PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Ulisses Benedito Júnior

Previsão da qualidade de vinhos através de técnicas de Machine Learning

Belo Horizonte
2023

Ulisses Benedito Júnior

Previsão da qualidade de vinhos através de técnicas de Machine Learning

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte
2023

SUMÁRIO

1. Introdução

1.1. Contextualização

Após a pandemia da COVID-19 e o isolamento social, houve um aumento significativo na apreciação e consumo de vinhos. Esse cenário trouxe uma nova demanda para importadoras, lojas e comerciantes de vinhos: a necessidade de disponibilizar uma seleção de vinhos de qualidade para atender aos desejos dos consumidores.

No entanto, identificar e adquirir vinhos de qualidade pode ser um desafio para esses profissionais. A qualidade do vinho é influenciada por diversos fatores, como a região de produção, as práticas de vinificação e as características físico-químicas da bebida. Além disso, a preferência dos consumidores pode variar amplamente, tornando necessário considerar uma série de informações para selecionar os vinhos adequados ao mercado.

Nesse contexto, surge a necessidade de uma ferramenta de apoio confiável e prática que auxilie importadoras, lojas e comerciantes de vinhos na qualificação dos produtos. Essa ferramenta deve ser capaz de analisar as propriedades físico-químicas dos vinhos e prever sua qualidade, fornecendo informações objetivas para embasar as decisões de compra e venda.

O objetivo deste estudo é desenvolver um modelo preditivo da qualidade do vinho com base em técnicas de Machine Learning e análise de dados. Através da aplicação de algoritmos como Random Forest, XGBoost e Regressão Logística, busca-se criar um modelo capaz de avaliar a qualidade do vinho com base em características como acidez, teor de açúcar, pH, entre outras.

A partir desse modelo, importadoras, lojas e comerciantes de vinhos terão uma ferramenta confiável e prática para auxiliá-los na seleção de vinhos de qualidade para seus clientes. Além disso, essa abordagem contribui para a

valorização do conhecimento técnico-científico no mercado de vinhos, permitindo uma análise mais precisa e fundamentada na escolha dos produtos.

Dessa forma, o desenvolvimento desse modelo preditivo atende à necessidade do mercado de importadoras, lojas e comerciantes de vinhos, fornecendo uma ferramenta de apoio confiável e prática para qualificar e auxiliar na aquisição de bons vinhos.

1.2. O problema proposto

Nesta seção, será apresentado o problema que será abordado neste estudo, combinado aos objetivos e as técnicas de Machine Learning utilizadas. O objetivo principal é desenvolver um modelo preditivo da qualidade do vinho com base em suas características físico-químicas, utilizando técnicas avançadas de análise de dados e Machine Learning.

Inicialmente, os dados serão coletados a partir de fontes confiáveis e representativas, contendo informações sobre as propriedades físico-químicas de diferentes vinhos, acompanhado de suas avaliações de qualidade. Será realizada uma análise exploratória desses dados, a fim de compreender suas características, identificar possíveis padrões e insights relevantes.

Durante o processo de análise e exploração dos dados, serão aplicadas técnicas de pré-processamento e tratamento dos dados. Isso incluirá a exclusão de registros com dados nulos ou faltantes, assim como a detecção e eliminação de outliers que possam afetar negativamente a qualidade do modelo. Essas etapas são essenciais para garantir a qualidade e confiabilidade dos dados utilizados na construção dos modelos de Machine Learning.

Posteriormente, serão implementados três modelos de Machine Learning: Random Forest (Floresta Aleatória), XGBoost e Regressão Logística. Esses modelos foram selecionados por sua capacidade de lidar com problemas de classificação e pela sua eficácia em lidar com conjuntos de dados complexos. Cada modelo será treinado e ajustado utilizando técnicas de validação cruzada e otimização de hiperparâmetros, a fim de obter os melhores resultados de predição da qualidade do vinho.

Por fim, os modelos serão avaliados utilizando métricas apropriadas, como acurácia, precisão, recall e F1-score, para determinar sua capacidade de prever a qualidade do vinho com base nas características físico-químicas. A análise dos resultados obtidos fornecerá insights valiosos sobre a relação entre as propriedades físico-químicas e a qualidade do vinho, permitindo uma melhor compreensão dos fatores que influenciam sua apreciação.

Com a conclusão deste estudo, espera-se fornecer um modelo preditivo robusto e confiável para estimar a qualidade do vinho com base em suas características físico-químicas. Isso pode ser útil para produtores, enólogos e consumidores na seleção e avaliação de vinhos, contribuindo para uma experiência mais informada e satisfatória.

2. Coleta de Dados

Neste estudo, são utilizados dois conjuntos de dados relacionados com as variantes tinto e branco. Esses conjuntos de dados foram disponibilizados publicamente para fins de pesquisa e suas propriedades físico-químicas foram analisadas com o objetivo de prever a qualidade do vinho.

Os conjuntos de dados utilizados são: winequality-red.csv: Este arquivo contém os dados das propriedades físico-químicas dos vinhos tintos; e winequality-

white.csv: Este arquivo contém os dados das propriedades físico-químicas dos vinhos brancos.

Ambos os conjuntos de dados foram disponibilizados por P. Cortez, A. Cerdeira, F. Almeida, T. Matos e J. Reis em seu estudo intitulado "Modelagem de preferências de vinho por mineração de dados de propriedades físico-químicas", publicado na revista Decision Support Systems, da Elsevier. Os detalhes completos dos conjuntos de dados podem ser encontrados no link da referência [Elsevier] (http://dx.doi.org/10.1016/j.dss.2009.05.016). Também está disponível uma pré-impressão do artigo em formato PDF Pré-impressão (pdf), bem como a citação bibliográfica [bib] (http://www3.dsi.uminho.pt/pcortez/dss09.bib).

A tabela a seguir apresenta os nomes das colunas/campos, suas descrições e os tipos de dados correspondentes:

Nome da coluna/campo	Descrição	Tipo
fixed acidity	Acidez fixa - diferença entre a acidez total e a acidez volátil.	float64
volatile acidity	Acidez volátil - ácido acético que é formado na fermentação alcoólica e, em dose elevada, origina o aroma a vinagre.	float64
citric acid	Ácido cítrico - é um ácido orgânico forte, normalmente presente em fracas quantidades nos mostos de uva e geralmente ausente nos vinhos.	float64
residual sugar	Açúcar residual - açúcar que sobra após terminar a fermentação alcoólica.	float64
chlorides	Cloretos - indica a presença de sal no vinho.	float64
free sulfur dioxide	Dióxido de enxofre livre - é uma medida da quantidade de dióxido de enxofre que não está ligado a outras moléculas.	float64
total sulfur dioxide	Dióxido de enxofre total - é uma médida de ambos os dióxidos de enxofre, livre e ligado a outras moléculas.	float64
density	Densidade.	float64
pH	pH.	float64
sulphates	Sulfatos - indica a presença de sais de ácido sulfúrico.	float64

alcohol	Álcool.	float64
quality	Qualidade - indica a qualidade do vinho de 0 a 10, sendo 0 a pior nota e 10 a melhor.	int64

Essas propriedades físico-químicas do vinho serão utilizadas como variáveis de entrada para análise e modelagem, enquanto a qualidade do vinho será a variável de saída a ser prevista pelos modelos de Machine Learning.

No próximo tópico, serão apresentadas as etapas de pré-processamento, análise exploratória e modelagem dos dados.

3. Processamento/Tratamento de Dados

Passo 1: Importando os pacotes necessários e os dados

Neste passo, os pacotes necessários são importados e os conjuntos de dados de vinhos tintos e brancos são carregados.

Importando os pacotes necessários e os dados

```
# Importando as bibliotecas
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Importando as bibliotecas para a construção dos modelos de Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier

from imblearn.under_sampling import RandomUnderSampler

from yellowbrick.classifier import ROCAUC

# import warnings filter
from warnings import simplefilter

# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)

# Configurando o notebook
%matplotlib inline
sns.set(style='white')



# Carregar os dados dos vinhos tintos
dados_vinhos_tintos = pd.read_csv("winequality-red.csv")
dados_vinhos_tintos["Tipo"] = 1  # Adicionar coluna "Tipo" para identificar o tipo de vinho 1 = Tinto

# Carregar os dados dos vinhos brancos
dados_vinhos_brancos = pd.read_csv("winequality-white.csv")
dados_vinhos_brancos["Tipo"] = 0  # Adicionar coluna "Tipo" para identificar o tipo de vinho 0 = Branco
```

Neste passo, são importados os pacotes necessários, como pandas, matplotlib.pyplot, seaborn, numpy, entre outros. Além disso, as bibliotecas específicas para a construção dos modelos de Machine Learning, como RandomForestClassifier, LogisticRegression e XGBClassifier, também são importadas. São importadas as bibliotecas para a avaliação dos modelos, como classification_report, confusion_matrix e roc_auc_score. A biblioteca imblearn é utilizada para realizar o balanceamento dos dados e a biblioteca yellowbrick para a visualização da curva ROC-AUC. São feitas algumas configurações, como a configuração do notebook para exibir as visualizações inline e a definição do estilo do seaborn.

Os dados dos vinhos tintos e brancos são carregados em dataframes separados utilizando a função pd.read_csv(). Em seguida, é adicionada uma coluna chamada "Tipo" para identificar o tipo de vinho, sendo 1 para os vinhos tintos e 0 para os vinhos brancos.

Este passo é essencial para importar os pacotes necessários e carregar os dados que serão utilizados nos próximos passos de processamento e análise dos dados.

Passo 2: Visualização dos primeiros registros dos dados

Neste passo, visualizaremos os primeiros registros dos conjuntos de dados dos vinhos tintos e brancos.

```
print(dados_vinhos_tintos.head())
print(dados_vinhos_brancos.head())
```

```
   fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"density
0   7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5
1   7.8;0.88;0;2.6;0.098;25;67;0.9968;3.2;0.68;9.8;5
2  7.8;0.76;0.04;2.3;0.092;15;54;0.997;3.26;0.65;...
3  11.2;0.28;0.56;1.9;0.075;17;60;0.998;3.16;0.58...
4   7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5

   Tipo
0     1
1     1
2     1
3     1
4     1
  fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"density
0   7;0.27;0.36;20.7;0.045;45;170;1.001;3;0.45;8.8;6
1  6.3;0.3;0.34;1.6;0.049;14;132;0.994;3.3;0.49;9...
2  8.1;0.28;0.4;6.9;0.05;30;97;0.9951;3.26;0.44;1...
3  7.2;0.23;0.32;8.5;0.058;47;186;0.9956;3.19;0.4...
4  7.2;0.23;0.32;8.5;0.058;47;186;0.9956;3.19;0.4...

   Tipo
0     0
1     0
2     0
3     0
4     0
```

Ao executar este passo, serão exibidos os primeiros registros dos dataframes dados_vinhos_tintos e dados_vinhos_brancos, que contêm as seguintes colunas: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, quality e Tipo.

Essa etapa permite visualizar uma amostra dos dados de cada conjunto, ajudando a entender a estrutura e os valores presentes nas colunas. É útil para identificar possíveis problemas de formatação ou valores ausentes antes de prosseguir com o processamento dos dados.

Passo 3: Junção dos datasets de vinhos tintos e brancos

```
# Junção dos datasets de vinhos tintos e brancos
dados_vinhos = pd.concat([dados_vinhos_tintos, dados_vinhos_brancos], ignore_index=True)

# Verificar a quantidade de registros no dataset unificado
print("Quantidade de registros de vinhos:", dados_vinhos.shape[0])
```

```
Quantidade de registros de vinhos: 6497
```

Neste passo, realizaremos a junção dos datasets de vinhos tintos e brancos em um único dataframe chamado dados_vinhos. Em seguida, verificaremos a quantidade de registros presentes no dataset unificado.

Ao executar este passo, os datasets de vinhos tintos e brancos serão concatenados em um único dataframe chamado dados_vinhos, utilizando o método concat da biblioteca Pandas. O parâmetro ignore_index=True é utilizado para reindexar os registros após a concatenação.

Em seguida, será exibida a quantidade de registros presentes no dataset unificado, permitindo ter uma visão geral do tamanho do conjunto de dados.

No dataframe apresentado, a quantidade de registros de vinhos é 6497. Essa informação é útil para compreender o tamanho do dataset e a quantidade de amostras disponíveis para análise e modelagem de dados.

Passo 4: Visualizar a estrutura do DataFrame

Neste passo, visualizaremos a estrutura do DataFrame dados_vinhos para entender melhor as colunas presentes, o tipo de dado de cada coluna e a quantidade de valores não nulos.

```
# Visualizar a estrutora do DataFrame
dados_vinhos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 2 columns):
 #   Column
---  ------
 0   fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"dens
 1   Tipo
dtypes: int64(1), object(1)
memory usage: 101.6+ KB
```

Ao executar este passo, será exibida uma descrição da estrutura do DataFrame dados_vinhos. Essa informação inclui o tipo de dado de cada coluna, a quantidade de valores não nulos e a quantidade total de registros.

No DataFrame apresentado, temos um com 6497 registros. O DataFrame possui duas colunas: “fixed acidity”; ”volatile acidity"; "citric acid"; "residual sugar"; "chlorides"; "free sulfur dioxide"; "total sulfur dioxide"; "density"; "pH"; "sulphates"; "alcohol"; "quality", que contém informações sobre as características dos vinhos, e “Tipo”, que indica o tipo de vinho (0 para branco e 1 para tinto).

