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

César Valença Porto

VINHOS: ANÁLISE DE FATORES DETERMINANTES DE QUALIDADE UTILIZANDO DADOS

Recife, Pernambuco
2021

César Valença Porto

VINHOS: ANÁLISE DE FATORES DETERMINANTES DE QUALIDADE UTILIZANDO DADOS

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.

Recife, Pernambuco
2021

SUMÁRIO

1. Introdução

1.1. Contextualização

Atualmente conseguimos encontrar diversos rankings e classificações de vinhos novos e antigos disponíveis no mundo, não havendo um consenso quanto às escalas de classificação.

Encontramos inclusive guias especializados por região com atualização periódica que nos fornece o panorama atualizado do que existe disponível no mercado com relação a vinhos por região, tipos de uva, modo de fabricação, faixa de preços e tantos outros critérios.

No entanto, tem-se como consenso que esta ou aquela classificação, esse ou aquele guia, são baseados na avaliação de pessoas de renome que tem uma reputação conquistada em anos de estudos e provas, e em alguns casos, equipes de pessoas treinadas que atribuem notas em escala de 0 a 10, 0 a 100, ou ainda em quantidade de estrelas de 1 a 5. Alguns sites e aplicativos também fornecem classificações a partir de coleta de opiniões de público em geral.

Percebemos à primeira vista que estes critérios parecem ser subjetivos, pois refletem, grosso modo, a opinião de uma pessoa ou grupo de pessoas treinadas, as quais certamente possuem um gosto particular, suas preferências pessoais, (não podemos deixar de considerar também as condições do momento da prova, o humor e outros fatores subjetivos) o que não nos garante a imparcialidade quanto ao atribuir uma classificação para determinado vinho.

Seria de grande valia se tivéssemos um modo de validar, confirmar ou testar as notas dadas por esses especialistas, baseados em um conjunto de dados estruturados e relevantes na determinação do paladar, olfato e experiencia de consumo de vinhos.

Aqui temos um problema possível: como determinar qual a melhor classificação ou se podemos realmente considerar a nota obtida por um vinho em determinado score como indicador de qualidade, e assim seja considerado fator que influencie minha decisão de compra.

1.2. O problema proposto

A proposta desse trabalho é sugerir um modo de classificar vinhos quanto à sua qualidade, sem a possibilidade de tendência para os gostos pessoais e/ou fatores subjetivos, ou ainda, ratificar através de um modelo computacional as notas e ranking atribuídos pelos especialistas.

Para tanto podemos imaginar que se obtivermos dados mensuráveis (no nosso caso, medições físico-químicas), poderemos classificar ou descobrir o que alguns agrupamentos de vinhos têm em comum visando assim, quem sabe, ratificar ou contrapor a classificações atribuídas por especialistas, no caso dos rankings supracitados, e assim, de maneira cega, obtermos um modelo de predição a partir de amostras. E quem sabe, definir um modelo para inferência quanto a possíveis intervenções no trato e cultivo das uvas para que tal ou qual índice seja alterado em novas safras, com a finalidade de obter um vinho excelente.

Convêm ressaltar a diferença entre a classificação baseada em medições fisico-quimicas e a baseada em experimentação e degustação. A proposta é unificar ou achar um elo de ligação para esses dois critérios.

Faremos uma breve delineação da questão, utilizando as clássicas questões do framework 5W:

(*Why*/Por quê?) Visando otimizar a qualidade dos vinhos produzidos, através da dedução dos fatores precisos que definem a qualidade do produto, dando condição ao especialista técnico intervir no modo produtivo do cultivo à vinificação, nas etapas onde há espaço para variabilidade, ou seja, atuar na fonte das causas.

(*Who*/Quem?) Foram encontrados *datasets* na web no site da UCI (https://archive.ics.uci.edu/ml/datasets/Wine+Quality), disponibilizados por Paulo Cortez, Universidade do Minho, Guimarães, Portugal, http://www3.dsi.uminho.pt/pcortez), contendo dados físico-químicos de amostas.

(*What*/ O que?): A análise será feita em cima de *datasets* contendo medições químicas de indicadores coletados de vinhos brancos e tintos da denominação Vinho Verde (provenientes da região noroeste de Portugal), visando entender o que é

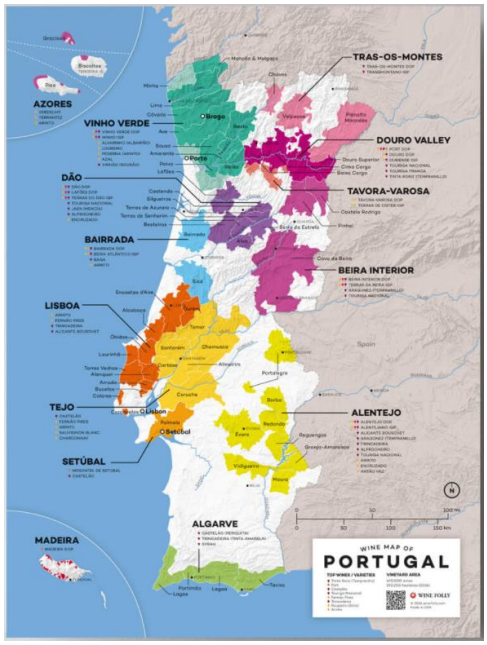
fisicamente significativo na definição dos critérios de qualidade, o que até então pode ser subjetivo quando apoiados apenas na opinião de especialistas.

(*Where*/ Onde?): Esta análise tem como objeto um modelo computacional sob a forma de produto de consumo e produção em escala global, além disso, devemos considerar que atualmente os especialistas também possuem renome mundial quanto às suas classificações. Dessa forma, entende-se que não há limitação geográfica para a aplicação da pesquisa ou uso dos resultados, mesmo considerando a restrição dos dados de análise (vinhos do noroeste de Portugal).

(*When*/ Quando?): Serão analisados todos os dados das bases de dados coletadas, as quais foram disponibilizadas no ano 2009. Devemos considerar que os dados foram coletados anteriormente. No entanto, dada a atemporalidade do produto, não consideramos o fator temporal como impactante na definição deste trabalho.

2. Coleta de Dados

Os dados foram coletados no site da UCI (*University of California, Irvine*) e estão divididos em dois arquivos em formato CSV, relativos a dados coletados para vinhos brancos e vinhos tintos, todos variantes da denominação portuguesa de Vinho Verde, que não se refere à cor do vinho, mas a uma classificação geográfica.



A coleta foi feita do site https://archive.ics.uci.edu/ml/datasets/Wine+Quality , no dia 18/06/2021.

Ambos os arquivos possuem a mesma estrutura, e foram colocados no repositório do *Github* criado para este projeto, juntamente com o arquivo descritivo do *dataset* disponibilizado no mesmo link.

Este arquivo descritivo contém maiores informações sobre como os dados foram utilizados na pesquisa acadêmica que o originou, juntamente com a referência para este trabalho:

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.

Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

As bases de dados em conjunto (brancos e tintos), possuem 4.898 registros, e os arquivos possuem a mesma estrutura de dados.

São 11 atributos de entrada, relativos a testes físico-químicos, e 1 variável de saída, correspondente à classificação:

Nome da coluna/campo	**Tipo**
Fixed acidity	Entrada
Volatile acidity	Entrada
Citric acid	Entrada
Residual sugar	Entrada
Chlorides	Entrada
Free sulphur dioxide	Entrada
Total sulphur dixide	Entrada
Density	Entrada
pH	Entrada
Sulphates	Entrada
Alcohol	Entrada
Quality	Saída (score 0 a 10)

3. Processamento/Tratamento de Dados

Antes de iniciar a exploração dos dados de forma analítica, convém garantir que o trabalho será feito em cima de dados consistentes.

Para obtermos essa garantia, foi feito um trabalho de validação, o qual será descrito a seguir. Convém citar que para todas as etapas práticas do presente trabalho, foi utilizada a linguagem *Python* no ambiente *Google Colab*.

1. Os arquivos foram carregados usando o Pandas;

```
#Passo 3.1
import pandas as pd
df_tinto = pd.read_csv('https://raw.githubusercontent.com/caesarporto/PUC_TCC_DataScience/main/datasource/winequality_red.csv', sep=',')
df_branco = pd.read_csv('https://raw.githubusercontent.com/caesarporto/PUC_TCC_DataScience/main/datasource/winequality_white.csv', sep=',')
```

2. Ratificamos as dimensões dos arquivos, quanto a linhas e colunas de cada um;

```
[2] #Passo 3.2
    df_tinto.shape

    (1599, 12)
```

```
df_branco.shape

(4898, 12)
```

3. A fim de entender os tipos de dados de cada variável, foi feita uma amostragem de registros do início do arquivo e também de forma aleatória;

```
[4] #Passo 3.3 (a)
    df_tinto.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
#Passo 3.3 (b)
df_branco.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

```
[6] #Passo 3.3 (c)
    df_tinto.sample(10)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1172	9.7	0.420	0.46	2.1	0.074	5.0	16.0	0.99649	3.27	0.74	12.3	6
99	8.1	0.545	0.18	1.9	0.080	13.0	35.0	0.99720	3.30	0.59	9.0	6
584	11.8	0.330	0.49	3.4	0.093	54.0	80.0	1.00020	3.30	0.76	10.7	7
753	8.3	0.650	0.10	2.9	0.089	17.0	40.0	0.99803	3.29	0.55	9.5	5
1525	6.7	0.480	0.08	2.1	0.064	18.0	34.0	0.99552	3.33	0.64	9.7	5
1017	8.0	0.180	0.37	0.9	0.049	36.0	109.0	0.99007	2.89	0.44	12.7	6
570	11.5	0.350	0.49	3.3	0.070	10.0	37.0	1.00030	3.32	0.91	11.0	6
535	9.1	0.220	0.24	2.1	0.078	1.0	28.0	0.99900	3.41	0.87	10.3	6
869	7.6	0.630	0.03	2.0	0.080	27.0	43.0	0.99578	3.44	0.64	10.9	6
552	9.5	0.460	0.24	2.7	0.092	14.0	44.0	0.99800	3.12	0.74	10.0	6

```
#Passo 3.3 (d)
df_branco.sample(10)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1101	6.3	0.17	0.42	2.8	0.028	45.0	107.0	0.99080	3.27	0.43	11.8	6
4585	5.0	0.33	0.23	11.8	0.030	23.0	158.0	0.99322	3.41	0.64	11.8	6
2449	6.7	0.27	0.26	2.3	0.043	61.0	181.0	0.99394	3.45	0.63	10.6	6
1106	5.2	0.24	0.45	3.8	0.027	21.0	128.0	0.99200	3.55	0.49	11.2	8
327	6.4	0.34	0.23	6.3	0.039	37.0	143.0	0.99440	3.19	0.65	10.0	6
400	6.3	0.20	0.40	1.5	0.037	35.0	107.0	0.99170	3.46	0.50	11.4	6
63	6.6	0.38	0.15	4.6	0.044	25.0	78.0	0.99310	3.11	0.38	10.2	6
1063	6.7	0.26	0.26	4.1	0.073	36.0	202.0	0.99560	3.30	0.67	9.5	5
1909	7.6	0.13	0.34	9.3	0.062	40.0	126.0	0.99660	3.21	0.39	9.6	5
4563	5.6	0.26	0.27	10.6	0.030	27.0	119.0	0.99470	3.40	0.34	10.7	7

4. Foi feita a detecção de registros duplicados, e o total de registros de cada arquivo foi obtido antes e depois da retirada das duplicidades. A opção por excluir as duplicidades se deve ao fato de a proposta da análise ser quanto a classificação, e não para estudo de população, onde a quantidade de indivíduos seria determinante. Caso se detecte a necessidade de considerar a população inteira, mesmo com duplicidades de classificação, este passo será modificado, sendo feita a referência a este passo;

