Projeto de Disciplina de Algoritmos de Classificação [25E1_2]

Aluna: Rachel Reuters

PARTE 1- MODULO KAGGLE: Kaggle Intro to Machine Learning

In [2]:
```
import pandas as pd
# Download latest version
melbourne_data    =  pd.read_csv('melb_data.csv', sep=',', decimal='.')
melbourne_data.describe()
```

Out[2]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000

In [3]:
```
melbourne_data = melbourne_data.dropna(axis=0)

y = melbourne_data.Price

melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
```

In [4]:
```
from sklearn.tree import DecisionTreeRegressor

X = melbourne_data[melbourne_features]


# Define model. Specify a number for random_state to ensure same results each run
melbourne_model = DecisionTreeRegressor(random_state=1)

# Fit model
melbourne_model.fit(X, y)

predict_tree = melbourne_model.predict(X)
```

In [5]:
```
X['RealPrice'] = y
X['Predict'] = predict_tree
```

```
c:\Users\belch\anaconda3\envs\tfd10\lib\site-packages\ipykernel_launcher.py:1: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
c:\Users\belch\anaconda3\envs\tfd10\lib\site-packages\ipykernel_launcher.py:2: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
```

In [6]:
```
X
```

Out[6]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude	RealPrice	Predict
1	2	1.0	156.0	-37.80790	144.99340	1035000.0	1035000.0
2	3	2.0	134.0	-37.80930	144.99440	1465000.0	1465000.0
4	4	1.0	120.0	-37.80720	144.99410	1600000.0	1600000.0
6	3	2.0	245.0	-37.80240	144.99930	1876000.0	1876000.0
7	2	1.0	256.0	-37.80600	144.99540	1636000.0	1636000.0
...	...	...	...	...	...	...	...
12205	3	2.0	972.0	-37.51232	145.13282	601000.0	601000.0
12206	3	1.0	179.0	-37.86558	144.90474	1050000.0	1050000.0
12207	1	1.0	0.0	-37.85588	144.89936	385000.0	385000.0
12209	2	1.0	0.0	-37.85581	144.99025	560000.0	560000.0
12212	6	3.0	1087.0	-37.81038	144.89389	2450000.0	2450000.0

6196 rows × 7 columns

In [7]:
```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(y, predict_tree)
```

Out[7]:
```
1115.7467183128902
```

In [8]:
```
from sklearn.model_selection import train_test_split
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 22)
melbourne_model = DecisionTreeRegressor()
```

```
melbourne_model.fit(train_X, train_y)

# get predicted prices on validation data
val_predictions = melbourne_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))
```

```
1856.286636539703
```

In [9]:
```
def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=22)
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)

# compare MAE with differing values of max_leaf_nodes
best_mae= 999999
for max_leaf_nodes in [5, 50, 500, 5000,20000]:
    current_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    if current_mae <best_mae :
        best_mae = current_mae
        best_leaf_node  = max_leaf_nodes

print(f"Best leaf = {best_leaf_node}")
model = DecisionTreeRegressor(max_leaf_nodes=best_leaf_node, random_state=22)
model.fit(X, y)
preds_val = model.predict(val_X)


mae_final = mean_absolute_error(val_y, preds_val)
print(mae_final)
```

```
Best leaf = 5000
0.0
```

In [10]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(random_state=22)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

```
1564.0011555842486
```

PARTE 2 - BASE DE DADOS

In [11]:
```python
import pandas as pd
import numpy as np
import warnings
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline


# Ignore all warnings
warnings.filterwarnings('ignore')

dataset_original = pd.read_csv('winequalityN.csv', sep=',', decimal='.')

dataset_filtrado = dataset_original[dataset_original['type'] == "white"]

dataset_filtrado['opinion'] = np.where(dataset_filtrado['quality'] > 5 , 1, 0)

dataset_filtrado.drop(columns='quality', inplace=True)

dataset_filtrado
```