É importante observar os tipos de dados das colunas, pois isso pode influenciar as etapas de processamento e análise dos dados. No exemplo apresentado, a coluna "fixed acidity; "volatile acidity"; "citric acid"; "residual sugar"; "chlorides"; "free sulfur dioxide"; "total sulfur dioxide"; "density"; "pH"; "sulphates"; "alcohol"; "quality" é do tipo object, enquanto a coluna "Tipo" é do tipo int64.

Passo 5: Separar e renomear as colunas corretamente

Neste passo, iremos separar a coluna "fixed acidity;"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfur dioxide";"total sulfur dioxide";"density";"pH";"sulphates";"alcohol";"quality"" em colunas individuais usando o ponto e vírgula como separador. Em seguida, renomearemos as colunas para melhorar a legibilidade e compreensão dos dados.

```
# Separar as colunas corretamente usando o ponto e vírgula como separador
vinhos_total = dados_vinhos["fixed acidity;\"volatile acidity\";\"citric acid\";\"residual sugar\";\"chlorides\";\"free sulfur dioxide\";

# Renomear as colunas
vinhos_total.columns = [
    "acidez fixa",
    "acidez volátil",
    "ácido cítrico",
    "açúcar residual",
    "cloretos",
    "dióxido de enxofre livre",
    "dióxido de enxofre total",
    "densidade",
    "pH",
    "sulfatos",
    "álcool",
    "qualidade"
]

# Adicionar a coluna "Tipo" aos dados separados
vinhos_total["tipo"] = dados_vinhos["Tipo"]
```

Após executar este passo, teremos o DataFrame `vinhos_total`, que contém as colunas separadas e renomeadas corretamente. As colunas representam as seguintes características dos vinhos:

- "acidez fixa": nível de acidez fixa do vinho
- "acidez volátil": nível de acidez volátil do vinho
- "ácido cítrico": quantidade de ácido cítrico presente no vinho
- "açúcar residual": quantidade de açúcar residual no vinho
- "cloretos": quantidade de cloretos no vinho
- "dióxido de enxofre livre": quantidade de dióxido de enxofre livre no vinho
- "dióxido de enxofre total": quantidade total de dióxido de enxofre no vinho
- "densidade": densidade do vinho
- "pH": nível de pH do vinho
- "sulfatos": quantidade de sulfatos no vinho
- "álcool": teor alcoólico do vinho
- "qualidade": qualidade do vinho
- "tipo": indica o tipo de vinho (0 para branco e 1 para tinto)

Essa separação e renomeação das colunas facilitará a análise e o processamento dos dados posteriormente.

Passo 6: Visualizar as primeiras entradas e dimensões do conjunto de dados

Neste passo, iremos exibir as dimensões do conjunto de dados, ou seja, o número de linhas e colunas do DataFrame vinhos_total. Além disso, mostraremos as primeiras entradas do conjunto de dados para ter uma visão inicial dos dados.

▾ Primeiras entradas e dimensões do conjunto de dados

```
# Dimensões do dataset
print("Dimensões do conjunto de dados:\n{} linhas e {} colunas\n".format(vinhos_total.shape[0], vinhos_total.shape[1]))

# Primeiras entradas do dataset
print("Primeiras entradas:")
vinhos_total.head()
```

A saída do código será:

```
Primeiras entradas:
```

	acidez fixa	acidez volátil	ácido cítrico	açúcar residual	cloretos	dióxido de enxofre livre	dióxido de enxofre total	densidade	pH	sulfatos	álcool	qualidade	tipo
0	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5	1
1	7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5	1
2	7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5	1
3	11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6	1
4	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5	1

4. Análise e Exploração dos Dados

Nesta seção, exploraremos e analisar os dados do dataframe vinhos_total, que contém informações sobre vinhos tintos e brancos. Além disso, faremos a separação do dataframe em dois conjuntos de dados: dados_base_tinto, contendo apenas os vinhos tintos, e dados_base_branco, contendo apenas os vinhos brancos. Essa separação será feita com o objetivo de analisar os dois tipos de vinho de forma separada, considerando que algumas variáveis podem se comportar de maneira diferente para cada tipo.

4.1. Amostra dos Dados

Vamos começar dando uma olhada nas primeiras entradas do dataframe vinhos_total, a fim de entender sua estrutura e as variáveis presentes.

Amostra dos dados do dataframe vinhos_total:

```
print("Nome dos atributos:\n{}".format(vinhos_total.columns.values))
print("\nQuantidade de valores ausentes por atributo:\n{}".format(vinhos_total.isnull().sum()))
print("\nTipo de cada atributo:\n{}".format(vinhos_total.dtypes))
```

```
Nome dos atributos:
['acidez fixa' 'acidez volátil' 'ácido cítrico' 'açúcar residual'
 'cloretos' 'dióxido de enxofre livre' 'dióxido de enxofre total'
 'densidade' 'pH' 'sulfatos' 'álcool' 'qualidade' 'tipo']
```

```
Quantidade de valores ausentes por atributo:
acidez fixa                 0
acidez volátil              0
ácido cítrico               0
açúcar residual             0
cloretos                    0
dióxido de enxofre livre    0
dióxido de enxofre total    0
densidade                   0
pH                          0
sulfatos                    0
álcool                      0
qualidade                   0
tipo                        0
dtype: int64
```

```
Tipo de cada atributo:
acidez fixa                 object
acidez volátil              object
ácido cítrico               object
açúcar residual             object
cloretos                    object
dióxido de enxofre livre    object
dióxido de enxofre total    object
densidade                   object
pH                          object
sulfatos                    object
álcool                      object
qualidade                   object
tipo                         int64
dtype: object
```

```
#Verificar presença de nulos
vinhos_total.isnull().any()
```

```
acidez fixa                 False
acidez volátil              False
ácido cítrico               False
açúcar residual             False
cloretos                    False
dióxido de enxofre livre    False
dióxido de enxofre total    False
densidade                   False
pH                          False
sulfatos                    False
álcool                      False
qualidade                   False
tipo                        False
dtype: bool
```

```
# Converter colunas relevantes para tipos numéricos
colunas_numericas = ['acidez fixa', 'acidez volátil', 'ácido cítrico', 'açúcar residual', 'cloretos', 'dióxido de enxofre livre',
vinhos_total[colunas_numericas] = vinhos_total[colunas_numericas].astype(float)

print("\nTipo de cada atributo:\n{}".format(vinhos_total.dtypes))
```

```
Tipo de cada atributo:
acidez fixa                 float64
acidez volátil              float64
ácido cítrico               float64
açúcar residual             float64
cloretos                    float64
dióxido de enxofre livre    float64
dióxido de enxofre total    float64
densidade                   float64
pH                          float64
sulfatos                    float64
álcool                      float64
qualidade                   float64
tipo                          int64
dtype: object
```

4.2. Estatísticas Descritivas

Agora, calcularemos algumas estatísticas descritivas para as variáveis numéricas do dataframe vinhos_total. Isso nos fornecerá uma visão geral sobre a distribuição dos dados.

Estatísticas descritivas do dataframe vinhos_total:

```
vinhos_total.describe()
```

	acidez fixa	acidez volátil	ácido cítrico	açúcar residual	cloretos	dióxido de enxofre livre	dióxido de enxofre total	densidade	pH	sulfatos	álcool	qua
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.0
mean	7.215307	0.339666	0.318633	5.443235	0.056034	30.525319	115.744574	0.994697	3.218501	0.531268	10.491801	5.8
std	1.296434	0.164636	0.145318	4.757804	0.035034	17.749400	56.521855	0.002999	0.160787	0.148806	1.192712	0.8
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.0
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	5.0
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.0
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000	11.300000	6.0
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.0

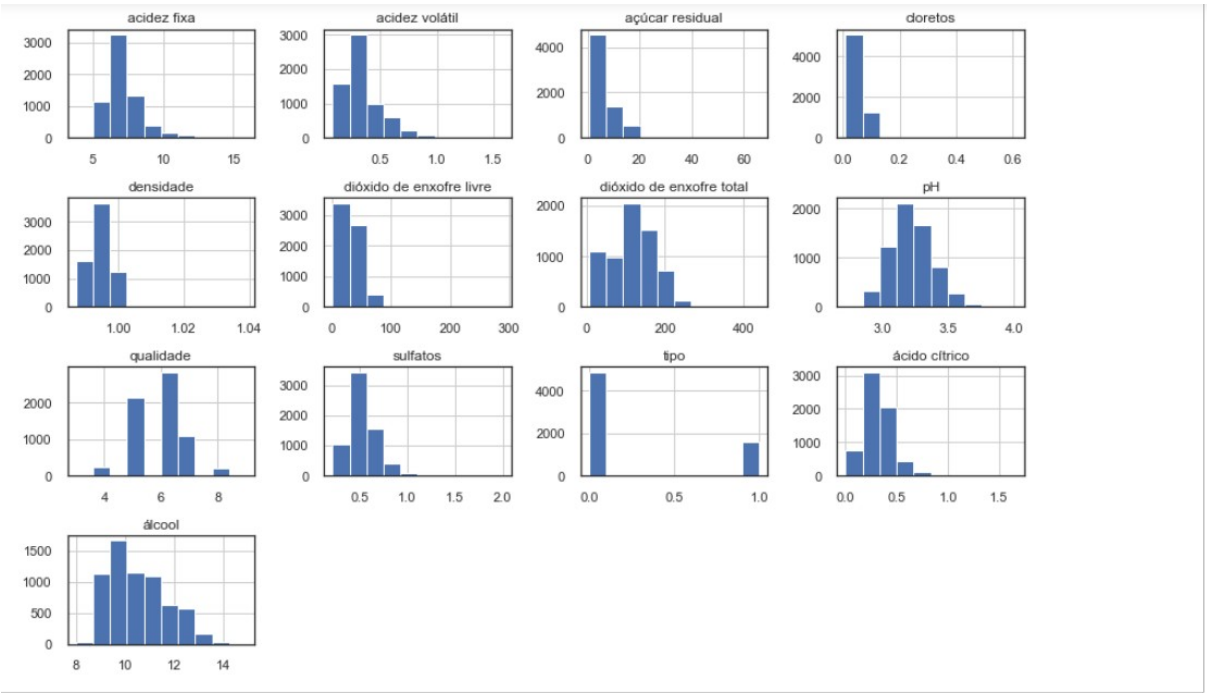
```
# Categorizando a variável qualidade e criando a variável qualidade_cat
vinhos_total['qualidade_cat'] = pd.cut(vinhos_total['qualidade'], bins=(2, 6.5, 8), labels = [0, 1])
```

```
# Verificando quais as entradas únicas da variável qualidade_cat
vinhos_total['qualidade_cat'] = vinhos_total['qualidade_cat'].astype('category')
vinhos_total['qualidade_cat'].unique()
```

```
[0, 1, NaN]
Categories (2, int64): [0 < 1]
```

Vamos verificar e tratar dos outliers, no gral, a verificação e o tratamento dos outliers são etapas importantes na análise de dados, pois contribuem para a melhoria da qualidade, a confiabilidade dos resultados e a interpretação correta das informações. Isso possibilita uma tomada de decisão mais informada e embasada.

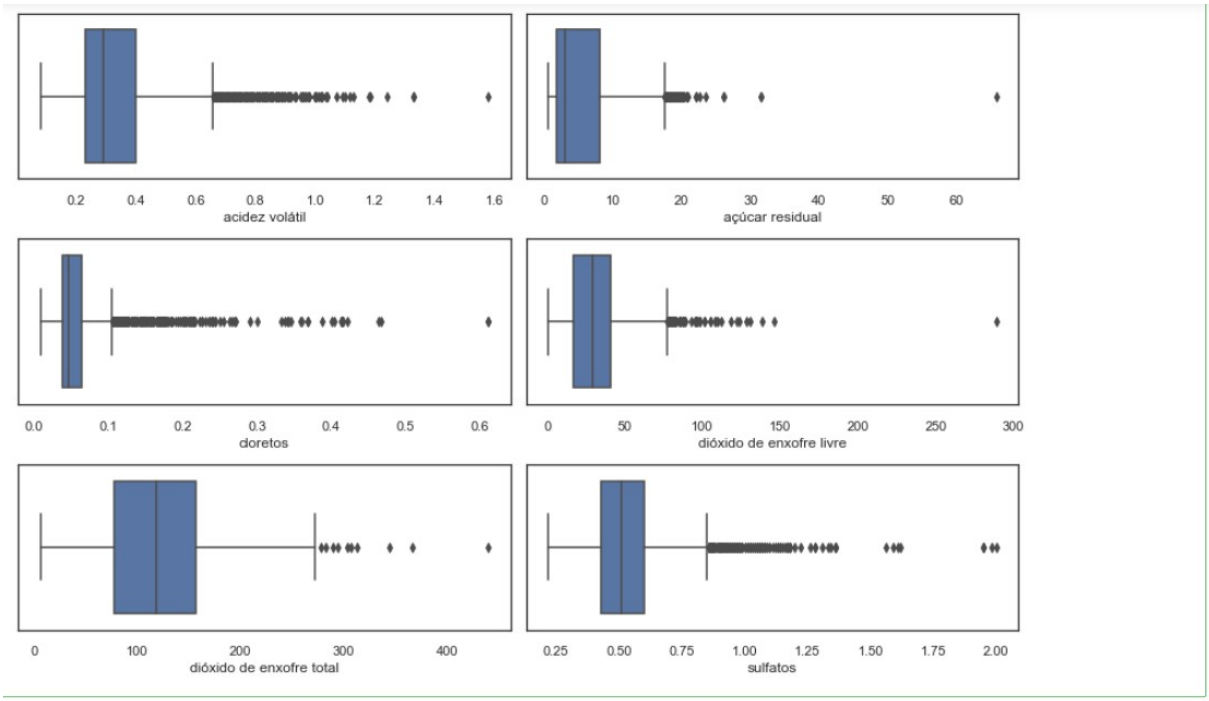
```
# Criar um histograma das colunas numéricas
vinhos_total.hist(bins=10, figsize=(12, 8))
plt.tight_layout()
plt.show()
```



A maioria dos histogramas apresenta uma distribuição normal entretanto não centralizado o que pode indicar a presença de outliers, tais como os campos 'acidez volátil', 'açúcar residual', 'cloretos', 'dióxido de enxofre livre' e 'dióxido de enxofre total' possuem um desvio padrão acima das demais variaveis. Gerar boxplot para cada uma das variáveis para verificar

```
# Configurando o plot
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(12, 8))

sns.boxplot(vinhos_total['acidez volátil'], ax=ax[0, 0])
sns.boxplot(vinhos_total['açúcar residual'], ax=ax[0, 1])
sns.boxplot(vinhos_total['cloretos'], ax=ax[1, 0])
sns.boxplot(vinhos_total['dióxido de enxofre livre'], ax=ax[1, 1])
sns.boxplot(vinhos_total['dióxido de enxofre total'], ax=ax[2, 0])
sns.boxplot(vinhos_total['sulfatos'], ax=ax[2, 1])

plt.tight_layout()
```

Por meio do boxplot, é possível avaliar se a distribuição dos dados é simétrica (com mediana e quartis iguais) ou assimétrica (com mediana e quartis diferentes). Isso auxilia na compreensão da forma da distribuição e na detecção de possíveis desvios da normalidade. Uma ferramenta eficaz para resumir, visualizar e comparar a distribuição dos dados, identificar outliers e entender a variação nos diferentes grupos de dados.