```
[8] #Passo 3.4 (a)
    df_tinto.count()
```

```
fixed acidity           1599
volatile acidity        1599
citric acid             1599
residual sugar          1599
chlorides               1599
free sulfur dioxide     1599
total sulfur dioxide    1599
density                 1599
pH                      1599
sulphates               1599
alcohol                 1599
quality                 1599
dtype: int64
```

```
#Passo 3.4 (b)
df_branco.count()
```

```
fixed acidity           4898
volatile acidity        4898
citric acid             4898
residual sugar          4898
chlorides               4898
free sulfur dioxide     4898
total sulfur dioxide    4898
density                 4898
pH                      4898
sulphates               4898
alcohol                 4898
quality                 4898
dtype: int64
```

```
#Passo 3.4 (c)
df_tinto.drop_duplicates(inplace=True)
df_branco.drop_duplicates(inplace=True)
```

```
[11] #Passo 3.4 (d)
     df_tinto.count()
```

```
fixed acidity           1359
volatile acidity        1359
citric acid             1359
residual sugar          1359
chlorides               1359
free sulfur dioxide     1359
total sulfur dioxide    1359
density                 1359
pH                      1359
sulphates               1359
alcohol                 1359
quality                 1359
dtype: int64
```

```
#Passo 3.5 (e)
df_branco.count()
```

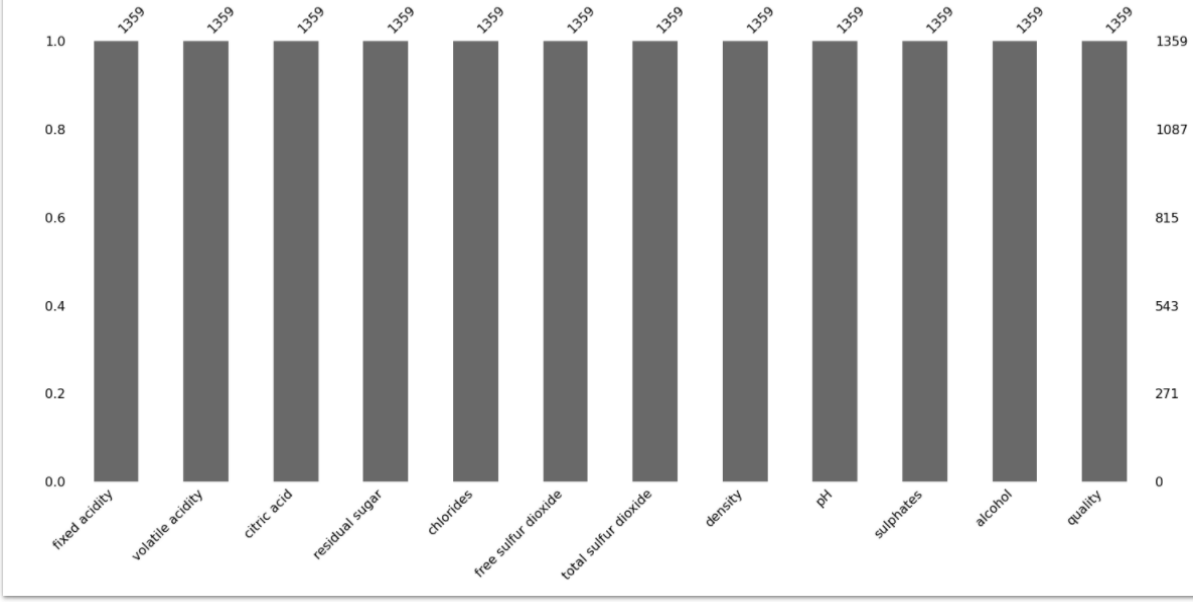
```
fixed acidity           3961
volatile acidity        3961
citric acid             3961
residual sugar          3961
chlorides               3961
free sulfur dioxide     3961
total sulfur dioxide    3961
density                 3961
pH                      3961
sulphates               3961
alcohol                 3961
quality                 3961
dtype: int64
```

5. Foi verificada a existência ou não de registros com valores vazios. Como não foram encontrados campos vazios em nenhum registro para nenhuma das variáveis, não houve necessidade de ação sobre eles.

Para descobrir que não temos campos vazio, foi usada a biblioteca **missingno** do Python, que nos permite ver a quantidade preenchida de cada atributo. Vemos que temos os atributos estão presentes, não falta nenhum;

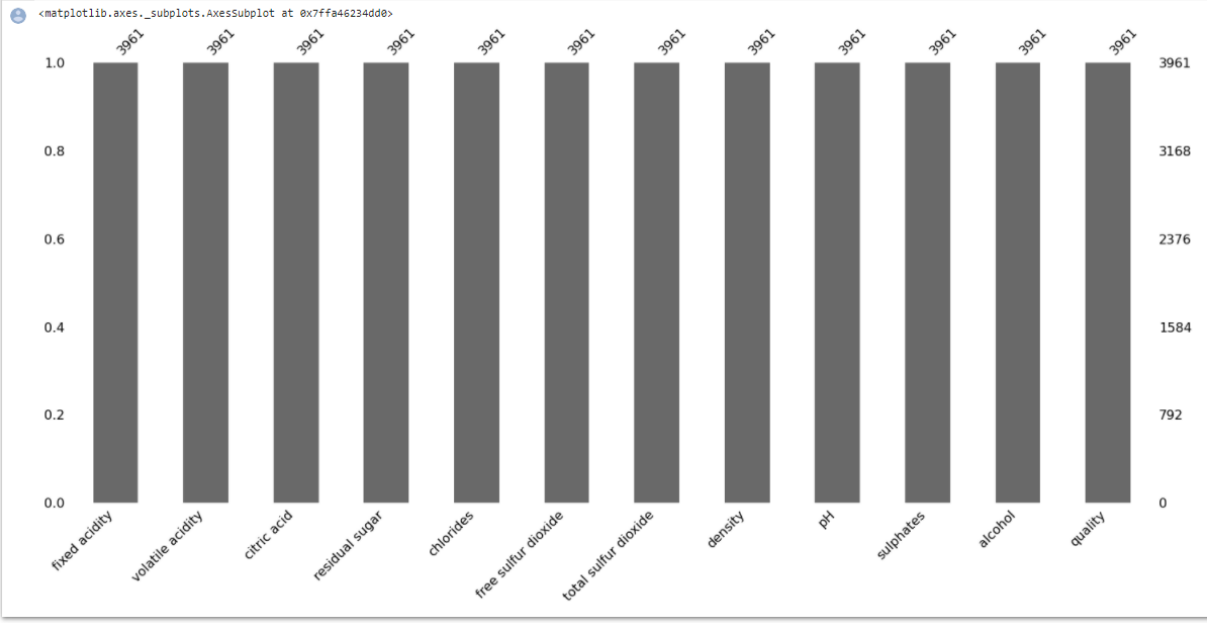
```
#Passo 3.5 (a)
import missingno as msno
msno.bar(df_tinto)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa4f754f50>
```




```
#Passo 3.5 (b)
msno.bar(df_branco)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa46234dd0>
```



Dessa forma entendemos agora ter um conjunto de dados consistente, sem valores nulos (missing values), registros duplicados, contendo 11 colunas de valores

numéricos de ponto flutuante e uma coluna contendo valores inteiros, sendo esta relativa à classificação (variável de saída).

Boa parte das análises anteriores pode ser obtida em único comando e extração, se utilizarmos a biblioteca *Pandas Profiling*, o qual usaremos mais detalhadamente para a exploração de dados.

4. Análise e Exploração dos Dados

Uma vez que temos os dados tratados, partindo para a exploração dos mesmos, foram usadas algumas abordagens de análise visando identificar fatores que ajudem no melhor entendimento do conjunto de dados, possibilitando em futura etapa a construção de um modelo de classificação o mais confiável possível.

4.1 Análise de distribuição e detecção de outliers

No primeiro momento foi feita uma análise da distribuição dos valores de cada variável, afim de encontrar valores outliers e assim poder decidir sobre a permanência ou retirada dos mesmos.

Incialmente são visualizados os valores obtidos das medidas descritivas, o que é de grande valor para um olho apurado em análise estatística.