Out[11]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	s
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	
...	...	...	...	...	...	...	...	...	...	...	...
4893	white	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	
4894	white	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	
4895	white	6.5	NaN	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	
4896	white	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	
4897	white	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	

4898 rows × 13 columns

PARTE 3 - PRE-PROCESSAMENTO E ANÁLISE DOS DADOS

Analisando as colunas que possuem linhas vazias,as colunas provavelmente tem dados vazios provavelmente por falta de informacao ou deixaram de anotar. Existem muitas possibilidades, porém para o problema proposto acredito que preencher com a média pode ser uma boa alternativa:

In [12]:
```
#Verificando dados NULOS
print(dataset_filtrado.isna().sum())
```

```
type                  0
fixed acidity         8
volatile acidity      7
citric acid           2
residual sugar        2
chlorides             2
free sulfur dioxide   0
total sulfur dioxide  0
density               0
pH                    7
sulphates             2
alcohol               0
opinion               0
dtype: int64
```

In [13]:
```
dataset_filtrado[dataset_filtrado.columns] = dataset_filtrado[dataset_filtrado.colu
```

Verificando se existem dados duplicados:

In [14]:
```
dataset_filtrado.duplicated().sum()
```

Out[14]:
```
928
```

In [15]:
```
dataset_vinho_branco_tratado = dataset_filtrado.drop_duplicates()
dataset_vinho_branco_tratado
```

Out[15]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	white	7.0	0.270000	0.36	20.7	0.045	45.0	170.0	1.00100	3.00
1	white	6.3	0.300000	0.34	1.6	0.049	14.0	132.0	0.99400	3.30
2	white	8.1	0.280000	0.40	6.9	0.050	30.0	97.0	0.99510	3.26
3	white	7.2	0.230000	0.32	8.5	0.058	47.0	186.0	0.99560	3.19
6	white	6.2	0.320000	0.16	7.0	0.045	30.0	136.0	0.99490	3.18
...	...	...	...	...	...	...	...	...	...	...
4893	white	6.2	0.210000	0.29	1.6	0.039	24.0	92.0	0.99114	3.27
4894	white	6.6	0.320000	0.36	8.0	0.047	57.0	168.0	0.99490	3.15
4895	white	6.5	0.278252	0.19	1.2	0.041	30.0	111.0	0.99254	2.99
4896	white	5.5	0.290000	0.30	1.1	0.022	20.0	110.0	0.98869	3.34
4897	white	6.0	0.210000	0.38	0.8	0.020	22.0	98.0	0.98941	3.26

3970 rows × 13 columns

Analisando os tipos das colunas da base de dados:

In [16]:
```
variable = dataset_vinho_branco_tratado.columns
data_input_analysis = pd.DataFrame(dataset_vinho_branco_tratado.dtypes)
data_input_analysis = data_input_analysis.rename(columns={0: 'Type'})

conditions = [
    (data_input_analysis['Type'] == "object"),
    (data_input_analysis['Type'] == "float64"),
    (data_input_analysis['Type'] == "int64") | (data_input_analysis['Type'] == "int
]

data_input_analysis["Type"] = np.select(conditions, ["categorical", "continuous", "

data_input_analysis["Average"] = dataset_vinho_branco_tratado[data_input_analysis.i

data_input_analysis["StandardDeviation"] = dataset_vinho_branco_tratado[data_input_

data_input_analysis["MissingValues"] = dataset_vinho_branco_tratado[data_input_anal
```

```
data_input_analysis
```

