE e um modelo de ML*

```
> pipeline = imbpipeline([('oversample', SMOTE()), ('modelo', modelo)])
```

## \_VALIDAÇÃO CRUZADA



*# importa a função do StratifiedKFold* 

```
> from sklearn.model_selection import StratifiedKFold
```

*# inicializa o KFold estratificado com 5 folds e embaralhamento dos dados*

```
> skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
```

*# importa a função de validação cruzada para resultados das métricas e tempo de execução*

```
> from sklearn.model_selection import cross_validate 
```

*# realiza a validação cruzada para resultados das métricas*

```
> cv_resultados = cross_validate(pipeline, X, y, cv=skf,  
                                scoring='recall_weighted')
```

```
> cv_resultados['test_score']
```

 clique nos ícones para **acessar** as respectivas **documentações**