```
df_tinto.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000
mean	8.310596	0.529478	0.272333	2.523400	0.088124	15.893304	46.825975	0.996709	3.309787	0.658705	10.432315	5.623252
std	1.736990	0.183031	0.195537	1.352314	0.049377	10.447270	33.408946	0.001869	0.155036	0.170667	1.082065	0.823578
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996700	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.430000	2.600000	0.091000	21.000000	63.000000	0.997820	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

```
[17] df_branco.describe()
```

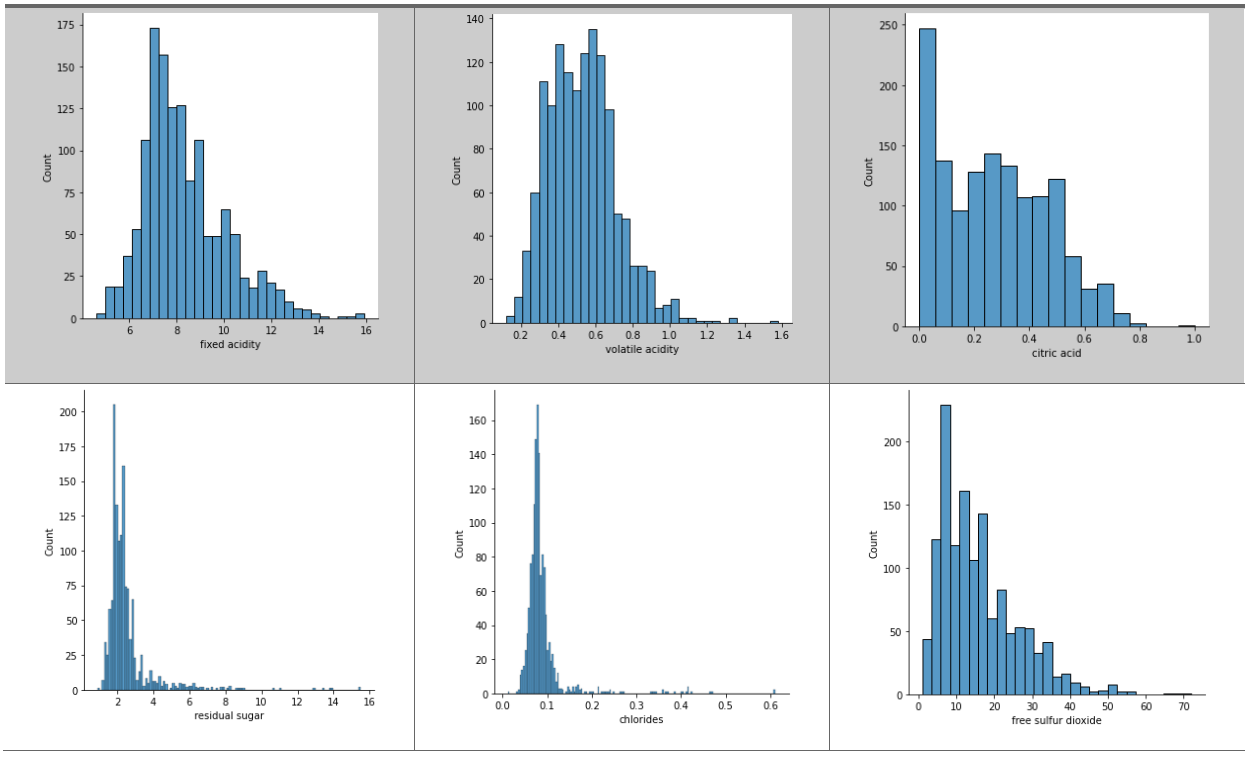
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	3961.000000	3961.000000	3961.000000	3961.000000	3961.000000	3961.000000	3961.000000	3961.000000	3961.000000	3961.000000	3961.000000	3961.000000
mean	6.839346	0.280538	0.334332	5.914819	0.045905	34.889169	137.193512	0.993790	3.195458	0.490351	10.589358	5.854835
std	0.866860	0.103437	0.122446	4.861646	0.023103	17.210021	43.129065	0.002905	0.151546	0.113523	1.217076	0.890683
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.300000	0.210000	0.270000	1.600000	0.035000	23.000000	106.000000	0.991620	3.090000	0.410000	9.500000	5.000000
50%	6.800000	0.260000	0.320000	4.700000	0.042000	33.000000	133.000000	0.993500	3.180000	0.480000	10.400000	6.000000
75%	7.300000	0.330000	0.390000	8.900000	0.050000	45.000000	166.000000	0.995710	3.290000	0.550000	11.400000	6.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	9.000000

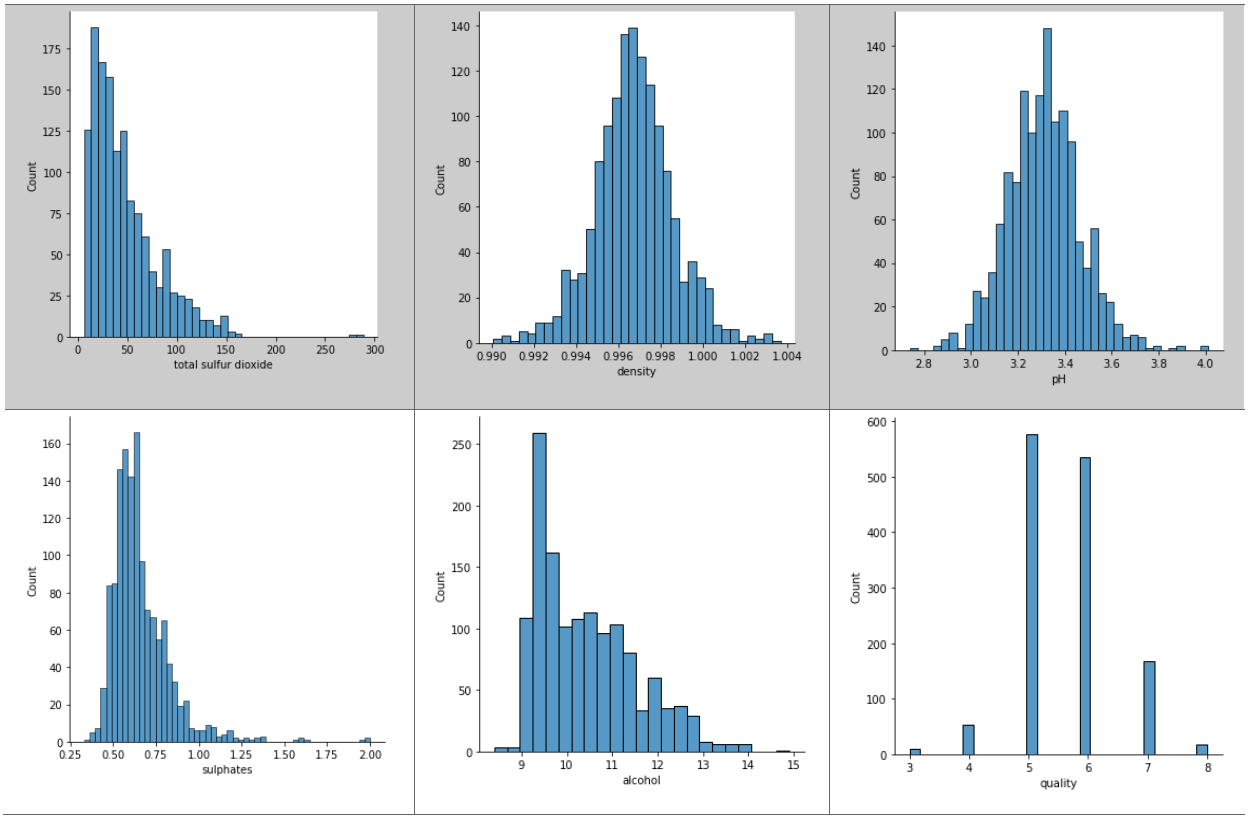
Percebemos a necessidade de complementar a análise descritiva utilizando gráficos de histograma, visando obter mais insights. Assim pudemos detectar as distribuições de cada variável para as amostras disponíveis.

```
[19] import matplotlib.pyplot as plt
     import seaborn as sns
     % matplotlib inline
```

```
sns.displot(df_tinto['fixed acidity'], kde=False)
sns.displot(df_tinto['volatile acidity'], kde=False)
sns.displot(df_tinto['citric acid'], kde=False)
sns.displot(df_tinto['residual sugar'], kde=False)
sns.displot(df_tinto['chlorides'], kde=False)
sns.displot(df_tinto['free sulfur dioxide'], kde=False)
sns.displot(df_tinto['total sulfur dioxide'], kde=False)
sns.displot(df_tinto['density'], kde=False)
sns.displot(df_tinto['pH'], kde=False)
sns.displot(df_tinto['sulphates'], kde=False)
sns.displot(df_tinto['alcohol'], kde=False)
sns.displot(df_tinto['quality'], kde=False)
```