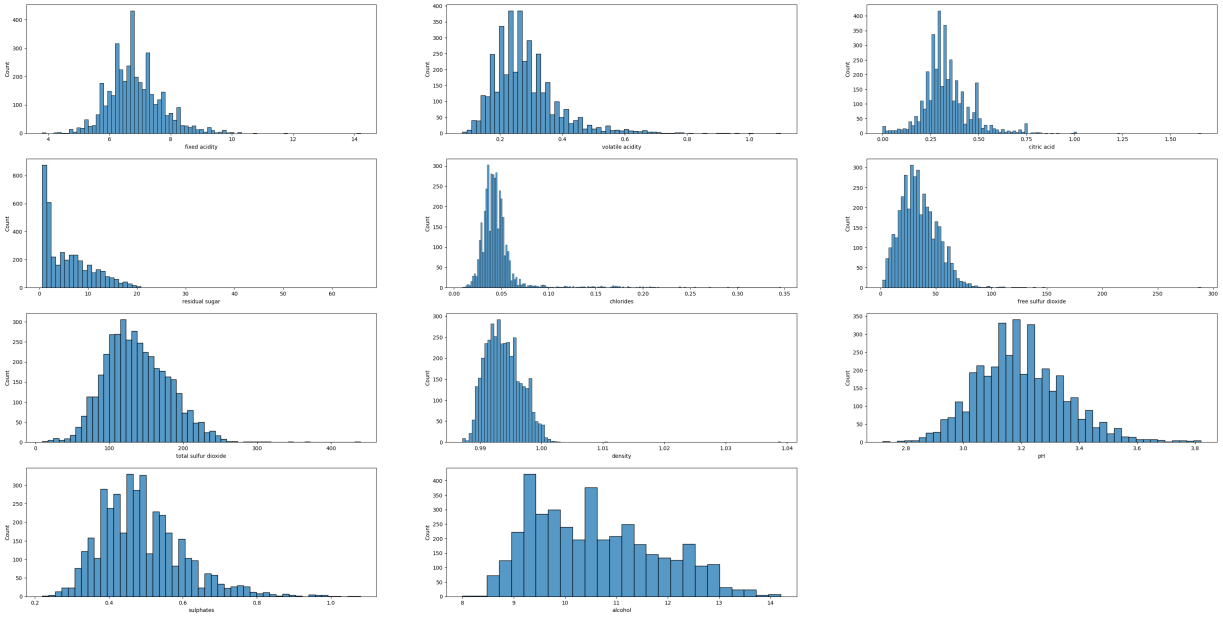
Out[16]:

	Type	Average	StandardDeviation	MissingValues
type	categorical	NaN	NaN	False
fixed acidity	continuous	6.840905	0.865528	False
volatile acidity	continuous	0.280637	0.103486	False
citric acid	continuous	0.334551	0.122449	False
residual sugar	continuous	5.920727	4.863427	False
chlorides	continuous	0.045895	0.023079	False
free sulfur dioxide	continuous	34.909698	17.218706	False
total sulfur dioxide	continuous	137.248992	43.133975	False
density	continuous	0.993792	0.002905	False
pH	continuous	3.195297	0.151345	False
sulphates	continuous	0.490398	0.113566	False
alcohol	continuous	10.588324	1.217302	False
opinion	discrete	0.659698	0.473870	False

Selecionando as colunas continuas para serem as features pois a unica categorica eh de tipo de vinho e so estamos analisando inicialmenteo vinho branco. Pelas distribuicoes, os valores estao fora da mesma escala e posteriormente vai ser necessario aplicar uma padronizacao dos dados.