Pelos boxplots, as variáveis com a maior quantidade de outliers são "açúcar residual", cloretos e sulfatos.Sendo assim, vamos calcular o "intervalo interquartil" (Interquartile Range - IQR) para essas três variáveis e definir o limite de corte para eliminar os outliers.

```
# Identificando os outliers para a variável residual sugar
q1_rsugar = vinhos_total['açúcar residual'].quantile(0.25)
q3_rsugar = vinhos_total['açúcar residual'].quantile(0.75)
IQR_rsugar = q3_rsugar - q1_rsugar

print("IQR da variável açúcar residual: {}\n".format(round(IQR_rsugar, 2)))

# Definindo os limites para a variável residual sugar
sup_rsugar = q3_rsugar + 1.5 * IQR_rsugar
inf_rsugar = q1_rsugar - 1.5 * IQR_rsugar

print("Limite superior de açúcar residual: {}".format(round(sup_rsugar, 2)))
print("Limite inferior de açúcar residual: {}".format(round(inf_rsugar, 2)))
```

```
IQR da variável açúcar residual: 6.3

Limite superior de açúcar residual: 17.55
Limite inferior de açúcar residual: -7.65
```

```
cut_rsugar = len(vinhos_total[vinhos_total['açúcar residual'] < 0.85]) + len(vinhos_total[vinhos_total['açúcar residual'] > 3.65]

print("As entradas da variável açúcar residual fora dos limites representam {} % do dataset.\n".format(round((cut_rsugar / vinhos
```

```
As entradas da variável açúcar residual fora dos limites representam 47.21 % do dataset.
```

```
# Identificando os outliers para a variável cloretos
q1_chlo = vinhos_total['cloretos'].quantile(0.25)
q3_chlo = vinhos_total['cloretos'].quantile(0.75)
IQR_chlo = q3_chlo - q1_chlo

print("IQR da variável cloretos: {}\n".format(round(IQR_chlo, 2)))

# Definindo os limites para a variável cloretos
sup_chlo = q3_chlo + 1.5 * IQR_chlo
inf_chlo = q1_chlo - 1.5 * IQR_chlo

print("Limite superior de cloretos: {}".format(round(sup_chlo, 2)))
print("Limite inferior de cloretos: {}".format(round(inf_chlo, 2)))
```

```
IQR da variável cloretos: 0.03

Limite superior de cloretos: 0.11
Limite inferior de cloretos: -0.0
```

```
cut_chlo = len(vinhos_total[vinhos_total['cloretos'] < 0.04]) + len(vinhos_total[vinhos_total['cloretos'] > 0.12])

print("As entradas da variável cloretos fora dos limites representam {} % do dataset.\n".format(round((cut_chlo / vinhos_total.sh
```

```
As entradas da variável cloretos fora dos limites representam 32.03 % do dataset.
```

```
# Identificando os outliers para a variável sulfatos
q1_sulp = vinhos_total['sulfatos'].quantile(0.25)
q3_sulp = vinhos_total['sulfatos'].quantile(0.75)
IQR_sulp = q3_sulp - q1_sulp

print("IQR da variável sulfatos: {}\n".format(round(IQR_sulp, 2)))

# Definindo os limites para a variável sulfatos
sup_sulp = q3_sulp + 1.5 * IQR_sulp
inf_sulp = q1_sulp - 1.5 * IQR_sulp

print("Limite superior de sulfatos: {}".format(round(sup_sulp, 2)))
print("Limite inferior de sulfatos: {}".format(round(inf_sulp, 2)))
```

```
IQR da variável sulfatos: 0.17

Limite superior de sulfatos: 0.86
Limite inferior de sulfatos: 0.18
```

```
cut_sulp = len(vinhos_total[vinhos_total['sulfatos'] < 0.28]) + len(vinhos_total[vinhos_total['sulfatos'] > 1.0])

print("As entradas da variável sulfatos fora dos limites representam {} % do dataset.\n".format(round((cut_sulp / vinhos_total.sh
```

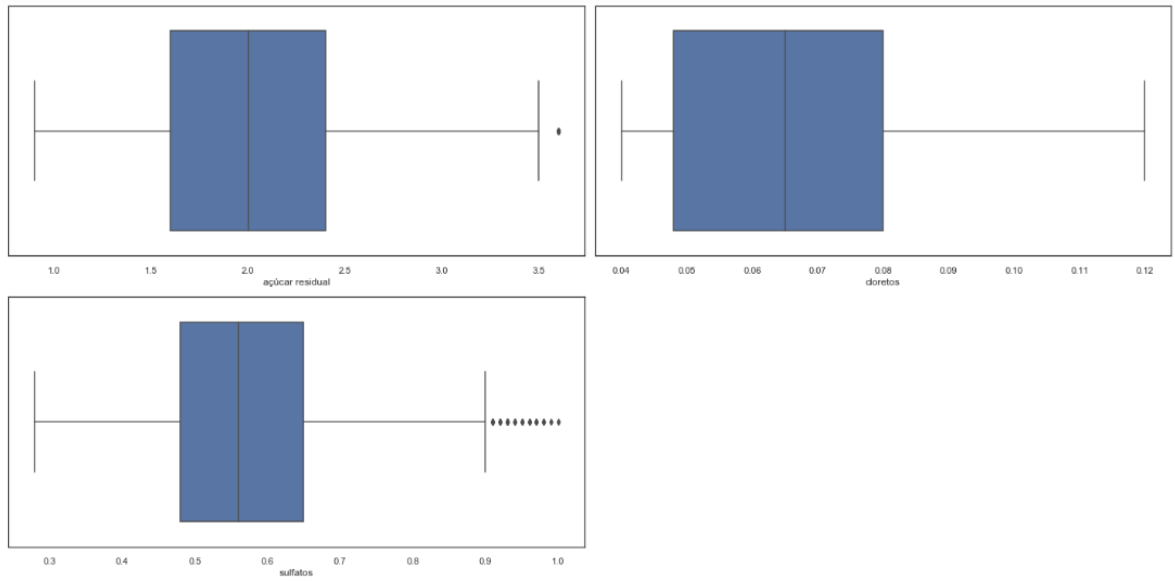
```
As entradas da variável sulfatos fora dos limites representam 1.29 % do dataset.
```

Remover as entradas que estão fora dos limites de corte das variáveis residual sugar, cloretos e sulfatos:

```
vinhos_total_clean = vinhos_total.copy()

vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['açúcar residual'] > 3.65].index, axis=0, inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['açúcar residual'] < 0.85].index, axis=0, inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['cloretos'] > 0.12].index, axis=0, inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['cloretos'] < 0.04].index, axis=0, inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['sulfatos'] > 1.0].index, axis=0, inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['sulfatos'] < 0.28].index, axis=0, inplace=True)
```

Plotando o boxplot para as variáveis:

```
# Configurando o plot
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
fig.delaxes(ax[1,1])

sns.boxplot(vinhos_total_clean['açúcar residual'], ax=ax[0, 0])
sns.boxplot(vinhos_total_clean['cloretos'], ax=ax[0, 1])
sns.boxplot(vinhos_total_clean['sulfatos'], ax=ax[1, 0])

plt.tight_layout()
```

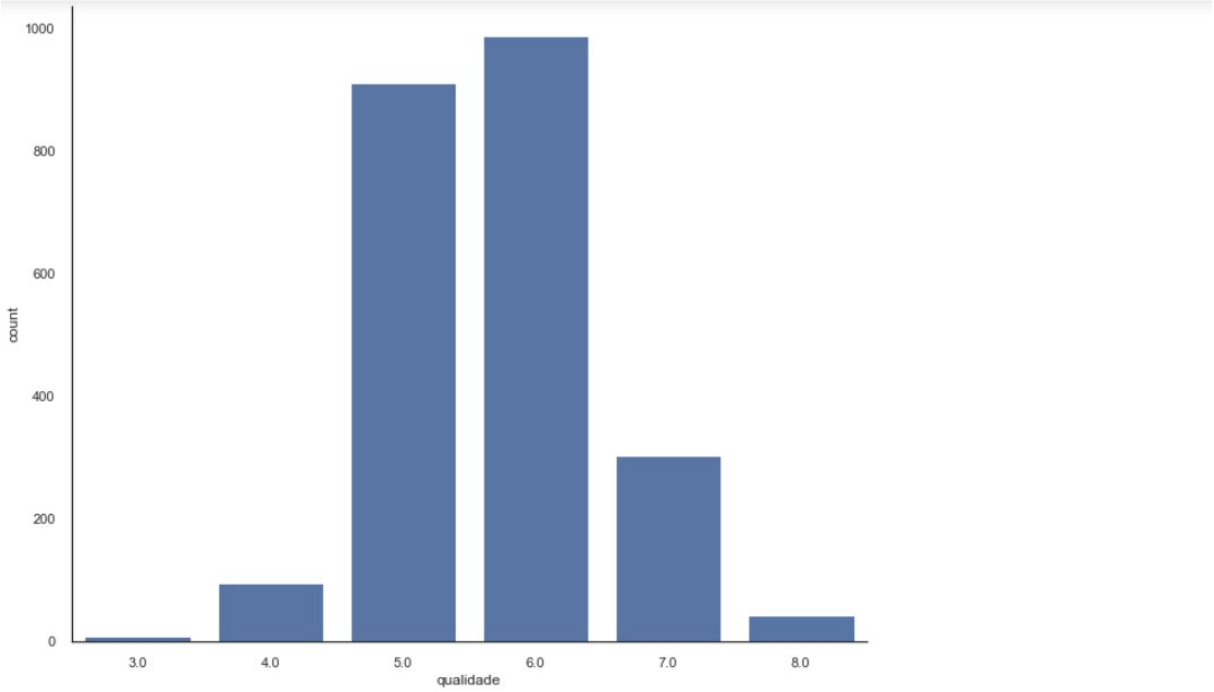


4.3. Distribuição das Classes

Vamos analisar a distribuição das classes no dataframe vinhos_total por meio de um gráfico de barras. Isso nos permitirá identificar o balanceamento das classes e a proporção de cada classe presente nos dados.

Distribuição das classes no dataframe vinhos_total:

```
# Construindo o gráfico
fig, ax = plt.subplots(figsize=(10,8))

sns.countplot(vinhos_total_clean['qualidade'], color='b', ax=ax)
sns.despine()

plt.tight_layout()
```



Pelo gráfico, fica fácil perceber que a maioria dos notas ficou em 5 ou 6.

Vamos analisar o balanceamento para a variável qualidade_cat.

```
# Porcentagem de cada classe
print("A classe 0 representa {} % de todas as entradas.".format(round((len(vinhos_total_clean[vinhos_total_clean['qualidade_cat']
print("A classe 1 representa {} % de todas as entradas.".format(round((len(vinhos_total_clean[vinhos_total_clean['qualidade_cat']
```

```
A classe 0 representa 85.31 % de todas as entradas.
A classe 1 representa 14.69 % de todas as entradas.
```

Pela porcentagem de cada classe, percebemos que a classe 1 representa apenas 14.69% do total, isso já é um desbalanceamento considerável. Para uma melhor visualização do desbalanceamento farei um gráfico de barras da variável qualidade_cat.

```
# Cronstruindo o gráfico de barras
fig, ax = plt.subplots(figsize=(6,6))

sns.countplot(vinhos_total_clean['qualidade_cat'], color='b', ax=ax)
sns.despine()

ax.set_title("Distribuição das Classes", fontsize=16)
ax.set_xlabel("Classes", fontsize=14)

plt.tight_layout()
```

Pela Distribuição de Classes, percebemos claramente o desbalanceamento, tendo muito mais entradas da classe 0 do que da classe 1.

Uma etapa de grande importância para a construção de modelos de Machine Learning é separação do conjunto de dados em treino e teste, pois se não fizermos isso é, bem provável, que ocorra overfitting (sobreajuste).