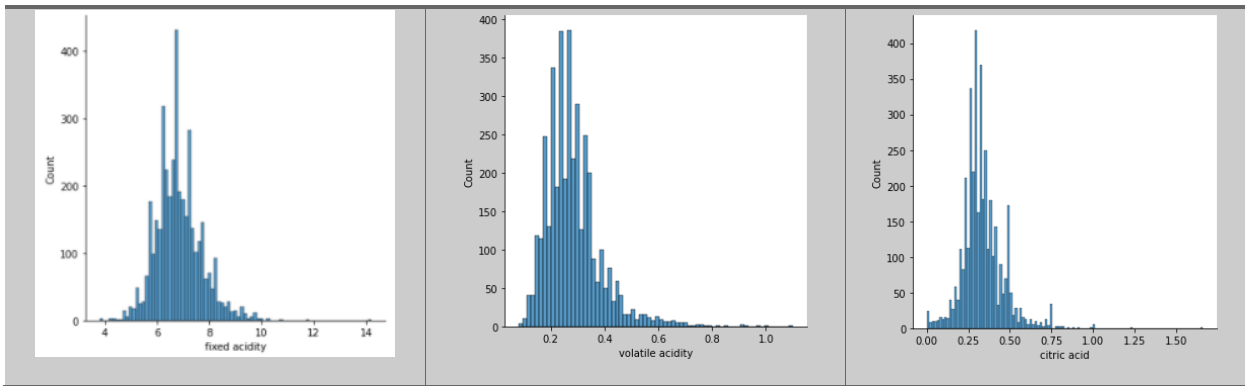
Vinhos tintos

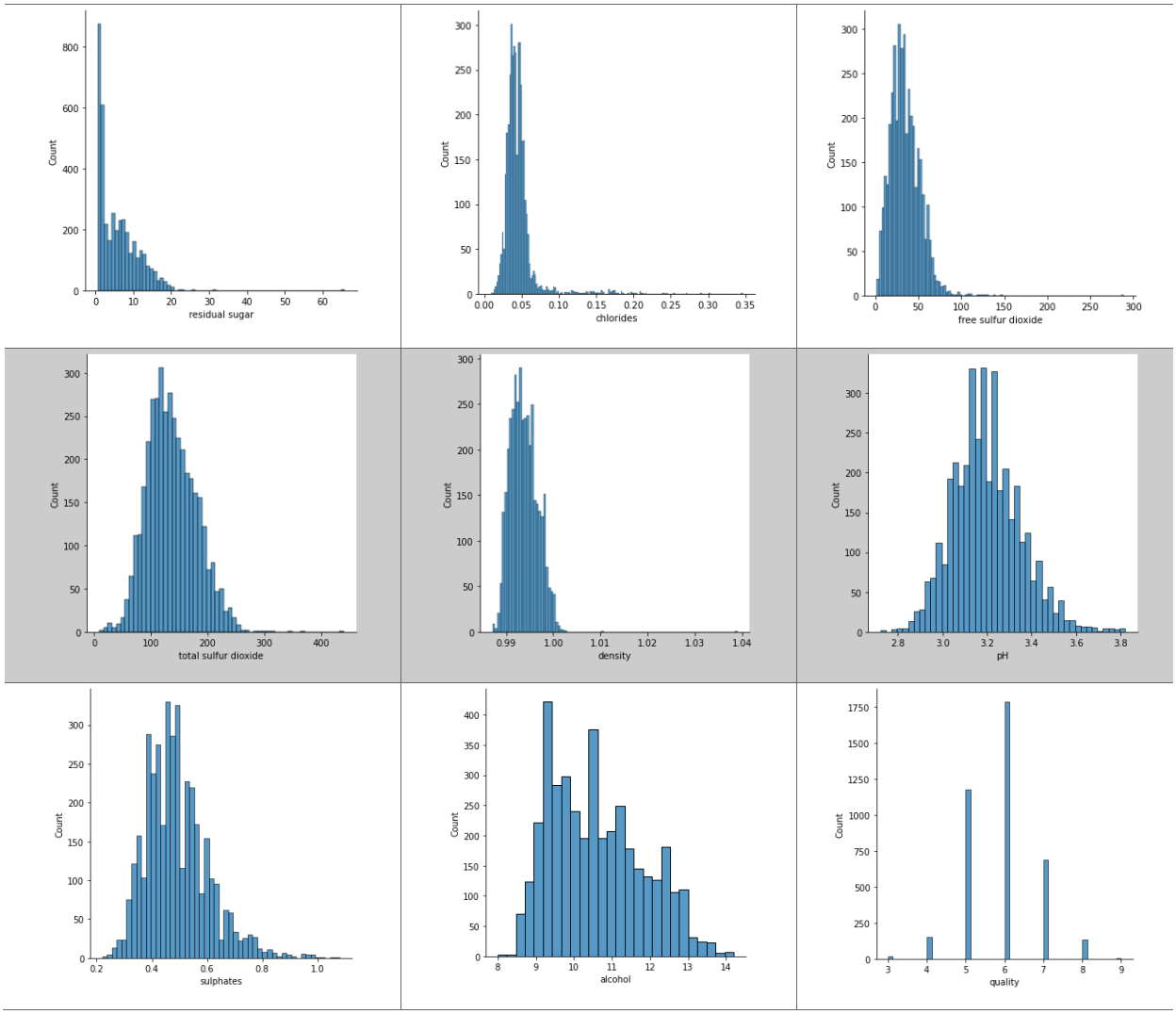





```
sns.displot(df_branco['fixed acidity'], kde=False)
sns.displot(df_branco['volatile acidity'], kde=False)
sns.displot(df_branco['citric acid'], kde=False)
sns.displot(df_branco['residual sugar'], kde=False)
sns.displot(df_branco['chlorides'], kde=False)
sns.displot(df_branco['free sulfur dioxide'], kde=False)
sns.displot(df_branco['total sulfur dioxide'], kde=False)
sns.displot(df_branco['density'], kde=False)
sns.displot(df_branco['pH'], kde=False)
sns.displot(df_branco['sulphates'], kde=False)
sns.displot(df_branco['alcohol'], kde=False)
sns.displot(df_branco['quality'], kde=False)
```

Vinhos brancos





Consultando literatura técnica, foi verificado que os valores, que a primeiro momento aparentam ser outliers, são valores aceitáveis para essas variáveis. Dessa forma, a decisão foi por manter os registros respectivos.

4.2 Correlação

Uma das finalidades de analisar as correlações existentes é detectar possíveis variáveis que poderiam ser agrupadas, dada a forte relação direta, tornando a modelagem mais simples.

Outro ganho enorme na análise de correlações é obter insights sobre as dependências e relações das classificações com relação às demais variáveis de entrada.

Para tal, mais uma vez utilizamos inicialmente uma avaliação tabular, o que pode ser desconfortável para a análise visual.

```
[21] df_tinto.corr()
```

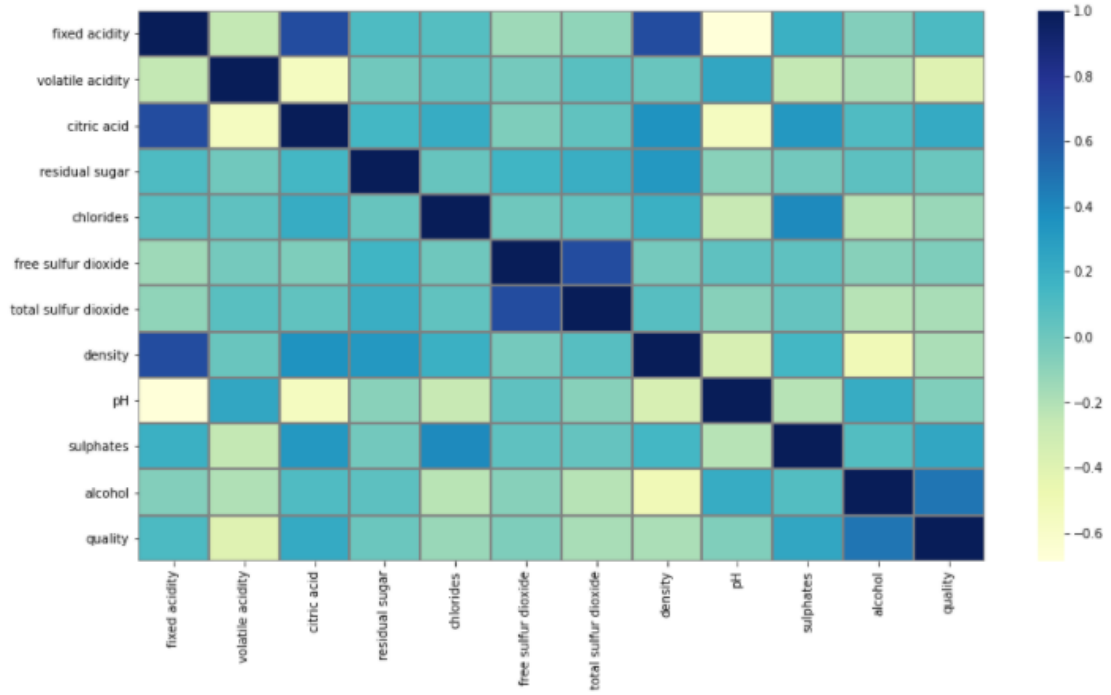
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.000000	-0.255124	0.667437	0.111025	0.085886	-0.140580	-0.103777	0.670195	-0.686685	0.190269	-0.061596	0.119024
volatile acidity	-0.255124	1.000000	-0.551248	-0.002449	0.055154	-0.020945	0.071701	0.023943	0.247111	-0.256948	-0.197812	-0.395214
citric acid	0.667437	-0.551248	1.000000	0.143892	0.210195	-0.048004	0.047358	0.357962	-0.550310	0.326062	0.105108	0.228057
residual sugar	0.111025	-0.002449	0.143892	1.000000	0.026656	0.160527	0.201038	0.324522	-0.083143	-0.011837	0.063281	0.013640
chlorides	0.085886	0.055154	0.210195	0.026656	1.000000	0.000749	0.045773	0.193592	-0.270893	0.394557	-0.223824	-0.130988
free sulfur dioxide	-0.140580	-0.020945	-0.048004	0.160527	0.000749	1.000000	0.667246	-0.018071	0.056631	0.054126	-0.080125	-0.050463
total sulfur dioxide	-0.103777	0.071701	0.047358	0.201038	0.045773	0.667246	1.000000	0.078141	-0.079257	0.035291	-0.217829	-0.177855
density	0.670195	0.023943	0.357962	0.324522	0.193592	-0.018071	0.078141	1.000000	-0.355617	0.146036	-0.504995	-0.184252
pH	-0.686685	0.247111	-0.550310	-0.083143	-0.270893	0.056631	-0.079257	-0.355617	1.000000	-0.214134	0.213418	-0.055245
sulphates	0.190269	-0.256948	0.326062	-0.011837	0.394557	0.054126	0.035291	0.146036	-0.214134	1.000000	0.091621	0.248835
alcohol	-0.061596	-0.197812	0.105108	0.063281	-0.223824	-0.080125	-0.217829	-0.504995	0.213418	0.091621	1.000000	0.480343
quality	0.119024	-0.395214	0.228057	0.013640	-0.130988	-0.050463	-0.177855	-0.184252	-0.055245	0.248835	0.480343	1.000000

```
[22] df_branco.corr()
```