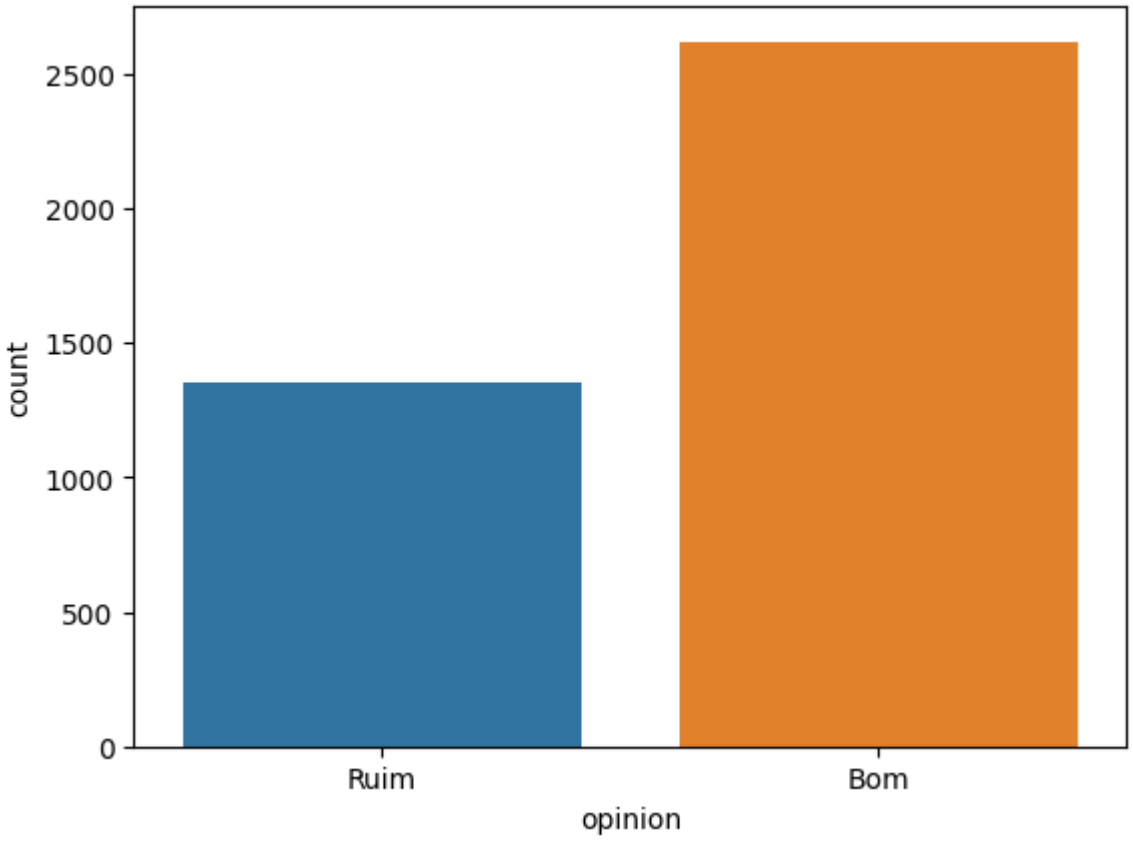
In [17]:
```
numerical_columns = data_input_analysis.loc[data_input_analysis['Type'] == "continu
print(numerical_columns)


#Verificando distribuicao dos dados
plt.figure(figsize=(40, 20))

for index, value in enumerate(numerical_columns):
    plt.subplot(4, 3,index+1)
    sns.histplot(dataset_vinho_branco_tratado[value])
    plt.xlabel(numerical_columns[index])
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol'],
      dtype='object')
```



Verificando balanceamento dos dados, podemos ver que trata-se de uma base desbalanceada, com maior numero de bons vinhos.