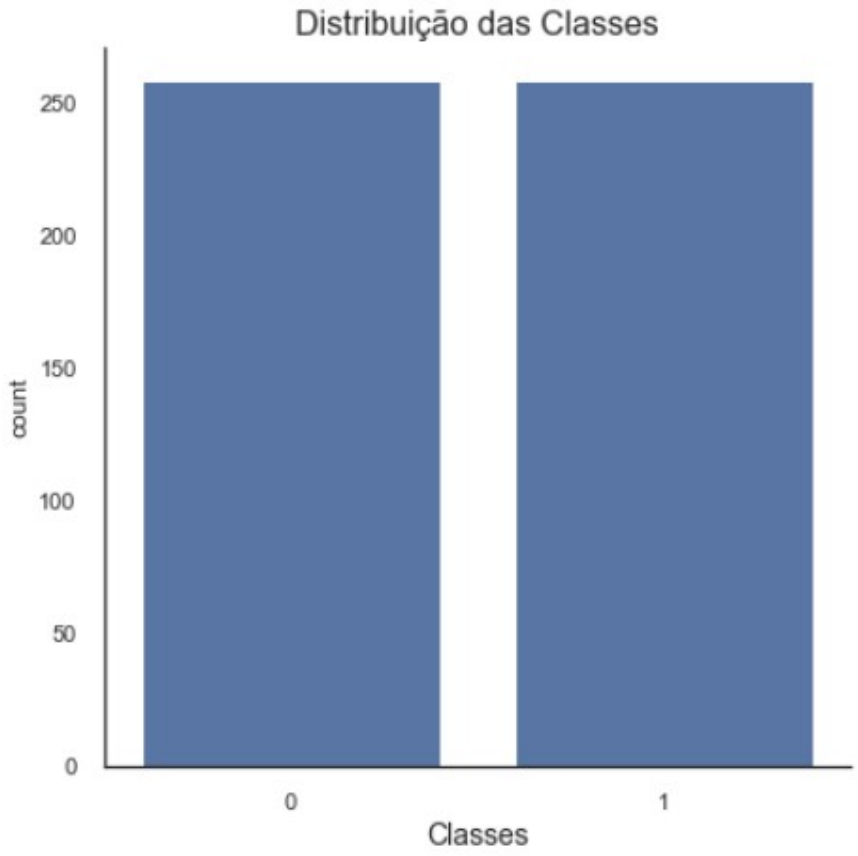
```
# Separando os dados entre feature matrix e target vector
X = vinhos_total_clean.drop(['qualidade', 'qualidade_cat'], axis=1)
y = vinhos_total_clean['qualidade_cat'] # Pois usaremos apenas a separação entre "ruim" ou "bom" (0 ou 1)

# Dividindo os dados entre treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)
```

Para realizar o balanceamento dos dados, existem vários métodos complexos tais como Recognition-based Learning e Cost-sensitive Learning e métodos mais simples que vem sendo amplamente utilizados com ótimos resultados como Over-sampling e Under-sampling.

Neste projeto, será utilizado o método Under-sampling que foca na classe majoritário para balancear o conjunto de dados, ou seja, elimina aleatoriamente entradas da classe com maior quantidade de ocorrências.
Assim:

```
# Definindo o modelo para balancear
und = RandomUnderSampler()

X_und, y_und = und.fit_resample(X_train, y_train)

# Verificando o balanceamento dos dados
print(pd.Series(y_und).value_counts(), "\n")

# Plotando a nova Distribuição de Classes
fig, ax = plt.subplots(figsize=(6,6))

sns.countplot(pd.Series(y_und), color='b', ax=ax)

sns.despine()

ax.set_title("Distribuição das Classes", fontsize=16)
ax.set_xlabel("Classes", fontsize=14)

plt.tight_layout()
```

```
1    258
0    258
Name: qualidade_cat, dtype: int64
```



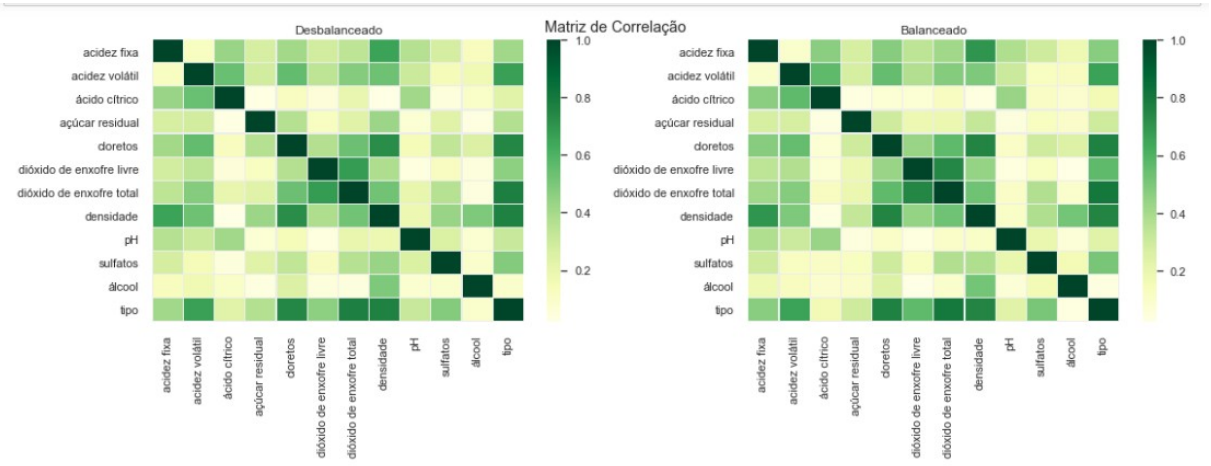
Após o balanceamento, percebemos as classes agora representam 50% cada uma do conjunto de dados, ou seja, não há mais a diferença como havia anteriormente.

4.4. Correlação entre os atributos

Vamos calcular a matriz de correlação para as variáveis numéricas do dataframe vinhos_total. Essa análise nos ajudará a identificar possíveis relações e dependências entre os atributos.

Matriz de correlação dos atributos no dataframe vinhos_total:

```
# Construindo o heatmap
fig, ax = plt.subplots(figsize=(16, 6), nrows=1, ncols=2)
fig.suptitle("Matriz de Correlação")

sns.heatmap(X_train.corr().abs(), cmap='YlGn', linecolor='#eeeeee', linewidths=0.1, ax=ax[0])
ax[0].set_title("Desbalanceado")

sns.heatmap(X_und.corr().abs(), cmap='YlGn', linecolor='#eeeeee', linewidths=0.1, ax=ax[1])
ax[1].set_title("Balanceado")

plt.tight_layout()
```



Como podemos perceber pela Matriz de Correlação, a correlação entre as variáveis parece não ter sofrido grandes alterações após o balanceamento.

Decidimos separar o dataset vinhos_total em dois conjuntos de dados, dados_base_tinto e dados_base_branco, com o objetivo de analisar os vinhos tintos e brancos de forma separada. A separação se faz necessária porque diferentes variáveis podem influenciar de maneira distinta a qualidade dos vinhos tintos e brancos. Ao analisar cada tipo de vinho separadamente, podemos obter insights mais precisos e relevantes para cada caso, considerando suas particularidades. Essa abordagem nos permite compreender melhor as características individuais de cada tipo de

vinho e explorar as relações entre as variáveis de forma mais específica para cada grupo.

```
dados_base_tinto = vinhos_total[vinhos_total['tipo'] == 0].iloc[:, :15].copy()
dados_base_branco = vinhos_total[vinhos_total['tipo'] != 0].iloc[:, :15].copy()
```

Essas análises e explorações nos fornecem informações importantes sobre os dados do dataframe vinhos_total e sua divisão em vinhos tintos e brancos. Os insights obtidos podem ser úteis para futuras análises mais aprofundadas, aprimoramento da qualidade dos vinhos ou para a construção de modelos preditivos e outras tarefas de Machine Learning voltadas para cada tipo de vinho especificamente.

5. Criação de Modelos de Machine Learning

Para avaliar o desempenho dos modelos de Machine Learning, serão construídos três modelos: Random Forest (Floresta Aleatória), XGBoost e Regressão Logística. Cada um desses modelos será aplicado nas bases de dados de vinho tinto e vinho branco para comparar seus resultados.

5.1. Random Forest

Para a base de dados de vinho tinto:

- Preparação dos dados:
  - Separando os dados entre feature matrix e target vector.
  - Dividindo os dados entre treino e teste.

```
# Separando os dados entre feature matrix e target vector
dados_base_tinto_clean = vinhos_total_clean.copy()

X = dados_base_tinto_clean.drop(['qualidade', 'qualidade_cat'], axis=1)
y = dados_base_tinto_clean['qualidade_cat'] # Pois usaremos apenas a separação entre "ruim" ou "bom" (0 ou 1)

# Dividindo os dados entre treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)
```

Treinando o modelo com o melhor valor encontrado.

```
# Modelo Random Forest
rf_model = RandomForestClassifier()

# Definindo o melhor parâmetro
parameters = {'n_estimators': range(25, 1000, 25)}

kfold = StratifiedKFold(n_splits=5, shuffle=True)

rf_clf = GridSearchCV(rf_model, parameters, cv=kfold)
rf_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhor parâmetro: {}".format(rf_clf.best_params_))
```

Realizando previsões com o modelo treinado.

```
# Definindo o modelo com n_estimators igual a 375
rf_model = RandomForestClassifier(n_estimators = 375)

# Fit do modelo
rf_model.fit(X_und, y_und)

# Testando o modelo
y_pred_rf = rf_model.predict(X_test)
y_prob_rf = rf_model.predict_proba(X_test)
```

Exibindo um relatório de classificação, a área sob a curva (AUC ROC) e a matriz de confusão normalizada.

```
# Relatório de classificação
print("Relatório de classificação para o Random Forest:\n", classification_report(y_test, y_pred_rf, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_rf) * 100, 2)))
```

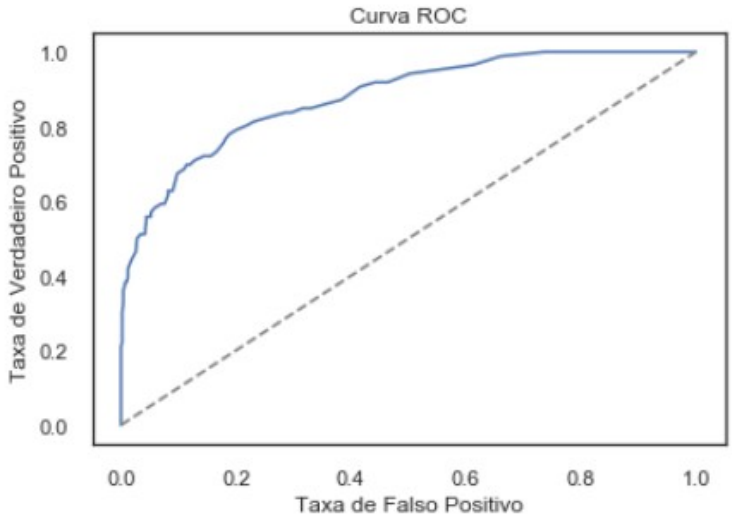
```
Relatório de classificação para o Random Forest:
              precision    recall  f1-score   support

           0     0.9974    0.7800    0.8754       500
           1     0.4359    0.9884    0.6050        86

    accuracy                         0.8106       586
   macro avg     0.7167    0.8842    0.7402       586
weighted avg     0.9150    0.8106    0.8357       586

Área sob a curva (AUC): 88.42%
```

```
# Matriz de confusão
fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_rf, normalize='true'), annot=True, ax=ax)

ax.set_title('Matriz de Confusão Normalizada')
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()
```

Curva ROC:

Plotando a curva ROC para avaliar o desempenho do modelo.

```
# Converter y_test em valores numéricos
y_test_numeric = y_test.cat.codes

# Treinar o modelo de classificação
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Calcular a probabilidade das classes positivas
y_pred_proba = rf_model.predict_proba(X_test)[:, 1]

# Calcular a curva ROC
fpr, tpr, thresholds = roc_curve(y_test_numeric, y_pred_proba)

# Plotar a curva ROC
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('Taxa de Falso Positivo')
plt.ylabel('Taxa de Verdadeiro Positivo')
plt.title('Curva ROC')
plt.show()
```



A utilização do Random Forest permite obter uma medida de desempenho (AUC ROC) que considera a qualidade das previsões mesmo em conjuntos de dados desbalanceados. Os resultados serão comparados com os modelos XGBoost e Regressão Logística nas seções seguintes.