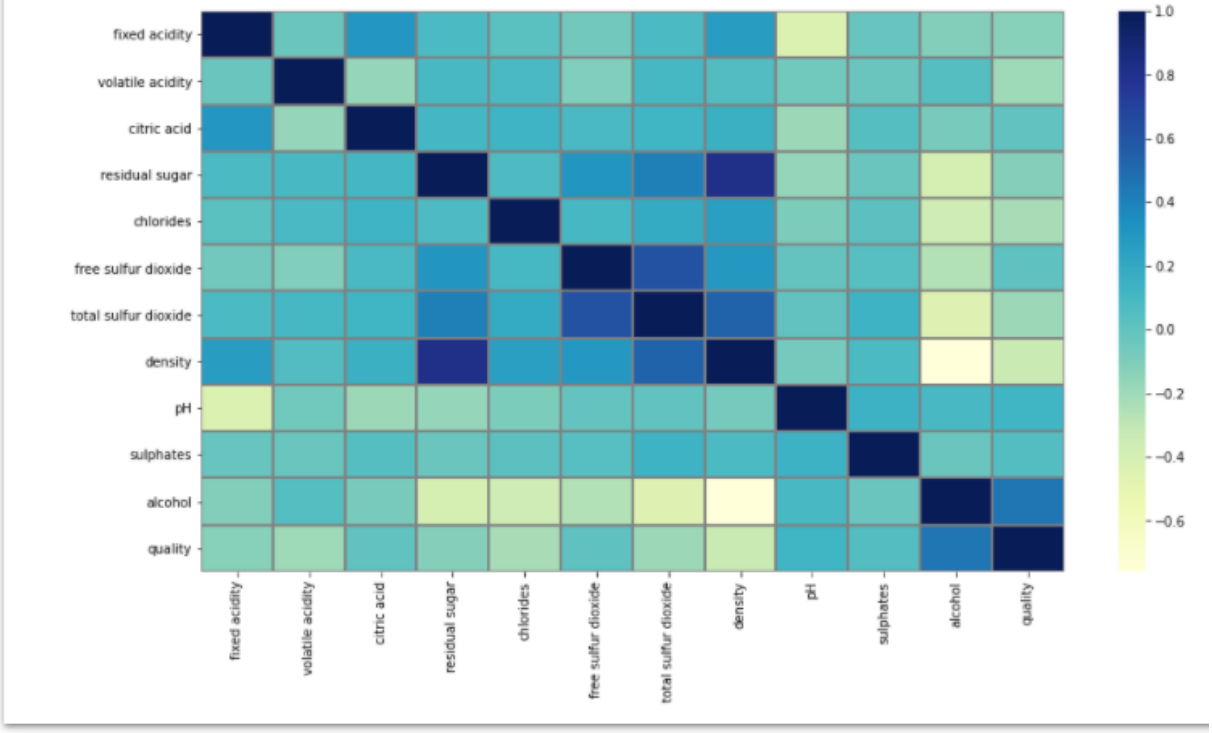
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.000000	-0.019214	0.298959	0.083620	0.024036	-0.058396	0.082425	0.266091	-0.431274	-0.017453	-0.110788	-0.124636
volatile acidity	-0.019214	1.000000	-0.163228	0.098340	0.086287	-0.102471	0.102315	0.060603	-0.046954	-0.021150	0.046815	-0.190678
citric acid	0.298959	-0.163228	1.000000	0.106269	0.132590	0.091681	0.122845	0.160076	-0.183015	0.049442	-0.076514	0.007065
residual sugar	0.083620	0.098340	0.106269	1.000000	0.076091	0.306835	0.409583	0.820498	-0.165997	-0.020503	-0.398167	-0.117339
chlorides	0.024036	0.086287	0.132590	0.076091	1.000000	0.101272	0.191145	0.253088	-0.090573	0.017871	-0.356928	-0.217739
free sulfur dioxide	-0.058396	-0.102471	0.091681	0.306835	0.101272	1.000000	0.619437	0.294638	-0.007750	0.037932	-0.251768	0.010507
total sulfur dioxide	0.082425	0.102315	0.122845	0.409583	0.191145	0.619437	1.000000	0.536868	0.008239	0.136544	-0.446643	-0.183356
density	0.266091	0.060603	0.160076	0.820498	0.253088	0.294638	0.536868	1.000000	-0.063734	0.082048	-0.760162	-0.337805
pH	-0.431274	-0.046954	-0.183015	-0.165997	-0.090573	-0.007750	0.008239	-0.063734	1.000000	0.142353	0.093095	0.123829
sulphates	-0.017453	-0.021150	0.049442	-0.020503	0.017871	0.037932	0.136544	0.082048	0.142353	1.000000	-0.022850	0.053200
alcohol	-0.110788	0.046815	-0.076514	-0.398167	-0.356928	-0.251768	-0.446643	-0.760162	0.093095	-0.022850	1.000000	0.462869
quality	-0.124636	-0.190678	0.007065	-0.117339	-0.217739	0.010507	-0.183356	-0.337805	0.123829	0.053200	0.462869	1.000000

Por isso, optamos por fazer a mesma análise de correlação usando mapas de calor, o que ajuda sobremaneira, dando velocidade à análise, facilitando a obtenção de insights possíveis e esperados.

```
[27] plt.subplots(figsize=(15, 8))
     sns.heatmap(df_tinto.corr(), cmap='YlGnBu', linecolor='gray', linewidths=1, annot=False);
```



```
plt.subplots(figsize=(15, 8))
sns.heatmap(df_branco.corr(), cmap='YlGnBu', linecolor='gray', linewidths=1, annot=False);
```



Avaliamos as correlações mais fortes como no caso dos tintos, entre *fixed acidity* e *citric acid* ou ainda entre *fixed acidity* e *density*.

No entanto, consultando material técnico sobre os estágios de vinificação, entendemos que esses indicadores podem ser convergentes em determinada fase do processo, mas podem ter sua correlação diminuída em outros momentos.

Dessa forma, visando não descartar dados que podem ser uteis em detrimento da performance na execução do modelo, que poderia ser melhorada com a unificação de variáveis, a opção foi por manter os *datasets* sem retirar variáveis.

4.3 Unificação de *datasets*

Temos dois conjuntos de dados, um relativo a valores obtidos para a categoria de vinho tinto e outro para vinho branco.

Em uma análise inicial, as correlações, distribuições e tendências são similares para ambos os conjuntos de dados de maneira que a opção foi por analisar os dois conjuntos em paralelo, e se possível e necessário, analisar as similaridades e diferenças.

Caso o resultado obtido não seja aceitável ou inconsistente, optaremos por revisar esta decisão de separar os *datasets*, fazendo a unificação dos mesmos, adicionando uma *feature* categórica para identificar o tipo de vinho.

4.4 Análises complementares

Com a utilização da biblioteca Pandas Profile foi possível ainda verificar a correlação sob outros parâmetros, não somente usando a correlação de Pearson.

Também graças a esta biblioteca e os demonstrativos gerados, foi possível analisar as dispersões ao se comparar duas variáveis de forma interativa gráfica e visual, o que foi feito buscando encontrar mais insights neste momento inicial de exploração.

Até o momento, além das conclusões relatadas em cada subitem deste tópico, ainda foi sugerida (insight a ser verificado) a possiblidade de não repetibilidade de padrões entre os *datasets* de vinhos tinto e branco.

Dada a interativade no resultado entregie pela biblioteca Pandas Profile, a referência foi restrita à citação, pois seria necessário um extenso registro de imagens. No entanto no repositório, o resultado está disponível com sua interatividade.

```
[ ] ! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
```

```
[ ] from pandas_profiling import ProfileReport
    profile_tinto = ProfileReport(df_tinto, title="Análise do dataset de vinhos tintos", html= {'style': {'full_width': True}})
    profile_tinto.to_notebook_iframe()
```

```
[ ] profile_branco = ProfileReport(df_tinto, title="Análise do dataset de vinhos brancos", html= {'style': {'full_width': True}})
    profile_branco.to_notebook_iframe()
```

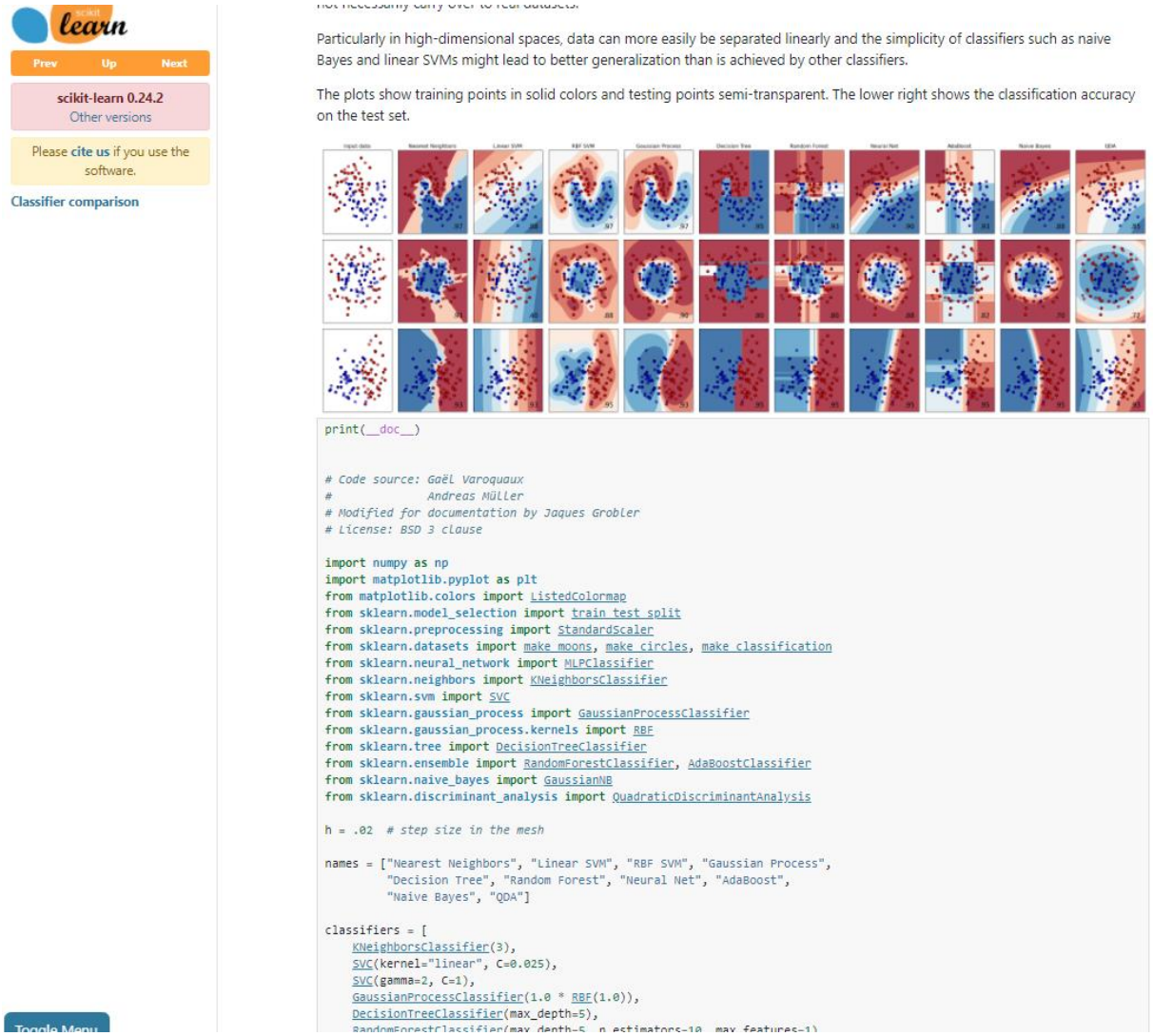
5. Criação de Modelos de Machine Learning

Para a criação do modelo de Machine Learning propriamente dito, foram levadas em consideração algumas premissas, na escolha dos algoritmos:

- Utilizar algoritmos que posibilitem a classificação, e não a regressão, pois queremos entender qual é a variável de destino, e não o quanto (apesar de as classes de resultado serem represantados como números, entendemos que os números são rótulos das classes).