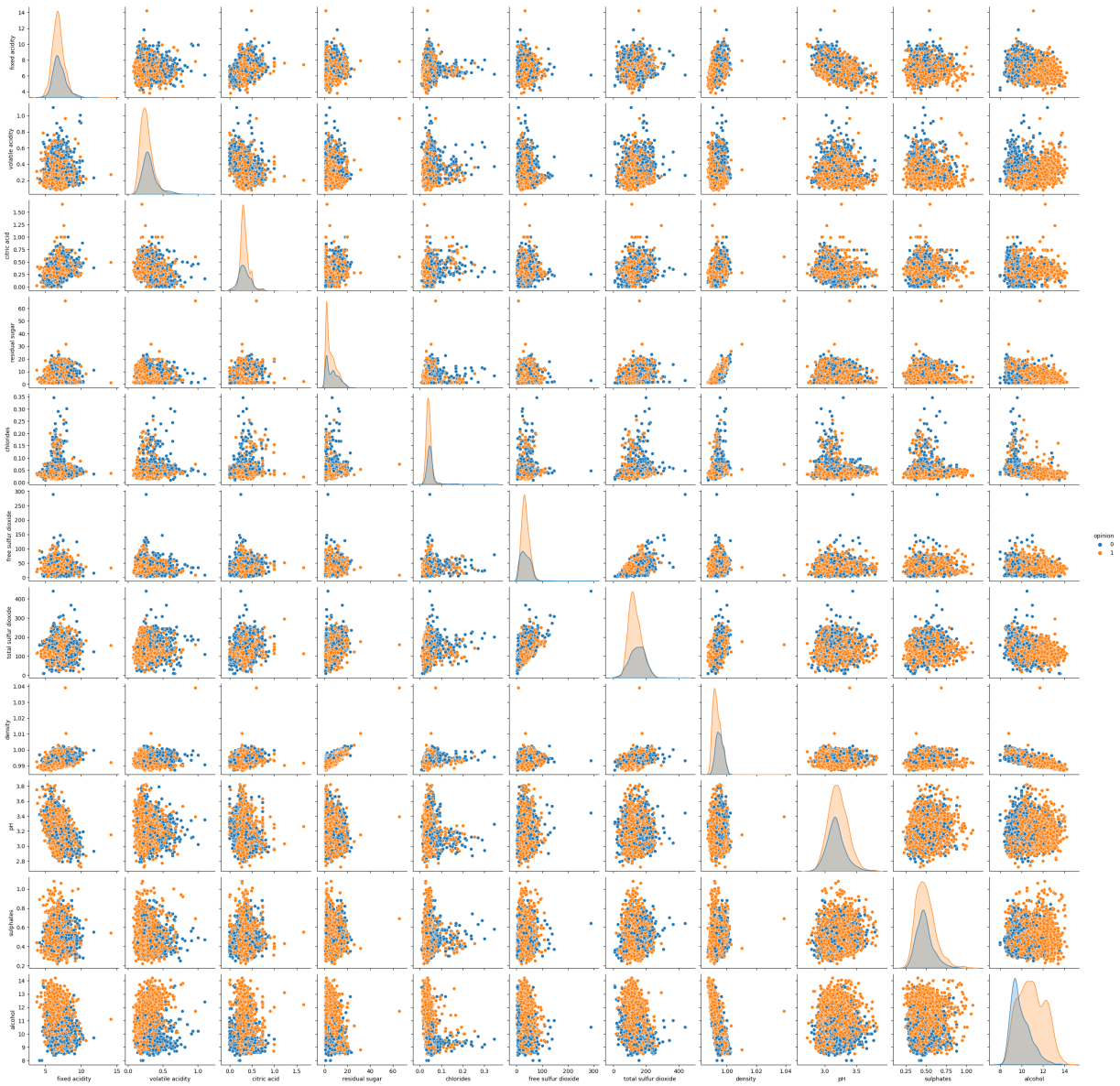
In [18]:
```
ax  = sns.countplot(x=dataset_vinho_branco_tratado["opinion"])
new_labels = ['Ruim', 'Bom']
ax.set_xticklabels(new_labels)
plt.show()
```



Analisando a distribuicao 2 a 2 nenhuma parece linearmente separavel.

In [113…
```
sns.pairplot(dataset_vinho_branco_tratado, hue='opinion')
```

Out[113]:
```
<seaborn.axisgrid.PairGrid at 0x19690138f08>
```



Separando o X e Y:

In [20]:
```
X= dataset_vinho_branco_tratado[numerical_columns]
y = dataset_vinho_branco_tratado[["opinion"]]
```

PARTE 4 - TREINAMENTO E CLASSIFICAÇÃO

4.a - Descreva as etapas necessárias para criar um modelo de classificação eficiente.

Entendimento do Problema: Compreender claramente o problema e definir o objetivo da classificação.