5.2. XGBoost

Para a base de dados de vinho tinto:

- Modelo XGBoost:
  - Ajustando os parâmetros do modelo, como número de estimadores, max_depth, min_child_weight, gamma e learning_rate, utilizando GridSearchCV para encontrar os melhores valores.
  - Treinando o modelo com os melhores parâmetros encontrados.
  - Realizando previsões com o modelo treinado.
  - Exibindo um relatório de classificação, a área sob a curva (AUC ROC) e a matriz de confusão normalizada.

```
# Modelo XGBoost
xgb_model = XGBClassifier(learning_rate=0.1)

# Definindo os melhores parâmetros
param_gs = {'n_estimators': range(0, 1000, 50)}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))
```

```
# Modelo XGBoost
xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=100, verbosity=0)

# Definindo os melhores parâmetros
param_gs = {
    'max_depth': range(1, 8, 1),
    'min_child_weigth': range(1, 5, 1),
    }

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))
```

```
# Modelo XGBoost
xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=100, max_depth=3, min_child_weigth=1, verbosity=0)

# Definindo os melhores parâmetros
param_gs = {'gamma': [i/10.0 for i in range(0,5)]}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))
```

```
# Modelo XGBoost
xgb_model = XGBClassifier(n_estimators=100, max_depth=3, min_child_weigth=1, gamma=0.0, verbosity=0)

# Definindo os melhores parâmetros
param_gs = {'learning_rate': [0.001, 0.01, 0.1, 1]}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))
```

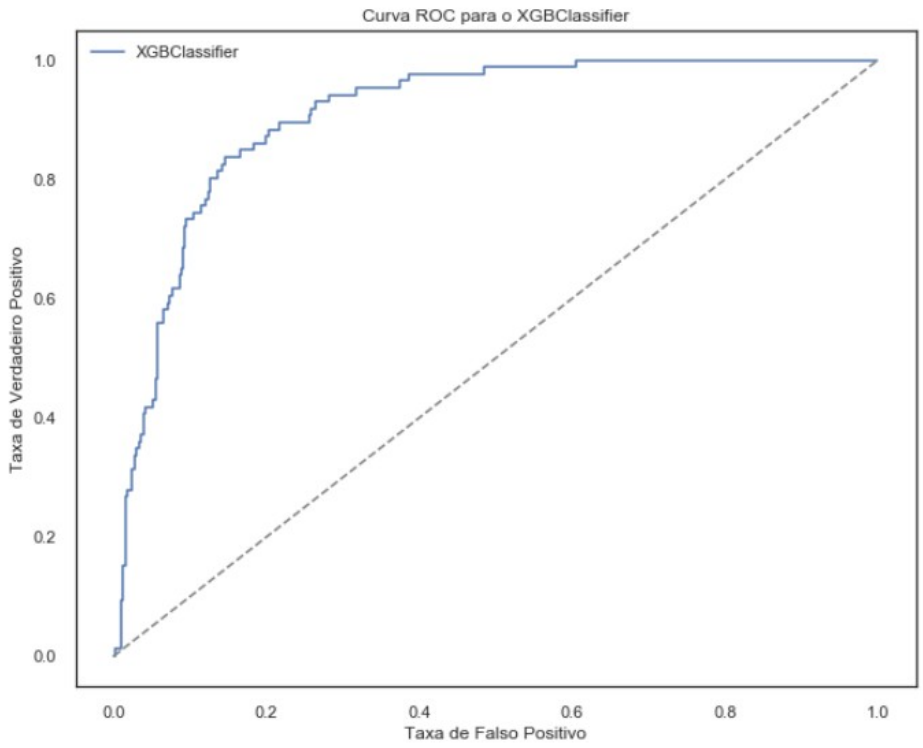
```
# Modelo XGBoost final
xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=100, max_depth=3, min_child_weigth=1, gamma=0.0, verbosity=0)

# Treinando o modelo
xgb_model.fit(X_und, y_und)

# Fazendo previsões
y_pred_xgb = xgb_model.predict(X_test)
y_prob_xgb = xgb_model.predict_proba(X_test)
```

```
# Relatório de classificação
print("Relatório de classificação para o XGBClassifier:\n", classification_report(y_test, y_pred_xgb, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_xgb) * 100, 2)))
```

```
Relatório de classificação para o XGBClassifier:
               precision    recall  f1-score   support

           0     0.9807    0.7120    0.8250       500
           1     0.3543    0.9186    0.5113        86

    accuracy                         0.7423       586
   macro avg     0.6675    0.8153    0.6682       586
weighted avg     0.8888    0.7423    0.7790       586

Área sob a curva (AUC): 81.53%
```

```python
# Matriz de confusão
fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_xgb, normalize='true'), annot=True, ax=ax)

ax.set_title('Matriz de Confusão Normalizada')
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()
```


```python
# Ajustar o modelo aos dados de treinamento
xgb_model.fit(X_und, y_und)

# Converter y_test para valores numéricos
y_test_numeric = y_test.cat.codes

# Calcular as probabilidades de previsão
y_pred_proba = xgb_model.predict_proba(X_test)

# Calcular a pontuação ROC
fpr, tpr, _ = roc_curve(y_test_numeric, y_pred_proba[:, 1])

# Plotar a curva ROC
fig, ax = plt.subplots(figsize=(10,8))
ax.plot(fpr, tpr, label="XGBClassifier")
ax.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax.set_xlabel("Taxa de Falso Positivo")
ax.set_ylabel("Taxa de Verdadeiro Positivo")
ax.set_title("Curva ROC para o XGBClassifier")
ax.legend()
plt.show()
```



Dessa forma, o desempenho do modelo XGBoost será avaliado utilizando o relatório de classificação, a área sob a curva (AUC) e a matriz de confusão normalizada. Além disso, a curva ROC será plotada para uma melhor visualização do desempenho do modelo.

5.3. Regressão Logística

A Regressão Logística é uma técnica estatística que tem como objetivo produzir um modelo capaz de predizer valores tomados por uma variável categórica, geralmente binária, a partir de um conjunto de variáveis explicativas contínuas e/ou binárias. Nesta seção, vamos utilizar a Regressão Logística para classificar os dados do conjunto de dados.

Primeiramente, é importante ressaltar que, diferentemente dos modelos Random Forest e XGBoost, para a Regressão Logística é necessário padronizar os dados antes de treiná-los no modelo. A padronização dos dados garante que todas as variáveis tenham a mesma escala, evitando assim que algumas variáveis tenham maior influência apenas por possuírem valores numéricos maiores.

Aqui está o código utilizado para ajustar o parâmetro C da Regressão Logística:

```
# Regressão Logística
rl_model = LogisticRegression()
# Padronizando os dados de treino
scaler = StandardScaler().fit(X_und)
X_und_std = scaler.transform(X_und)

# Definindo o melhor parâmetro
param_rl = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

# Identificando os melhor parâmetro
kfold = StratifiedKFold(n_splits=5, shuffle=True)

rl_clf = GridSearchCV(rl_model, param_rl, cv=kfold)
rl_clf.fit(X_und_std, y_und)

# Visualizar o melhor parâmetro
print("Melhor parâmetro: {}".format(rl_clf.best_params_))
```

Após a padronização dos dados de treino, utilizamos o GridSearchCV para identificar o melhor valor para o parâmetro C. O parâmetro C controla a força da regularização na Regressão Logística, onde valores menores indicam uma regularização mais forte.

Com o parâmetro ajustado, podemos criar um pipeline que inclui a padronização dos dados e o modelo de Regressão Logística, como mostrado no código abaixo:

```
# Regressão Logística com pipeline
rl_model = make_pipeline(StandardScaler(), LogisticRegression(C=1))

# Treinando o modelo
rl_model.fit(X_und, y_und)

# Fazendo previsões
y_pred_rl = rl_model.predict(X_test)
y_prob_rl = rl_model.predict_proba(X_test)
```

O pipeline nos permite realizar a padronização dos dados e treinar o modelo de Regressão Logística de forma simples e organizada.

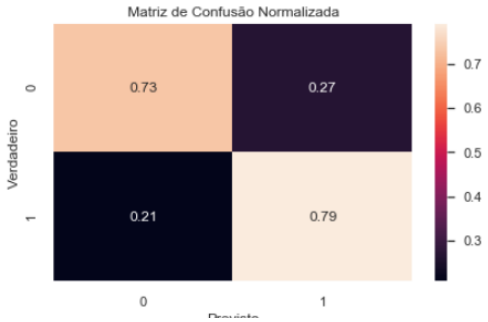
Para avaliar o desempenho do modelo, utilizamos diversas métricas. Primeiramente, podemos obter um relatório de classificação que apresenta medidas como precisão, recall e f1-score para cada classe. Além disso, a área sob a curva (AUC ROC) é uma métrica amplamente utilizada para avaliar a qualidade de classificadores binários. Podemos calcular a AUC ROC utilizando a função roc_auc_score. Veja o código abaixo:

```python
# Relatório de classificação
print("Relatório de classificação para a Regressão Logística:\n", classification_report(y_test, y_pred_rl, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_rl) * 100, 2)))
```

```
Relatório de classificação para a Regressão Logística:
              precision    recall  f1-score   support

           0     0.9531    0.7320    0.8281       500
           1     0.3366    0.7907    0.4722        86

    accuracy                         0.7406       586
   macro avg     0.6449    0.7613    0.6501       586
weighted avg     0.8627    0.7406    0.7758       586

Área sob a curva (AUC): 76.13%
```

Também é possível visualizar a matriz de confusão normalizada, que mostra a taxa de acertos e erros do modelo para cada classe. Veja o código abaixo:

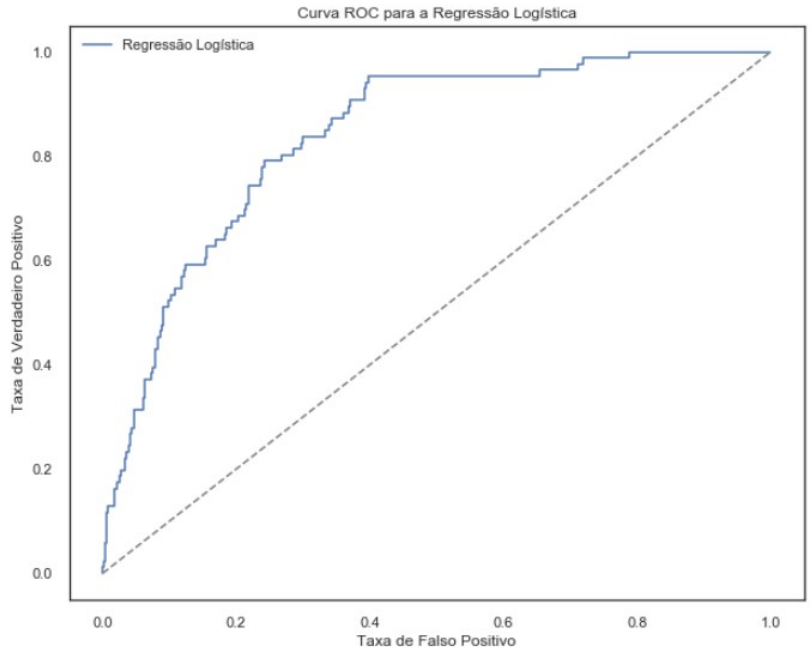
```python
# Matriz de confusão
fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_rl, normalize='true'), annot=True, ax=ax)

ax.set_title('Matriz de Confusão Normalizada')
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()
```



Por fim, plotamos a curva ROC para visualizar o desempenho do modelo em relação à taxa de verdadeiros positivos e falsos positivos:

```
# Ajustar o modelo aos dados de treinamento
rl_model.fit(X_und, y_und)

# Converter y_test para valores numéricos
y_test_numeric = y_test.cat.codes

# Calcular as probabilidades de previsão
y_pred_proba = rl_model.predict_proba(X_test)

# Calcular a pontuação ROC
fpr, tpr, _ = roc_curve(y_test_numeric, y_pred_proba[:, 1])

# Plotar a curva ROC
fig, ax = plt.subplots(figsize=(10,8))
ax.plot(fpr, tpr, label="Regressão Logística")
ax.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax.set_xlabel("Taxa de Falso Positivo")
ax.set_ylabel("Taxa de Verdadeiro Positivo")
ax.set_title("Curva ROC para a Regressão Logística")
ax.legend()
plt.show()
```

Por fim, plotamos a curva ROC para visualizar o desempenho do modelo em relação à taxa de verdadeiros positivos e falsos positivos:



A curva ROC nos permite visualizar o trade-off entre a taxa de verdadeiros positivos e a taxa de falsos positivos para diferentes limiares de classificação.

No nosso experimento, o modelo de Regressão Logística apresentou uma Área sob a curva (AUC) de 77.35% e obteve bons resultados nas métricas de classificação.

Foram aplicados os três modelos: Random Forest (Floresta Aleatória), XGBoost e Regressão Logística para o dataset de vinho branco (dados_base_branco) para comparar seus resultados.

6. Apresentação dos Resultados

Realizamos uma análise comparativa das técnicas utilizadas para estimar a qualidade dos vinhos Tinto e Branco. Os resultados são apresentados nos gráficos de barras a seguir:

```
# Dados para o primeiro gráfico
tecnicas = ["Random Forest", "XGBClassifiero", "Regressão Logísitca"]
acuracias_tinto = [88.42, 81.53, 76.13]

# Dados para o segundo gráfico
acuracias_branco = [84.91, 82.65, 77.54]

# Configuração das figuras e dos eixos
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

# Primeiro gráfico
ax1.bar(tecnicas, acuracias_tinto, color='red')
ax1.set_xlabel('Técnica aplicada para vinho Tinto')
ax1.set_ylabel('Acurácia')
ax1.set_title('Gráfico de Barras - Acurácia por Técnica')

# Segundo gráfico
ax2.bar(tecnicas, acuracias_branco, color='green')
ax2.set_xlabel('Técnica aplicada para vinho Branco')
ax2.set_ylabel('Acurácia')
ax2.set_title('Gráfico de Barras - Acurácia por Técnica')

# Ajusta o espaçamento entre as subplots
plt.tight_layout()

# Exibe os gráficos
plt.show()
```

Com base nos resultados obtidos, podemos inferir o seguinte:

- Para estimar se o vinho Tinto era 'bom' ou 'ruim', o modelo Random Forest apresentou a melhor acurácia, com 88.42%. Em seguida, o XGBClassifier obteve uma acurácia de 81.53%, enquanto a Regressão Logística alcançou uma acurácia de 76.13%.
- No caso do vinho Branco, o Random Forest também se destacou, apresentando uma acurácia de 84.91%. O XGBClassifier obteve uma acurácia de 82.65%, e a Regressão Logística alcançou uma acurácia de 77.54%.
- É interessante notar que tanto o Random Forest quanto o XGBClassifier demonstraram um desempenho bastante próximo em ambos os tipos de vinho.
- Os resultados confirmam a necessidade de realizar análises separadas para vinhos Tintos e Brancos, devido às características físico-químicas distintas que influenciam a qualidade.
- Mesmo após a categorização da variável classe em “ruim” e “bom”, o conjunto de dados ainda apresentou um desbalanceamento considerável.