- A caracteristica de todas as variáveis de entrada serem quantitativas contínuas;
- Utilizar a biblioteca Scikit Learn do Python, amplamente usada e altamente estável e confiável.

Dentre os diversos algoritmos disponíveis, como ilustrado abaixo do site (https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html) , foram escolhidos dois, os quais serão utilizados abaixo com a avaliação de seus resultados.



Como citado anteriormente, as análises serão feitas nos dois datasets (vinhos tintos e brancos) em paralelo, possibilitando uma comparação de resultados.

5.1 Decision Tree Classifier

A variável de resultado, ‘quality’, é isolada, e submetida ao algoritmo, como se segue.

```
[27] import pandas as pd
     from sklearn.tree import DecisionTreeClassifier
     df_tinto.head()
     x = df_tinto.drop('quality', axis=1)
     y = df_tinto['quality']
     modelo = DecisionTreeClassifier(max_depth=3)
     modelo.fit(x,y)
     modelo.score(x,y)
```

```
0.5805739514348786
```

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
df_branco.head()
x = df_branco.drop('quality', axis=1)
y = df_branco['quality']
modelo = DecisionTreeClassifier(max_depth=3)
modelo.fit(x,y)
modelo.score(x,y)
```

```
0.5415299166877051
```

Percebemos aqui a baixa acurácia obtida utilizando uma profundidade máxima de [3] no modelo.

A profundidade da árvore foi aumentada para 10, e assim a acurácia melhorou.

```
[33] import pandas as pd
     from sklearn.tree import DecisionTreeClassifier
     df_tinto.head()
     x = df_tinto.drop('quality', axis=1)
     y = df_tinto['quality']
     modelo = DecisionTreeClassifier(max_depth=10)
     modelo.fit(x,y)
     modelo.score(x,y)
```

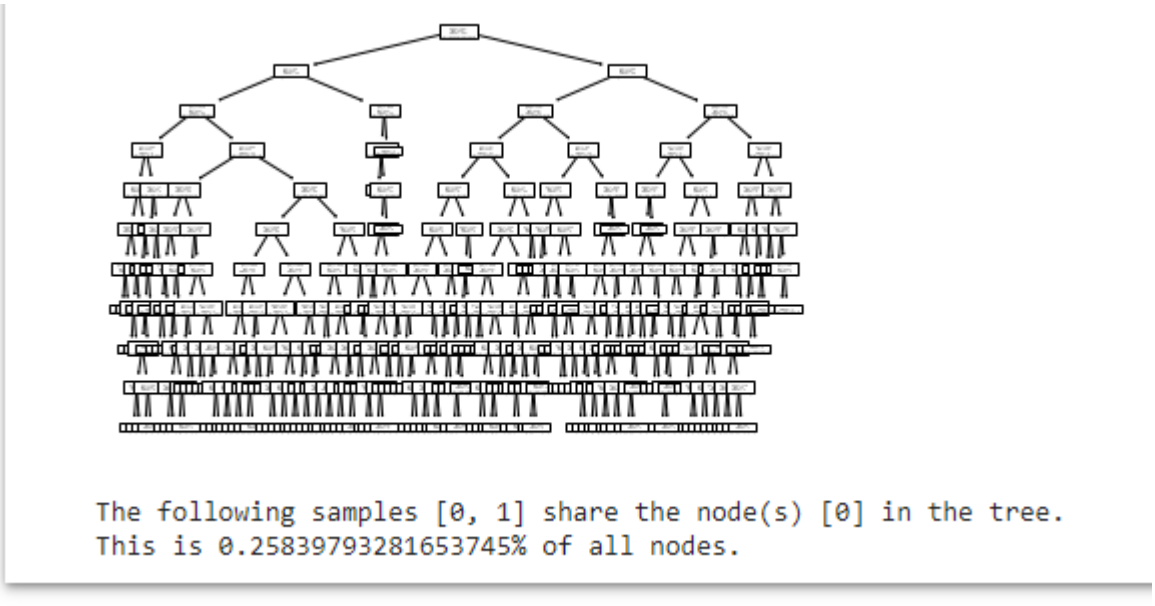
```
0.8623988226637234
```

```
[34] import pandas as pd
     from sklearn.tree import DecisionTreeClassifier
     df_branco.head()
     x = df_branco.drop('quality', axis=1)
     y = df_branco['quality']
     modelo = DecisionTreeClassifier(max_depth=10)
     modelo.fit(x,y)
     modelo.score(x,y)
```

```
0.7147185054279223
```

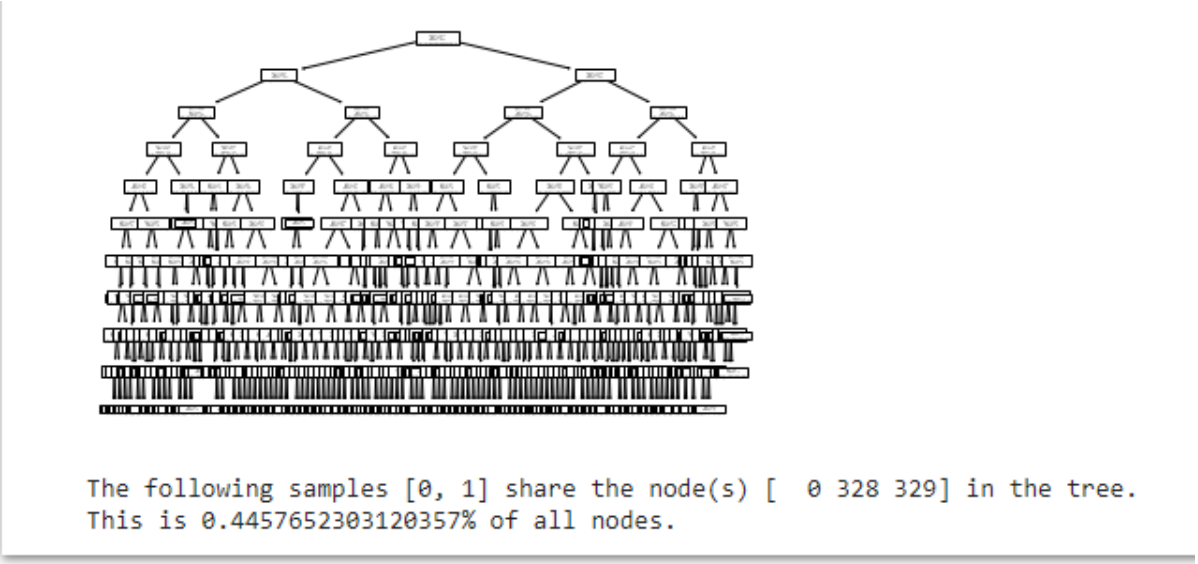
Utilizando ferramentas de plotagem da biblioteca Matplotlib, conseguimos obter um output de como o algoritmo é executado, e ao final obtemos uma representação gráfica.

Data a exensão do resultado, o que tornaria a apresentação exaustiva, foi incorparada ao presente apenas a imagem gráfica final. No entanto todo o resultado está disponível no repositório do Github para este projeto, na seção de scripts.



```
The following samples [0, 1] share the node(s) [0] in the tree.
This is 0.25839793281653745% of all nodes.
```

Para os vinhos tintos foram gerados 387 nós.



```
The following samples [0, 1] share the node(s) [  0 328 329] in the tree.
This is 0.4457652303120357% of all nodes.
```

Para os vinhos tintos foram gerados 673 nós.

Apesar de ter melhorado o resultado da acurácia, percebemos pelas imagens a alta complexidade para o alcance do resultado.

5.2 Linear Regression

Ainda utilizando o Scikit Learn, para possibilitar o uso da Regressão Linear, as bases serão divididas em treinamento e teste, procuraremos obter indicadores que nos apresente o quanto se prediz, usando os datasets.

Lembrando, foram executados os scripts para ambos os datasets.

```
import numpy as np
b = np.log(df_tinto.quality)
a = df_tinto.drop('quality', axis=1)

from sklearn.model_selection import train_test_split
a_treino, a_teste, b_treino, b_teste = train_test_split(a, b, random_state=42, test_size=.33)

from sklearn import linear_model
rl = linear_model.LinearRegression()

modelo = rl.fit(a_treino, b_treino)

print(f"R2: {modelo.score(a_teste, b_teste)}")
predictor = modelo.predict(a_teste)

from sklearn.metrics import mean_squared_error
print(f"RMSE: {mean_squared_error(b_teste, predictor)}")
```

```
R2: 0.3357768929832625
RMSE: 0.014835327725660053
```

```
import numpy as np
b = np.log(df_branco.quality)
a = df_branco.drop('quality', axis=1)

from sklearn.model_selection import train_test_split
a_treino, a_teste, b_treino, b_teste = train_test_split(a, b, random_state=42, test_size=.33)

from sklearn import linear_model
rl = linear_model.LinearRegression()

modelo = rl.fit(a_treino, b_treino)

print(f"R2: {modelo.score(a_teste, b_teste)}")
predictor = modelo.predict(a_teste)

from sklearn.metrics import mean_squared_error
print(f"RMSE: {mean_squared_error(b_teste, predictor)}")
```

```
R2: 0.24881543776185933
RMSE: 0.019113574793888134
```

Assim como foi feito no algritmo anterior, houve um ajuste em parametros para a obtenção de melhores resultados de classificação.

Abaixo o resultado otimizado, após a modificação no random_state.