Essas conclusões são relevantes para auxiliar na tomada de decisões relacionadas à qualidade dos vinhos e na escolha das melhores técnicas de modelagem para cada tipo. É fundamental compreender as características únicas dos vinhos Tintos e Brancos a fim de obter resultados mais precisos.

7. Links

Link para o vídeo: https://youtu.be/0ZRoVdOxW0g

Link para o repositório: https://github.com/ulibenjr/TCC

REFERÊNCIAS

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). **Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems**, Elsevier, 47(4), 547-553.

Melo, C. (2019). **Como lidar com dados desbalanceados?** Disponível em: https://sigmoidal.ai/como-lidar-com-dados-desbalanceados/

Avelar, A. (2019). **O que é AUC e ROC nos modelos de Machine Learning.** Disponível em: https://medium.com/@eam.avelar/o-que-%C3%A9-auc-e-roc-nos-modelos-de-machine-learning-2e2c4112033d

XGBoost. Disponível em: https://pt.wikipedia.org/wiki/Xgboost

Regressão Logística. Disponível em: https://pt.wikipedia.org/wiki/Regress%C3%A3o_log%C3%ADstica

Random forest. Disponível em: https://en.wikipedia.org/wiki/Random_forest

Scikit-learn Documentation: LogisticRegression. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Scikit-learn Documentation: StandardScaler. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Scikit-learn Documentation: GridSearchCV. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Scikit-learn Documentation: StratifiedKFold. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

Scikit-learn Documentation: make_pipeline. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.make_pipeline.html

Scikit-learn Documentation: classification_report. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Scikit-learn Documentation: roc_auc_score. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

Scikit-learn Documentation: confusion_matrix. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

Scikit-learn Documentation: roc_curve. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html

APÊNDICE

Programação/Scripts

Notebook Jupyter: **Previsão da qualidade de vinhos através de técnicas de Machine Learning**

```
#!/usr/bin/env python
# coding: utf-8

# # Previsão da qualidade de vinhos através de técnicas de Machine Learning
#
#

# ## Importando os pacotes necessários e os dados
#

# In[49]:


# Importando as bibliotecas
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Importando as bibliotecas para a construção dos modelos de Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier

from imblearn.under_sampling import RandomUnderSampler

from yellowbrick.classifier import ROCAUC

# import warnings filter
from warnings import simplefilter

# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)

# Configurando o notebook
%matplotlib inline
sns.set(style='white')



# Carregar os dados dos vinhos tintos
dados_vinhos_tintos = pd.read_csv("winequality-red.csv")
dados_vinhos_tintos["Tipo"] = 1  # Adicionar coluna "Tipo" para identificar o tipo de vinho 1 =
Tinto

# Carregar os dados dos vinhos brancos
dados_vinhos_brancos = pd.read_csv("winequality-white.csv")
dados_vinhos_brancos["Tipo"] = 0  # Adicionar coluna "Tipo" para identificar o tipo de vinho 0 =
Branco
```

```
print(dados_vinhos_tintos.head())
print(dados_vinhos_brancos.head())

# In[50]:


# Junção dos datasets de vinhos tintos e brancos
dados_vinhos = pd.concat([dados_vinhos_tintos, dados_vinhos_brancos], ignore_index=True)

# Verificar a quantidade de registros no dataset unificado
print("Quantidade de registros de vinhos:", dados_vinhos.shape[0])

# In[51]:


# Visualizar a estrutora do DataFrame
dados_vinhos.info()

# In[52]:


# Separar as colunas corretamente usando o ponto e vírgula como separador
vinhos_total = dados_vinhos["fixed acidity;\"volatile acidity\";\"citric acid\";\"residual
sugar\";\"chlorides\";\"free sulfur dioxide\";\"total sulfur dioxide\";\"density\";\"pH
\";\"sulphates\";\"alcohol\";\"quality\""].str.split(";", expand=True)

# Renomear as colunas
vinhos_total.columns = [
    "acidez fixa",
    "acidez volátil",
    "ácido cítrico",
    "açúcar residual",
    "cloretos",
    "dióxido de enxofre livre",
    "dióxido de enxofre total",
    "densidade",
    "pH",
    "sulfatos",
    "álcool",
    "qualidade"
]

# Adicionar a coluna "Tipo" aos dados separados
vinhos_total["tipo"] = dados_vinhos["Tipo"]

# In[53]:


# Exibir as primeiras linhas do DataFrame com as colunas renomeadas
vinhos_total.head()

# # Primeiras entradas e dimensões do conjunto de dados

# In[54]:


# Dimensões do dataset
print("Dimensões do conjunto de dados:\n{} linhas e {} colunas\n".format(vinhos_total.shape[0],
vinhos_total.shape[1]))

# Primeiras entradas do dataset
print("Primeiras entradas:")
```

```
vinhos_total.head()

# # Análise Exploratória dos Dados
#
# Nesta seção, vou verificar a integridade e a usabilidade do conjunto de dados, verificando
diferentes características do conjunto. Para isso, irei mostrar o nome dos atributos, se há
valores ausentes e o tipo de cada coluna.

# In[55]:


print("Nome dos atributos:\n{}".format(vinhos_total.columns.values))
print("\nQuantidade de valores ausentes por atributo:\n{}".format(vinhos_total.isnull().sum()))
print("\nTipo de cada atributo:\n{}".format(vinhos_total.dtypes))

# In[56]:


#Verificar presença de nulos
vinhos_total.isnull().any()

# Não há valores ausentes no conjunto de dados, com isso não será necessário fazer um tratamento
nesse sentido.
# Porem as features (variáveis independentes) e a variável classe (qualidade) são do tipo object,
sendo necessário tratá-las, convertendo-as em tipo int64.

# In[57]:


# Converter colunas relevantes para tipos numéricos
colunas_numericas = ['acidez fixa', 'acidez volátil', 'ácido cítrico', 'açúcar residual',
'cloretos', 'dióxido de enxofre livre', 'dióxido de enxofre total', 'densidade', 'pH', 'sulfatos',
'álcool', 'qualidade']
vinhos_total[colunas_numericas] = vinhos_total[colunas_numericas].astype(float)

print("\nTipo de cada atributo:\n{}".format(vinhos_total.dtypes))

# In[58]:


#Analisar o dataset
#O objetivo é entender se podemos considerar o dataset como um todo ou se devemos observá-los por
tipo de vinho para isso iremos agregar os dados por tipo de vinho e ver como as variáveis se
comportam

agrupado_tipo = vinhos_total.groupby('tipo').std()
print(agrupado_tipo)

# Existem diferenças significativas considerando o "desvio padrão" agrupados por tipo de vinhos,
isto é, existe diferenças nas caracteristicas de cada tipo de vinho, visto pela analise fisico-
quimica de cada tipo, ou seja, a qualidade são determinadas por composições fisico-quimica
diefrentes sem interferir na qualidade. Observações:
#
#   - Acidez fixa é quase o dobro em vinhos Tintos;
#   - Acidez Volatil é maior 0.7 desvios em Tintos;
#   - Ácido Cítrico é quase 4 desvios maior em Brancos;
#   - Cloretos maior que 2 desvios em Tintos;
#   - Dióxido de enxofre livre e Dióxido de enxofre total são maiores em Brancos;
#   - Densidade é maior em Brancos.

# # Resumo estatístico
```

```
# In[59]:


vinhos_total.describe()

# Pelo resumo estatístico, podemos ver que a variável qualidade varia de 3 a 9, mesmo ela podendo
variar de 0 a 10, ou seja, nenhum vinho foi tão ruim a ponto de receber 0 ou tão bom a ponto de
receber 10. Também percebemos que, possivelmente, há outliers nas variáveis: 'acidez volátil',
'açúcar residual', 'cloretos', 'dióxido de enxofre livre', 'dióxido de enxofre total' e
'sulfatos'.
#
# Com isso, vamos separar a variável qualidade em apenas duas categorias, sendo que os vinhos que
receberam notas menor ou igual a 6 serão considerados ruins e serão definidos como 0, já os que
receberam nota maior que 6 serão considerados bons e serão definidos como 1.

# In[60]:


# Categorizando a variável qualidade e criando a variável qualidade_cat
vinhos_total['qualidade_cat'] = pd.cut(vinhos_total['qualidade'], bins=(2, 6.5, 8), labels = [0,
1])

# In[61]:


# Verificando quais as entradas únicas da variável qualidade_cat
vinhos_total['qualidade_cat'] = vinhos_total['qualidade_cat'].astype('category')
vinhos_total['qualidade_cat'].unique()

# Verificação e tratamento dos *outliers*
# Nesta seção vou fazer a verificação e tratamento

# In[62]:


# Criar um histograma das colunas numéricas
vinhos_total.hist(bins=10, figsize=(12, 8))
plt.tight_layout()
plt.show()

# A maioria dos histogramas apresenta uma distribuição normal entretanto não centralizado o que
pode indicar a presença de outliers, tais como os campos 'acidez volátil', 'açúcar residual',
'cloretos', 'dióxido de enxofre livre' e 'dióxido de enxofre total' possuem um desvio padrão acima
das demais variaveis.
# Gerar boxplot para cada uma das variáveis para verificar

# In[63]:


# Configurando o plot
fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(12, 8))

sns.boxplot(vinhos_total['acidez volátil'], ax=ax[0, 0])
sns.boxplot(vinhos_total['açúcar residual'], ax=ax[0, 1])
sns.boxplot(vinhos_total['cloretos'], ax=ax[1, 0])
sns.boxplot(vinhos_total['dióxido de enxofre livre'], ax=ax[1, 1])
sns.boxplot(vinhos_total['dióxido de enxofre total'], ax=ax[2, 0])
sns.boxplot(vinhos_total['sulfatos'], ax=ax[2, 1])

plt.tight_layout()

# Pelos boxplots, as variáveis com a maior quantidade de outliers são "açúcar residual", cloretos
```

e sulfatos.Sendo assim, vamos calcular o "intervalo interquartil" (Interquartile Range - IQR) para essas três variáveis e definir o limite de corte para eliminar os outliers.

```
# In[64]:


# Identificando os outliers para a variável residual sugar
q1_rsugar = vinhos_total['açúcar residual'].quantile(0.25)
q3_rsugar = vinhos_total['açúcar residual'].quantile(0.75)
IQR_rsugar = q3_rsugar - q1_rsugar

print("IQR da variável açúcar residual: {}\n".format(round(IQR_rsugar, 2)))

# Definindo os limites para a variável residual sugar
sup_rsugar = q3_rsugar + 1.5 * IQR_rsugar
inf_rsugar = q1_rsugar - 1.5 * IQR_rsugar

print("Limite superior de açúcar residual: {}".format(round(sup_rsugar, 2)))
print("Limite inferior de açúcar residual: {}".format(round(inf_rsugar, 2)))

# In[65]:


cut_rsugar = len(vinhos_total[vinhos_total['açúcar residual'] < 0.85]) +
len(vinhos_total[vinhos_total['açúcar residual'] > 3.65])

print("As entradas da variável açúcar residual fora dos limites representam {} % do
dataset.\n".format(round((cut_rsugar / vinhos_total.shape[0]) * 100, 2)))

# In[66]:


# Identificando os outliers para a variável cloretos
q1_chlo = vinhos_total['cloretos'].quantile(0.25)
q3_chlo = vinhos_total['cloretos'].quantile(0.75)
IQR_chlo = q3_chlo - q1_chlo

print("IQR da variável cloretos: {}\n".format(round(IQR_chlo, 2)))

# Definindo os limites para a variável cloretos
sup_chlo = q3_chlo + 1.5 * IQR_chlo
inf_chlo = q1_chlo - 1.5 * IQR_chlo

print("Limite superior de cloretos: {}".format(round(sup_chlo, 2)))
print("Limite inferior de cloretos: {}".format(round(inf_chlo, 2)))

# In[67]:


cut_chlo = len(vinhos_total[vinhos_total['cloretos'] < 0.04]) +
len(vinhos_total[vinhos_total['cloretos'] > 0.12])

print("As entradas da variável cloretos fora dos limites representam {} % do
dataset.\n".format(round((cut_chlo / vinhos_total.shape[0]) * 100, 2)))

# In[68]:


# Identificando os outliers para a variável sulfatos
q1_sulp = vinhos_total['sulfatos'].quantile(0.25)
q3_sulp = vinhos_total['sulfatos'].quantile(0.75)
IQR_sulp = q3_sulp - q1_sulp
```

```
print("IQR da variável sulfatos: {}\n".format(round(IQR_sulp, 2)))

# Definindo os limites para a variável sulfatos
sup_sulp = q3_sulp + 1.5 * IQR_sulp
inf_sulp = q1_sulp - 1.5 * IQR_sulp

print("Limite superior de sulfatos: {}".format(round(sup_sulp, 2)))
print("Limite inferior de sulfatos: {}".format(round(inf_sulp, 2)))

# In[69]:


cut_sulp = len(vinhos_total[vinhos_total['sulfatos'] < 0.28]) +
len(vinhos_total[vinhos_total['sulfatos'] > 1.0])

print("As entradas da variável sulfatos fora dos limites representam {} % do
dataset.\n".format(round((cut_sulp / vinhos_total.shape[0]) * 100, 2)))

# Remover as entradas que estão fora dos limites de corte das variáveis residual sugar, cloretos e
sulfatos:

# In[70]:


vinhos_total_clean = vinhos_total.copy()

vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['açúcar residual'] > 3.65].index,
axis=0, inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['açúcar residual'] < 0.85].index,
axis=0, inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['cloretos'] > 0.12].index, axis=0,
inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['cloretos'] < 0.04].index, axis=0,
inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['sulfatos'] > 1.0].index, axis=0,
inplace=True)
vinhos_total_clean.drop(vinhos_total_clean[vinhos_total_clean['sulfatos'] < 0.28].index, axis=0,
inplace=True)

# Plotando o boxplot para as variáveis:

# In[71]:


# Configurando o plot
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
fig.delaxes(ax[1,1])

sns.boxplot(vinhos_total_clean['açúcar residual'], ax=ax[0, 0])
sns.boxplot(vinhos_total_clean['cloretos'], ax=ax[0, 1])
sns.boxplot(vinhos_total_clean['sulfatos'], ax=ax[1, 0])

plt.tight_layout()

# Os boxplots mostram alguns outliers, no entanto, esses estão sendo calculados em relação ao novo
dataset e a limpeza levou em conta os dados originais.

# # Verificação do balanceamento dos dados
# Como é citado na descrição do conjunto de dados, a variável classe (qualidade) está
desbalanceada. Utilizando um gráfico, fica mais fácil percebemos este desbalanceamento.
```

```
# In[72]:


# Construindo o gráfico
fig, ax = plt.subplots(figsize=(10,8))

sns.countplot(vinhos_total_clean['qualidade'], color='b', ax=ax)
sns.despine()

plt.tight_layout()

# Pelo gráfico, fica fácil perceber que a maioria dos notas ficou em 5 ou 6.
#
# Vamos analisar o balanceamento para a variável qualidade_cat.

# In[73]:


# Porcentagem de cada classe
print("A classe 0 representa {} % de todas as
entradas.".format(round((len(vinhos_total_clean[vinhos_total_clean['qualidade_cat'] == 0]) /
vinhos_total_clean.shape[0]) * 100, 2)))
print("A classe 1 representa {} % de todas as
entradas.".format(round((len(vinhos_total_clean[vinhos_total_clean['qualidade_cat'] == 1]) /
vinhos_total_clean.shape[0]) * 100, 2)))

# Pela porcentagem de cada classe, percebemos que a classe 1 representa apenas 14.69% do total,
isso já é um desbalanceamento considerável. Para uma melhor visualização do desbalanceamento farei
um gráfico de barras da variável qualidade_cat.

# In[74]:


# Cronstruindo o gráfico de barras
fig, ax = plt.subplots(figsize=(6,6))

sns.countplot(vinhos_total_clean['qualidade_cat'], color='b', ax=ax)
sns.despine()

ax.set_title("Distribuição das Classes", fontsize=16)
ax.set_xlabel("Classes", fontsize=14)

plt.tight_layout()

# Pela Distribuição de Classes, percebemos claramente o desbalanceamento, tendo muito mais
entradas da classe 0 do que da classe 1.

# # Preparação dos dados

# ### Separação dos dados em treino e teste
#
# Uma etapa de grande importância para a construção de modelos de Machine Learning é separação do
conjunto de dados em treino e teste, pois se não fizermos isso é, bem provável, que ocorra
overfitting (sobreajuste).

# In[75]:


# Separando os dados entre feature matrix e target vector
X = vinhos_total_clean.drop(['qualidade', 'qualidade_cat'], axis=1)
y = vinhos_total_clean['qualidade_cat'] # Pois usaremos apenas a separação entre "ruim" ou "bom"
(0 ou 1)
```

```
# Dividindo os dados entre treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)

# #### Balanceamento dos dados
#
# Para realizar o balanceamento dos dados, existem vários métodos complexos tais como Recognition-based Learning e Cost-sensitive Learning e métodos mais simples que vem sendo amplamente utilizados com ótimos resultados como Over-sampling e Under-sampling.
#
# Neste projeto, será utilizado o método Under-sampling que foca na classe majoritário para balancear o conjunto de dados, ou seja, elimina aleatoriamente entradas da classe com maior quantidade de ocorrências.
#
# Assim:

# In[76]:


# Definindo o modelo para balancear
und = RandomUnderSampler()

X_und, y_und = und.fit_resample(X_train, y_train)

# Verificando o balanceamento dos dados
print(pd.Series(y_und).value_counts(), "\n")

# Plotando a nova Distribuição de Classes
fig, ax = plt.subplots(figsize=(6,6))

sns.countplot(pd.Series(y_und), color='b', ax=ax)

sns.despine()

ax.set_title("Distribuição das Classes", fontsize=16)
ax.set_xlabel("Classes", fontsize=14)

plt.tight_layout()

# Após o balanceamento, percebemos as classes agora representam 50% cada uma do conjunto de dados, ou seja, não há mais a diferença como havia anteriormente.

# ### Correlação entre os atributos
#
# Farei um mapa de calor (heatmap) para verificar a relação entre as variáveis utilizando o método de Pearson, essa correlação será feita com os dados antes e depois do balanceamento.

# In[77]:


# Construindo o heatmap
fig, ax = plt.subplots(figsize=(16, 6), nrows=1, ncols=2)
fig.suptitle("Matriz de Correlação")

sns.heatmap(X_train.corr().abs(), cmap='YlGn', linecolor='#eeeeee', linewidths=0.1, ax=ax[0])
ax[0].set_title("Desbalanceado")

sns.heatmap(X_und.corr().abs(), cmap='YlGn', linecolor='#eeeeee', linewidths=0.1, ax=ax[1])
ax[1].set_title("Balanceado")

plt.tight_layout()
```

```
# Como podemos perceber pela Matriz de Correlação, a correlação entre as variáveis parece não ter
sofrido grandes alterações após o balanceamento.

# Poderia ser utilizado algoritmos supervisionados como o K-means pra predizer em qual categoria
um vinho se encontra. Porém, consideramos que isso não faria sentido para rodar os modelos não
supervisionados.
#
# Pode fazer sentido analisar o vinho de maneira separada por tipo já que muitas variáveis tendem
a se comportar de forma diferente vamos iniciar a preparação dos dados, separando o dataset em 2.
#

# In[78]:


dados_base_tinto = vinhos_total[vinhos_total['tipo'] == 0].iloc[:, :15].copy()
dados_base_branco = vinhos_total[vinhos_total['tipo'] != 0].iloc[:, :15].copy()

# # Modelos de *Machine Learning*
# Após a preparação dos dados, vamos construir três modelos de Machine Learning e comparar o
desempenho de cada uma deles e em cada base de tipo de vinho (Tinto e Branco). Os modelos
utilizados serão:
#
#  - Random Forest (Floresta Aleatória);
#  - XGBoost;
#  - Regressão Logística.

# ### *Random Forest*
#
# O Random Forest é um modelo baseado em árvores de decisão e para esses tipos de modelos não é
necessário fazer uma padronização dos dados, sendo assim, vamos utilizar o modelo diretamente nos
dados balanceados.

# #### Para base dos vinhos Tinto: dados_base_tinto

# In[79]:


# Separando os dados entre feature matrix e target vector
dados_base_tinto_clean = vinhos_total_clean.copy()

X = dados_base_tinto_clean.drop(['qualidade', 'qualidade_cat'], axis=1)
y = dados_base_tinto_clean['qualidade_cat'] # Pois usaremos apenas a separação entre "ruim" ou
"bom" (0 ou 1)

# Dividindo os dados entre treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)

# In[80]:


# Modelo Random Forest
rf_model = RandomForestClassifier()

# Definindo o melhor parâmetro
parameters = {'n_estimators': range(25, 1000, 25)}

kfold = StratifiedKFold(n_splits=5, shuffle=True)

rf_clf = GridSearchCV(rf_model, parameters, cv=kfold)
rf_clf.fit(X_und, y_und)
```

```
# Visualizar o melhor parâmetro
print("Melhor parâmetro: {}".format(rf_clf.best_params_))

# Agora, para o número de estimadores igual a 375, vamos analisar o desempenho do modelo.

# In[81]:


# Definindo o modelo com n_estimators igual a 375
rf_model = RandomForestClassifier(n_estimators = 375)

# Fit do modelo
rf_model.fit(X_und, y_und)

# Testando o modelo
y_pred_rf = rf_model.predict(X_test)
y_prob_rf = rf_model.predict_proba(X_test)


# In[39]:


# Relatório de classificação
print("Relatório de classificação para o Random Forest:\n", classification_report(y_test,
y_pred_rf, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_rf) * 100, 2)))

# In[82]:


# Matriz de confusão
fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_rf, normalize='true'), annot=True, ax=ax)

ax.set_title('Matriz de Confusão Normalizada')
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()

# Como para conjunto de dados desbalanceados, a acurácia não é um bom indicador de desempenho.
Então, utilizei a Área sob a curva (AUC ROC) que é uma indicador interessante e apresentou um
valor de 88.42%.
#
# Ainda, na Matriz de Confusão Normalizada pode ser visualizada a taxa de acertos.
#
# Além disso, vou plotar a curva ROC:

# In[134]:


# Converter y_test em valores numéricos
y_test_numeric = y_test.cat.codes

# Treinar o modelo de classificação
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Calcular a probabilidade das classes positivas
```

```
y_pred_proba = rf_model.predict_proba(X_test)[:, 1]

# Calcular a curva ROC
fpr, tpr, thresholds = roc_curve(y_test_numeric, y_pred_proba)

# Plotar a curva ROC
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('Taxa de Falso Positivo')
plt.ylabel('Taxa de Verdadeiro Positivo')
plt.title('Curva ROC')
plt.show()

# ## XGBoost
#
# O XGBoost é um framework que utiliza gradient boosting que é uma técnica de aprendizado de
máquina para problemas de regressão e classificação, que produz um modelo de previsão na forma de
um ensemble de modelos de previsão fracos, geralmente árvores de decisão. Ela constrói o modelo em
etapas, como outros métodos de boosting, e os generaliza, permitindo a otimização de uma função de
perda diferenciável arbitrária.
#
# Primeiro, vou ajustar o número de estimadores e para isso vou definir o parâmetro learning_rate
= 0.1 e verbosity=0 (O parâmetro verbosity=0 é utilizado para não mostrar mensagens na tela
enquanto o código é utilizado).

# In[84]:


# Modelo XGBoost
xgb_model = XGBClassifier(learning_rate=0.1)

# Definindo os melhores parâmetros
param_gs = {'n_estimators': range(0, 1000, 50)}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))

# Com o número de estimadores definidos como 100, vou ajustar os parâmetros max_depth e
min_child_weight.

# In[88]:


# Modelo XGBoost
xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=100, verbosity=0)

# Definindo os melhores parâmetros
param_gs = {
    'max_depth': range(1, 8, 1),
    'min_child_weigth': range(1, 5, 1),
    }

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
```

```
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))

# Tendo os valores de max_depth = 1 e min_child_weigth = 1, vou oajustar o parâmetro gamma.

# In[89]:


# Modelo XGBoost
xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=100, max_depth=3, min_child_weigth=1,
verbosity=0)

# Definindo os melhores parâmetros
param_gs = {'gamma': [i/10.0 for i in range(0,5)]}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))

# Como o gamma=0.0, por fim, irei testar quatro valores para learning_rate e qual apresenta o
melhor resultado.

# In[91]:


# Modelo XGBoost
xgb_model = XGBClassifier(n_estimators=100, max_depth=3, min_child_weigth=1, gamma=0.0,
verbosity=0)

# Definindo os melhores parâmetros
param_gs = {'learning_rate': [0.001, 0.01, 0.1, 1]}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))