```
[62] import numpy as np
     b = np.log(df_tinto.quality)
     a = df_tinto.drop('quality', axis=1)

     from sklearn.model_selection import train_test_split
     a_treino, a_teste, b_treino, b_teste = train_test_split(a, b, random_state=45, test_size=.33)

     from sklearn import linear_model
     rl = linear_model.LinearRegression()

     modelo = rl.fit(a_treino, b_treino)

     print(f"R2: {modelo.score(a_teste, b_teste)}")
     predictor = modelo.predict(a_teste)

     from sklearn.metrics import mean_squared_error
     print(f"RMSE: {mean_squared_error(b_teste, predictor)}")
```

```
R2: 0.3484386340605171
RMSE: 0.015701813303946262
```

```
import numpy as np
b = np.log(df_branco.quality)
a = df_branco.drop('quality', axis=1)

from sklearn.model_selection import train_test_split
a_treino, a_teste, b_treino, b_teste = train_test_split(a, b, random_state=10, test_size=.33)

from sklearn import linear_model
rl = linear_model.LinearRegression()

modelo = rl.fit(a_treino, b_treino)

print(f"R2: {modelo.score(a_teste, b_teste)}")
predictor = modelo.predict(a_teste)

from sklearn.metrics import mean_squared_error
print(f"RMSE: {mean_squared_error(b_teste, predictor)}")
```

```
R2: 0.30655882438162874
RMSE: 0.018239907250757678
```

Podemos inferir que com o baixo índice de R2 (entre 0,30 e 0,34), este modelo não é muito explicativo para as amostras.

6. Apresentação dos Resultados

De forma sintética, a partir do uso de dois algoritmos de Machine Learning (a saber, *Decision Tree Classifier* e *Linear Regression*), a fim de alcançar um bom resultado de classificação em um modelo computacional, conseguimos chegar a algumas conclusões e resultados que relataremos a seguir.

Foi utilizada a metodologia padrão descrita no presente trabalho, considerando as etapas de:

- Coleta de dados, descrita no Item 2;
- Processamento e tratamento de dados, descritos no Item 3;
- Análise e exploração dos dados, descritos no Item 4;
- Criação de modelos de *Machine Learning* propriamente dita no Item 5.

A escolha dos algoritmos foi feita por conta da facilidade didática de uso dos mesmos, e no caso específico da Árvore de Decisão (*Decision Tree Classifier*), a intenção foi obter uma visão inteligível para o usuário, o qual conseguiria navegar pelos ramos da árvore e assim entender como se deu a classificação. Mais adiante veremos que dada a grande extensão das árvores montadas, essa possibilidade torna-se inexequível, pois para obter um resultado próximo de satisfatório de classificação são necessários muitos níveis e ramificações.

Desde o início optamos por executar todas as atividades nos dois *datasets* em separado, apesar dos mesmos terem estrutura de dados iguais, o que nos possibilitaria a união em único arquivo e a adição de uma coluna contendo o tipo de vinho, se tinto ou branco. Seria um parâmetro de entrada a mais. A opção por analisar as duas bases em separado foi para que pudéssemos entender no primeiro momento se os resultados eram muito destoantes, ou seja, se havia indícios de os critérios de classificação serem muito diferentes.

Chegamos à conclusão de que a mecânica de classificação é muito próxima, dada a similaridade de resultados encontrados a partir da variação de hiper parâmetros na execução dos algoritmos (exceto no caso do vinho branco na Regressão Linear, como veremos mais à frente).

Convêm frisar que nos passos de tratamento e análise de dados, foram tomadas decisões conservadoras, ou seja, fazendo o mínimo de intervenções nas bases de dados originais.

Partindo do princípio de que uma boa análise e tratamento de dados requer conhecimentos de metodologias computacionais (ETL), mas também é grandemente beneficiada pelo conhecimento do assunto tratado, no caso, o conhecimento da relevância das variáveis físico-químicas na qualificação dos vinhos das duas categorias, tintos e brancos de uma região específica do mundo. Esse

conservadorismo levou à decisão de fazer o mínimo de intervenções, visando não descartar dados relevantes como talvez um outlier que tenha um grande significado, ou uma distribuição precisando de ajuste, mas que na verdade é assim mesmo que o dado deve se comportar.

Quando da execução dos algoritmos, partimos de valores quaisquer para os hiper parâmetros e fomos ajustando até obter melhores indicadores de qualidade na execução do algoritmo, como se segue nas tabelas abaixo:

Algoritmo	Dataset	Parâmetro	Valor de Entrada	Medida de Ganho	Valor Obtido
Decision Tree	Tinto	max_depth	3	score	0,581
		max_depth	10	score	0,862
	Branco	max_depth	3	score	0,542
		max_depth	10	score	0,715

Com o aumento da profundidade tendemos ao *overfitting* (não desejável), buscando uma maior generalização do modelo. Isso foi obtido como demonstrado pelo aumento do score obtido, em ambos os casos com um aumento de 48% e 32% (tintos e brancos, respectivamente) no valor do mesmo.

Perdemos, no entanto, a vantagem da fácil visualização do critério de classificação (ficou complexo), com a geração de centenas de ramificações dentro do limite de 10 níveis de profundidade.

Para a Regressão Linear obtivemos:

Algoritmo	Dataset	Parâmetro	Valor de Entrada	Medida de Ganho	Valor Obtido
Linear Regression	Tinto	random_state	42	R2	0,336
			42	RMSE	0,015
		random_state	45	R2	0,348
			45	RMSE	0,016

Algoritmo	Dataset	Parâmetro	Valor de Entrada	Medida de Ganho	Valor Obtido
Linear Regression	Branco	random_state	42	R2	0,249
			42	RMSE	0,019
		random_state	10	R2	0,307
			10	RMSE	0,018

Considerando:

- R2 (Coeficiente de determinação) alto indica um bom ajuste aos dados, indicando que o modelo linear explica bem a amostra;
- RMSE (*Root mean square error*, ou Erro médio quadrático), como agregador de resíduos (medida de desvio padrão dos resíduos), quanto menor nos indica um melhor ajuste;

Após experimentar diversos valores para o *random_state*, não obtivemos grande variação nas variáveis de saída, mas atentamos para o fato de que o RMSE foi um indicativo favorável para a escolha deste algoritmo, se mantendo baixo em todos os experimentos. No entanto, o valor de R2 não foi satisfatório como indicador de aplicabilidade do modelo. Variou pouco.

Chegamos à conclusão de que o algoritmo de *Decision Tree Classifier* obteve um melhor desempenho, apesar de ainda não prover um índice satisfatório de assertividade de classificação. Além disso, o que imaginávamos ser uma vantagem a ser usufruída o fato de ser mais inteligível, considerando a profundidade necessária de níveis para alcançar este resultado, este também não foi um fator que redimisse este algoritmo.

Por fim, após ter acesso a outros trabalhos similares, inclusive no Brasil (SANTOS, 2006), entendemos que há espaço para uma maior pesquisa utilizando outros algoritmos em busca da descoberta do que mais se adeque à essa necessidade, considerando a necessidade de maior conhecimento do tema abordado.

Considerando a finalidade do presente trabalho ser a aplicabilidade de Ciência de Dados com uso de *Machine Learning*, entendemos que a profundidade de conhecimento do tema foi suficiente para demonstrar esta aplicabilidade.

7. Links

Abaixo estão disponibilizados os links para os dois entregáveis adicionais e complementares ao presente trabalho, a saber:

- Vídeo com apresentação:
  - https://youtu.be/LzxoItOl35A
- Repositório no GitHub contendo todos os artefatos usados no projeto, como *datasets*, scripts, arquivos Powerpoint e em formato *PDF*:
  - https://github.com/caesarporto/PUC_TCC_DataScience

REFERÊNCIAS

Dada a não obrigatoriedade da inclusão de referências bibliográficas por ser um projeto/relatório sem necessidade de revisão bibliográfica, essa sessão será dedicada apenas a citação de obras referentes ao objeto da análise que deram suporte à tomada de decisão em algumas fases do projeto.

Não foram adicionadas referências tecnológicas ou metodológicas, pois todo o trabalho foi realizado utilizando como base o arcabouço teórico / prático do curso.

BONTEMPO, Márcio. **A saúde da água para o vinho.** Brasília: Thesaurus, 2012.

LAGO-VANZELA, SILVA, BAFFI (org.). **Uvas e Vinhos, química, bioquímica e microbiologia.** São Paulo: Editora Unesp; Editora Senac, 2015.

MARGEON, Gérard. **Vocabulário básico do vinho.** São Paulo: Martins Fontes, 2015.

PUCKETTE e HAMMACK, Madeline e Justin. **O guia essencial do vinho: *wine folly*.** Rio de Janeiro: Intrínseca, 2016.

SANTOS, Betânia Araújo Cosme dos. **Compostos voláteis e qualidade dos vinhos secos jovens varietal cabernet sauvignon produzidos em diferentes regiões do Brasil.** Campinas, SP: [s.n.](Tese de Doutorado na UNICAMP em Ciência de Alimentos), 2006.

TAPIA, Patricio. **Guia descorchados.** São Paulo: Inner, 2020.

VALENCA, Alfeu. **Apostila de Vinhos.xlsx.** Rio de Janeiro: 2002.

APÊNDICE

Programação/Scripts

Abaixo são apresentados os scripts executados em cada seção do trabalho ora apresentado.