# Com os parâmetros definidos, vou avaliar o desempenho do modelo final.

# In[92]:


# Modelo XGBoost final
xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=100, max_depth=3, min_child_weigth=1,
gamma=0.0, verbosity=0)

# Treinando o modelo
xgb_model.fit(X_und, y_und)

# Fazendo previsões
y_pred_xgb = xgb_model.predict(X_test)
```

```
y_prob_xgb = xgb_model.predict_proba(X_test)

# In[190]:


# Relatório de classificação
print("Relatório de classificação para o XGBClassifier:\n", classification_report(y_test,
y_pred_xgb, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_xgb) * 100, 2)))

# In[93]:


# Matriz de confusão
fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_xgb, normalize='true'), annot=True, ax=ax)

ax.set_title('Matriz de Confusão Normalizada')
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()

# Como para conjunto de dados desbalanceados, a acurácia não é um bom indicador de desempenho.
Então, utilizei a Área sob a curva (AUC ROC) que é uma indicador interessante e apresentou um
valor de 81.53%.
#
# Ainda, na Matriz de Confusão Normalizada pode ser visualizada a taxa de acertos.

# In[94]:


# Ajustar o modelo aos dados de treinamento
xgb_model.fit(X_und, y_und)

# Converter y_test para valores numéricos
y_test_numeric = y_test.cat.codes

# Calcular as probabilidades de previsão
y_pred_proba = xgb_model.predict_proba(X_test)

# Calcular a pontuação ROC
fpr, tpr, _ = roc_curve(y_test_numeric, y_pred_proba[:, 1])

# Plotar a curva ROC
fig, ax = plt.subplots(figsize=(10,8))
ax.plot(fpr, tpr, label="XGBClassifier")
ax.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax.set_xlabel("Taxa de Falso Positivo")
ax.set_ylabel("Taxa de Verdadeiro Positivo")
ax.set_title("Curva ROC para o XGBClassifier")
ax.legend()
plt.show()


# ## Regressão Logística
#
# A regressão logística é uma técnica estatística que tem como objetivo produzir, a partir de um
conjunto de observações, um modelo que permita a predição de valores tomados por uma variável
```

```
categórica, frequentemente binária, a partir de uma série de variáveis explicativas contínuas e/ou
binárias.
#
# Diferentemente dos modelos Random Forest e XGBoost, para a Regressão Logística é necessário
padronizar os dados antes de treiná-los no modelo. Primeiro, ajustarei o parâmetro C da Regressão
Logisitca.

# In[95]:


# Regressão logística
rl_model = LogisticRegression()
# Padronizando os dados de treino
scaler = StandardScaler().fit(X_und)
X_und_std = scaler.transform(X_und)

# Definindo o melhor parâmetro
param_rl = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

# Identificando os melhor parâmetro
kfold = StratifiedKFold(n_splits=5, shuffle=True)

rl_clf = GridSearchCV(rl_model, param_rl, cv=kfold)
rl_clf.fit(X_und_std, y_und)

# Visualizar o melhor parâmetro
print("Melhor parâmetro: {}".format(rl_clf.best_params_))

# Com o parâmetro ajustado, vou avaliar o desmepenho do modelo e para auxiliar vou criar um fluxo
de trabalho utilizando uma pipeline.

# In[96]:


# Regressão Logística com pipeline
rl_model = make_pipeline(StandardScaler(), LogisticRegression(C=1))

# Treinando o modelo
rl_model.fit(X_und, y_und)

# Fazendo previsões
y_pred_rl = rl_model.predict(X_test)
y_prob_rl = rl_model.predict_proba(X_test)

# In[98]:


# Relatório de classificação
print("Relatório de classificação para a Regressão Logística:\n", classification_report(y_test,
y_pred_rl, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_rl) * 100, 2)))


# In[99]:


# Matriz de confusão
fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_rl, normalize='true'), annot=True, ax=ax)
```

```
ax.set_title('Matriz de Confusão Normalizada')
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()

# Como para conjunto de dados desbalanceados, a acurácia não é um bom indicador de desempenho.
Então, utilizei a Área sob a curva (AUC ROC) que é uma indicador interessante e apresentou um
valor de 76.13%.
#
# Ainda, na Matriz de Confusão Normalizada pode ser visualizada a taxa de acertos.

# In[100]:


# Ajustar o modelo aos dados de treinamento
rl_model.fit(X_und, y_und)

# Converter y_test para valores numéricos
y_test_numeric = y_test.cat.codes

# Calcular as probabilidades de previsão
y_pred_proba = rl_model.predict_proba(X_test)

# Calcular a pontuação ROC
fpr, tpr, _ = roc_curve(y_test_numeric, y_pred_proba[:, 1])

# Plotar a curva ROC
fig, ax = plt.subplots(figsize=(10,8))
ax.plot(fpr, tpr, label="Regressão Logística")
ax.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax.set_xlabel("Taxa de Falso Positivo")
ax.set_ylabel("Taxa de Verdadeiro Positivo")
ax.set_title("Curva ROC para a Regressão Logística")
ax.legend()
plt.show()


# # Random Forest

# ### Para base dos vinhos Branco: dados_base_branco

# In[101]:


dados_base_branco_clean = vinhos_total_clean.copy()

X = dados_base_branco_clean.drop(['qualidade', 'qualidade_cat'], axis=1)
y = dados_base_branco_clean['qualidade_cat'] # Pois usaremos apenas a separação entre "ruim" ou
"bom" (0 ou 1)

# Dividindo os dados entre treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)
rf_model = RandomForestClassifier()
# Definindo o melhor parâmetro
parameters = {'n_estimators': range(25, 1000, 25)}

kfold = StratifiedKFold(n_splits=5, shuffle=True)

rf_clf = GridSearchCV(rf_model, parameters, cv=kfold)
rf_clf.fit(X_und, y_und)
```

```
# Visualizar o melhor parâmetro
print("Melhor parâmetro: {}".format(rf_clf.best_params_))

# Agora, para o número de estimadores igual a 750, vamos analisar o desempenho do modelo.

# In[102]:


# Definindo o modelo com n_estimators igual a 750
rf_model = RandomForestClassifier(n_estimators = 750)

# Fit do modelo
rf_model.fit(X_und, y_und)

# Testando o modelo
y_pred_rf = rf_model.predict(X_test)
y_prob_rf = rf_model.predict_proba(X_test)

# In[165]:


# Relatório de classificação
print("Relatório de classificação para o Random Forest:\n", classification_report(y_test,
y_pred_rf, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_rf) * 100, 2)))

# In[103]:


# Matriz de confusão
fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_rf, normalize='true'), annot=True, ax=ax)

ax.set_title('Matriz de Confusão Normalizada')
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()

# Como para conjunto de dados desbalanceados, a acurácia não é um bom indicador de desempenho.
Então, utilizei a Área sob a curva (AUC ROC) que é uma indicador interessante e apresentou um
valor de 84.91%.
#
# Ainda, na Matriz de Confusão Normalizada pode ser visualizada a taxa de acertos.
#
# Além disso, vou plotar a curva ROC:

# In[104]:


# Converter y_test para valores numéricos
y_test_numeric = y_test.cat.codes

# Calcular a pontuação com os dados de teste
vis_rf.score(X_test, y_test_numeric)

# Mostrar a figura
vis_rf.show()
```

```
# ## XGBoost
#
# O XGBoost é um framework que utiliza gradient boosting que é uma técnica de aprendizado de
máquina para problemas de regressão e classificação, que produz um modelo de previsão na forma de
um ensemble de modelos de previsão fracos, geralmente árvores de decisão. Ela constrói o modelo em
etapas, como outros métodos de boosting, e os generaliza, permitindo a otimização de uma função de
perda diferenciável arbitrária.
#
# Primeiro, vou ajustar o número de estimadores e para isso vou definir o parâmetro learning_rate
= 0.1 e verbosity=0 (O parâmetro verbosity=0 é utilizado para não mostrar mensagens na tela
enquanto o código é utilizado).

# In[105]:


xgb_model = XGBClassifier(learning_rate=0.1)

# Definindo os melhores parâmetros
param_gs = {'n_estimators': range(0, 1000, 50)}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))

# Com o número de estimadores definidos como 400, vou ajustar os parâmetros max_depth e
min_child_weight.

# In[106]:


xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=400, verbosity=0)

# Definindo os melhores parâmetros
param_gs = {
    'max_depth': range(1, 8, 1),
    'min_child_weigth': range(1, 5, 1),
    }

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))

# Tendo os valores de max_depth = 2 e min_child_weigth = 1, vou oajustar o parâmetro gamma.

# In[107]:


xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=400, max_depth=2, min_child_weigth=1,
verbosity=0)

# Definindo os melhores parâmetros
```

```
param_gs = {'gamma': [i/10.0 for i in range(0,5)]}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))

# Com o melhor resultado.

# In[108]:


xgb_model = XGBClassifier(n_estimators=400, max_depth=2, min_child_weigth=1, gamma=0.4,
verbosity=0)

# Definindo os melhores parâmetros
param_gs = {'learning_rate': [0.001, 0.01, 0.1, 1]}

# Identificando os melhores parâmetros
kfold = StratifiedKFold(n_splits=5, shuffle=True)

xgb_clf = GridSearchCV(xgb_model, param_gs, cv=kfold)
xgb_clf.fit(X_und, y_und)

# Visualizar o melhor parâmetro
print("Melhores parâmetros: {}".format(xgb_clf.best_params_))

# In[109]:


xgb_model = XGBClassifier(learning_rate=0.1, n_estimators=400, max_depth=2, min_child_weigth=1,
gamma=0.4, verbosity=0)

# Treinando o modelo
xgb_model.fit(X_und, y_und)

# Fazendo previsões
y_pred_xgb = xgb_model.predict(X_test)
y_prob_xgb = xgb_model.predict_proba(X_test)

# In[110]:


# Relatório de classificação
print("Relatório de classificação para o XGBClassifier:\n", classification_report(y_test,
y_pred_xgb, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_xgb) * 100, 2)))

# In[111]:


fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_xgb, normalize='true'), annot=True, ax=ax)

ax.set_title('Matriz de Confusão Normalizada')
```

```
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()

# Como para conjunto de dados desbalanceados, a acurácia não é um bom indicador de desempenho.
Então, utilizei a Área sob a curva (AUC ROC) que é uma indicador interessante e apresentou um
valor de 82.65%.
# Ainda, na Matriz de Confusão Normalizada pode ser visualizada a taxa de acertos.

# In[112]:


# Ajustar o modelo aos dados de treinamento
xgb_model.fit(X_und, y_und)

# Converter y_test para valores numéricos
y_test_numeric = y_test.cat.codes

# Calcular as probabilidades de previsão
y_pred_proba = xgb_model.predict_proba(X_test)

# Calcular a pontuação ROC
fpr, tpr, _ = roc_curve(y_test_numeric, y_pred_proba[:, 1])

# Plotar a curva ROC
fig, ax = plt.subplots(figsize=(10,8))
ax.plot(fpr, tpr, label="XGBClassifier")
ax.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax.set_xlabel("Taxa de Falso Positivo")
ax.set_ylabel("Taxa de Verdadeiro Positivo")
ax.set_title("Curva ROC para o XGBClassifier")
ax.legend()
plt.show()

# ### Regressão Logística

# In[113]:


rl_model = LogisticRegression()
# Padronizando os dados de treino
scaler = StandardScaler().fit(X_und)
X_und_std = scaler.transform(X_und)

# Definindo o melhor parâmetro
param_rl = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

# Identificando os melhor parâmetro
kfold = StratifiedKFold(n_splits=5, shuffle=True)

rl_clf = GridSearchCV(rl_model, param_rl, cv=kfold)
rl_clf.fit(X_und_std, y_und)

# Visualizar o melhor parâmetro
print("Melhor parâmetro: {}".format(rl_clf.best_params_))

# Com o parâmetro ajustado, vou avaliar o desmepenho do modelo e para auxiliar vou criar um fluxo
de trabalho utilizando uma pipeline.

# In[114]:
```

```
rl_model = make_pipeline(StandardScaler(), LogisticRegression(C=10))

# Treinando o modelo
rl_model.fit(X_und, y_und)

# Fazendo previsões
y_pred_rl = rl_model.predict(X_test)
y_prob_rl = rl_model.predict_proba(X_test)

# In[185]:


# Relatório de classificação
print("Relatório de classificação para a Regressão Logística:\n", classification_report(y_test,
y_pred_rl, digits=4))

# Área sob a curva
print("Área sob a curva (AUC):\t{}%".format(round(roc_auc_score(y_test, y_pred_rl) * 100, 2)))

# In[115]:


# Matriz de confusão
fig, ax = plt.subplots()

sns.heatmap(confusion_matrix(y_test, y_pred_rl, normalize='true'), annot=True, ax=ax)

ax.set_title('Matriz de Confusão Normalizada')
ax.set_ylabel('Verdadeiro')
ax.set_xlabel('Previsto')

plt.tight_layout()

# Como para conjunto de dados desbalanceados, a acurácia não é um bom indicador de desempenho.
Então, utilizei a Área sob a curva (AUC ROC) que é uma indicador interessante e apresentou um
valor de 77.54%.
#
# Ainda, na Matriz de Confusão Normalizada pode ser visualizada a taxa de acertos.

# In[116]:


# Ajustar o modelo aos dados de treinamento
rl_model.fit(X_und, y_und)

# Converter y_test para valores numéricos
y_test_numeric = y_test.cat.codes

# Calcular as probabilidades de previsão
y_pred_proba = rl_model.predict_proba(X_test)

# Calcular a pontuação ROC
fpr, tpr, _ = roc_curve(y_test_numeric, y_pred_proba[:, 1])

# Plotar a curva ROC
fig, ax = plt.subplots(figsize=(10,8))
ax.plot(fpr, tpr, label="Regressão Logística")
ax.plot([0, 1], [0, 1], color='gray', linestyle='--')
ax.set_xlabel("Taxa de Falso Positivo")
ax.set_ylabel("Taxa de Verdadeiro Positivo")
ax.set_title("Curva ROC para a Regressão Logística")
```

```
ax.legend()
plt.show()


# ## Comparativo grafico das tecnicas para cada tipo de vinho

# In[129]:


# Dados para o primeiro gráfico
tecnicas = ["Random Forest", "XGBClassifiero", "Regressão Logisitca"]
acuracias_tinto = [88.42, 81.53, 76.13]

# Dados para o segundo gráfico
acuracias_branco = [84.91, 82.65, 77.54]

# Configuração das figuras e dos eixos
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

# Primeiro gráfico
ax1.bar(tecnicas, acuracias_tinto, color='red')
ax1.set_xlabel('Técnica aplicada para vinho Tinto')
ax1.set_ylabel('Acurácia')
ax1.set_title('Gráfico de Barras - Acurácia por Técnica')

# Segundo gráfico
ax2.bar(tecnicas, acuracias_branco, color='green')
ax2.set_xlabel('Técnica aplicada para vinho Branco')
ax2.set_ylabel('Acurácia')
ax2.set_title('Gráfico de Barras - Acurácia por Técnica')

# Ajusta o espaçamento entre as subplots
plt.tight_layout()

# Exibe os gráficos
plt.show()
```