```
#Passo 3.1
import pandas as pd
df_tinto = pd.read_csv('https://raw.githubusercontent.com/caesarporto/PUC_TCC_DataScience/main/datasource/winequality_red.csv', sep=',')
df_branco = pd.read_csv('https://raw.githubusercontent.com/caesarporto/PUC_TCC_DataScience/main/datasource/winequality_white.csv', sep=',')

#Passo 3.2 (a)
df_tinto.shape

#Passo 3.2 (b)
df_branco.shape

#Passo 3.3 (a)
df_tinto.head()

#Passo 3.3 (b)
df_branco.head()

#Passo 3.3 (c)
df_tinto.sample(10)

#Passo 3.3 (d)
df_branco.sample(10)

#Passo 3.4 (a)
df_tinto.count()

#Passo 3.4 (b)
df_branco.count()

#Passo 3.4 (c)
df_tinto.drop_duplicates(inplace=True)
df_branco.drop_duplicates(inplace=True)

#Passo 3.4 (d)
df_tinto.count()
```

```
#Passo 3.5 (e)
df_branco.count()

#Passo 3.5 (a)
import missingno as msno
msno.bar(df_tinto)

#Passo 3.5 (b)
msno.bar(df_branco)

#Passo 4.1 (a)
df_tinto.describe()

#Passo 4.1 (b)
df_branco.describe()

#Passo 4.1 (c)
import matplotlib.pyplot as plt
import seaborn as sns
% matplotlib inline

#Passo 4.1 (d)
sns.displot(df_tinto['fixed acidity'], kde=False)
sns.displot(df_tinto['volatile acidity'], kde=False)
sns.displot(df_tinto['citric acid'], kde=False)
sns.displot(df_tinto['residual sugar'], kde=False)
sns.displot(df_tinto['chlorides'], kde=False)
sns.displot(df_tinto['free sulfur dioxide'], kde=False)
sns.displot(df_tinto['total sulfur dioxide'], kde=False)
sns.displot(df_tinto['density'], kde=False)
sns.displot(df_tinto['pH'], kde=False)
sns.displot(df_tinto['sulphates'], kde=False)
sns.displot(df_tinto['alcohol'], kde=False)
sns.displot(df_tinto['quality'], kde=False)

#Passo 4.1 (e)
sns.displot(df_branco['fixed acidity'], kde=False)
sns.displot(df_branco['volatile acidity'], kde=False)
sns.displot(df_branco['citric acid'], kde=False)
sns.displot(df_branco['residual sugar'], kde=False)
sns.displot(df_branco['chlorides'], kde=False)
sns.displot(df_branco['free sulfur dioxide'], kde=False)
sns.displot(df_branco['total sulfur dioxide'], kde=False)
sns.displot(df_branco['density'], kde=False)
sns.displot(df_branco['pH'], kde=False)
sns.displot(df_branco['sulphates'], kde=False)
sns.displot(df_branco['alcohol'], kde=False)
```

```
sns.displot(df_branco['quality'], kde=False)

#Passo 4.2 (a)
df_tinto.corr()

#Passo 4.2 (b)
df_branco.corr()

#Passo 4.2 (c)
plt.subplots(figsize=(15, 8))
sns.heatmap(df_tinto.corr(), cmap='YlGnBu', linecolor='gray', linewidths=1, annot=F
alse);

#Passo 4.2 (d)
plt.subplots(figsize=(15, 8))
sns.heatmap(df_branco.corr(), cmap='YlGnBu', linecolor='gray', linewidths=1, annot=
False);

#Passo 4.4 (a)
! pip install https://github.com/pandas-profiling/pandas-
profiling/archive/master.zip

#Passo 4.4 (b)
from pandas_profiling import ProfileReport
profi-
le_tinto = ProfileReport(df_tinto, title="Análise do dataset de vinhos tintos", htm
l= {'style': {'full_width': True}})
profile_tinto.to_notebook_iframe()

#Passo 4.4 (c)
profi-
le_branco = ProfileReport(df_tinto, title="Análise do dataset de vinhos brancos", h
tml= {'style': {'full_width': True}})
profile_branco.to_notebook_iframe()

#Passo 5.1 (a)
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
df_tinto.head()
x = df_tinto.drop('quality', axis=1)
y = df_tinto['quality']
modelo = DecisionTreeClassifier(max_depth=3)
modelo.fit(x,y)
modelo.score(x,y)

#Passo 5.1 (b)
import pandas as pd
```

```
from sklearn.tree import DecisionTreeClassifier
df_branco.head()
x = df_branco.drop('quality', axis=1)
y = df_branco['quality']
modelo = DecisionTreeClassifier(max_depth=3)
modelo.fit(x,y)
modelo.score(x,y)

#Passo 5.1 (c)
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
df_tinto.head()
x = df_tinto.drop('quality', axis=1)
y = df_tinto['quality']
modelo = DecisionTreeClassifier(max_depth=10)
modelo.fit(x,y)
modelo.score(x,y)

#Passo 5.1 (d)
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
df_branco.head()
x = df_branco.drop('quality', axis=1)
y = df_branco['quality']
modelo = DecisionTreeClassifier(max_depth=10)
modelo.fit(x,y)
modelo.score(x,y)

#Passo 5.1 (e)
import numpy as np
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

X = df_tinto.drop('quality', axis=1)
y = df_tinto['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = DecisionTreeClassifier(max_depth=10)
clf.fit(X_train, y_train)

n_nodes = clf.tree_.node_count
children_left = clf.tree_.children_left
children_right = clf.tree_.children_right
feature = clf.tree_.feature
threshold = clf.tree_.threshold
```

```
node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, 0)]
while len(stack) > 0:
    node_id, depth = stack.pop()
    node_depth[node_id] = depth

    is_split_node = children_left[node_id] != children_right[node_id]

    if is_split_node:
        stack.append((children_left[node_id], depth + 1))
        stack.append((children_right[node_id], depth + 1))
    else:
        is_leaves[node_id] = True

print("The binary tree structure has {n} nodes and has "
      "the following tree structure:\n".format(n=n_nodes))
for i in range(n_nodes):
    if is_leaves[i]:
        print("{space}node={node} is a leaf node.".format(
            space=node_depth[i] * "\t", node=i))
    else:
        print("{space}node={node} is a split node: "
              "go to node {left} if X[:, {feature}] <= {threshold} "
              "else to node {right}.".format(
                  space=node_depth[i] * "\t",
                  node=i,
                  left=children_left[i],
                  feature=feature[i],
                  threshold=threshold[i],
                  right=children_right[i]))

tree.plot_tree(clf)
plt.show()

node_indicator = clf.decision_path(X_test)
leaf_id = clf.apply(X_test)

sample_id = 0
node_index = node_indicator.indices[node_indicator.indptr[sample_id]:
                                    node_indicator.indptr[sample_id + 1]]

sample_ids = [0, 1]
common_nodes = (node_indicator.toarray()[sample_ids].sum(axis=0) ==
                len(sample_ids))
common_node_id = np.arange(n_nodes)[common_nodes]

print("\nThe following samples {samples} share the node(s) {nodes} in the "
```

```
      "tree.".format(samples=sample_ids, nodes=common_node_id))
print("This is {prop}% of all nodes.".format(
    prop=100 * len(common_node_id) / n_nodes))

#Passo 5.1 (f)
import numpy as np
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

X = df_branco.drop('quality', axis=1)
y = df_branco['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = DecisionTreeClassifier(max_depth=10)
clf.fit(X_train, y_train)

n_nodes = clf.tree_.node_count
children_left = clf.tree_.children_left
children_right = clf.tree_.children_right
feature = clf.tree_.feature
threshold = clf.tree_.threshold

node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, 0)]
while len(stack) > 0:
    node_id, depth = stack.pop()
    node_depth[node_id] = depth

    is_split_node = children_left[node_id] != children_right[node_id]

    if is_split_node:
        stack.append((children_left[node_id], depth + 1))
        stack.append((children_right[node_id], depth + 1))
    else:
        is_leaves[node_id] = True

print("The binary tree structure has {n} nodes and has "
      "the following tree structure:\n".format(n=n_nodes))
for i in range(n_nodes):
    if is_leaves[i]:
        print("{space}node={node} is a leaf node.".format(
            space=node_depth[i] * "\t", node=i))
    else:
        print("{space}node={node} is a split node: "
              "go to node {left} if X[:, {feature}] <= {threshold} "
```

```
                "else to node {right}.".format(
                    space=node_depth[i] * "\t",
                    node=i,
                    left=children_left[i],
                    feature=feature[i],
                    threshold=threshold[i],
                    right=children_right[i]))

tree.plot_tree(clf)
plt.show()

node_indicator = clf.decision_path(X_test)
leaf_id = clf.apply(X_test)

sample_id = 0
node_index = node_indicator.indices[node_indicator.indptr[sample_id]:
                                    node_indicator.indptr[sample_id + 1]]

sample_ids = [0, 1]
common_nodes = (node_indicator.toarray()[sample_ids].sum(axis=0) ==
                len(sample_ids))
common_node_id = np.arange(n_nodes)[common_nodes]

print("\nThe following samples {samples} share the node(s) {nodes} in the "
      "tree.".format(samples=sample_ids, nodes=common_node_id))
print("This is {prop}% of all nodes.".format(
    prop=100 * len(common_node_id) / n_nodes))

#Passo 5.2 (a)
import numpy as np
b = np.log(df_tinto.quality)
a = df_tinto.drop('quality', axis=1)

from sklearn.model_selection import train_test_split
a_treino, a_teste, b_treino, b_teste = train_test_split(a, b, random_state=42, test
_size=.33)

from sklearn import linear_model
rl = linear_model.LinearRegression()

modelo = rl.fit(a_treino, b_treino)

print(f"R2: {modelo.score(a_teste, b_teste)}")
predictor = modelo.predict(a_teste)

from sklearn.metrics import mean_squared_error
print(f"RMSE: {mean_squared_error(b_teste, predictor)}")
```

```
#Passo 5.2 (b)
import numpy as np
b = np.log(df_branco.quality)
a = df_branco.drop('quality', axis=1)

from sklearn.model_selection import train_test_split
a_treino, a_teste, b_treino, b_teste = train_test_split(a, b, random_state=42, test
_size=.33)

from sklearn import linear_model
rl = linear_model.LinearRegression()

modelo = rl.fit(a_treino, b_treino)

print(f"R2: {modelo.score(a_teste, b_teste)}")
predictor = modelo.predict(a_teste)

from sklearn.metrics import mean_squared_error
print(f"RMSE: {mean_squared_error(b_teste, predictor)}")

#Passo 5.2 (c)
import numpy as np
b = np.log(df_tinto.quality)
a = df_tinto.drop('quality', axis=1)

from sklearn.model_selection import train_test_split
a_treino, a_teste, b_treino, b_teste = train_test_split(a, b, random_state=45, test
_size=.33)

from sklearn import linear_model
rl = linear_model.LinearRegression()

modelo = rl.fit(a_treino, b_treino)

print(f"R2: {modelo.score(a_teste, b_teste)}")
predictor = modelo.predict(a_teste)

from sklearn.metrics import mean_squared_error
print(f"RMSE: {mean_squared_error(b_teste, predictor)}")

#Passo 5.2 (d)
import numpy as np
b = np.log(df_branco.quality)
a = df_branco.drop('quality', axis=1)

from sklearn.model_selection import train_test_split
a_treino, a_teste, b_treino, b_teste = train_test_split(a, b, random_state=10, test
_size=.33)
```

```
from sklearn import linear_model
rl = linear_model.LinearRegression()

modelo = rl.fit(a_treino, b_treino)

print(f"R2: {modelo.score(a_teste, b_teste)}")
predictor = modelo.predict(a_teste)

from sklearn.metrics import mean_squared_error
print(f"RMSE: {mean_squared_error(b_teste, predictor)}")
```