Coleta de Dados: Obter um conjunto de dados relevante e de qualidade.

Pré-processamento dos Dados:

-Limpeza de Dados: Remover valores ausentes ou substituir por media ou mediana, remover duplicados e outliers.

-Transformação de Dados: Converter variaveis categóricas em variáveis numéricas (por exemplo, usando OHE).

-Normalização/Escala: Normalizar ou padronizar os dados .

Divisão dos Dados: Dividir os dados em conjuntos de treinamento e teste. Uma divisão comum é 70% para treinamento e 30% para teste.

Escolha do Algoritmo de Classificação: Escolher um ou mais algoritmos de classificação para comparar os resultados.

Treinamento do Modelo: Treinar o modelo usando o conjunto de treinamento. Usar a validação cruzada para melhorar o desempenho e evitar o overfitting.

Avaliação do Modelo: Avaliar o desempenho do modelo usando métricas apropriadas, como acurácia, precisão, recall, F1-score e a curva ROC-AUC, aplicando-as ao conjunto de teste. O ideal e escolher corretamente a metrica dependendo da figura de merito .

Ajuste e Otimização do Modelo: Fazer ajustes e otimizações conforme necessário. Isso pode incluir ajuste de hiperparâmetros, engenharia de recursos adicionais ou tentativa de diferentes algoritmos.

Validação Final: Teste o modelo final no conjunto de teste para obter uma avaliação imparcial do desempenho.

Implantação do Modelo: Se o modelo estiver pronto para uso, implantar no ambiente de produção.

Monitoramento e Manutenção: Monitorar o desempenho do modelo em produção e fazer atualizações e manutenções conforme necessário para garantir que ele continue a funcionar corretamente.

4.b Treine um modelo de regressão logística usando um modelo de validação cruzada estratificada com k-folds (k=10) para realizar a classificação. Calcule para a base de teste:

i. a média e desvio da acurácia dos modelos obtidos

ii. a média e desvio da precisão dos modelos obtidos

iii. a média e desvio da recall dos modelos obtidos

iv. a média e desvio do f1-score dos modelos obtidos.

A regressão logística é usada para prever variáveis categóricas, geralmente binárias (0 ou 1). Ela estima a probabilidade de um evento ocorre, diferente da Regressao linear que é usada para prever valores contínuos. Ela tenta encontrar a melhor linha reta que representa a relação entre uma variável dependente (y) e uma ou mais variáveis independentes (X).

Para o problema em questao, estamos analisando se um vinho e bom (1) ou ruim (0). Logo, a regressao logistica faz muito mais sentido para esse tipo de problema.

In [115…

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import  RobustScaler


def analyze_performance(model,  y_test,  x_test):
  yhat_test = model.predict(x_test)

  performance_teste = classification_report(y_test, yhat_test, output_dict=True)

  return performance_teste


#Criando uma funcao generica com GS
def calculate_metrics_with_CV_for_model(model,model_name, x_train, x_test, y_train,
    skfold = StratifiedKFold(n_splits=kfold,shuffle=True, random_state=22)
    detalhes_best_results = []
    best_results_per_metric = pd.DataFrame()
    for metric in metrics:
        if( isRandom ):
            gs_search = RandomizedSearchCV(
                estimator=model,
                param_distributions=paramGrid,
                scoring=metric,
                cv=skfold,
                refit=True,
                error_score=0,
                verbose=0,
                n_iter = 100
            )
        else :
            gs_search = GridSearchCV(
                estimator=model,
                param_grid=paramGrid,
                scoring=metric,
                cv=skfold,
                refit=True,
                error_score=0,
                verbose=0,
            )

        gs_search.fit(x_train, y_train)
```

```
        results = pd.DataFrame(gs_search.cv_results_)
        best_result = results.loc[results['rank_test_score'] == 1].head(1)

        best_result['metric'] = metric
        best_result['model'] = model_name
        results['metric'] = metric
        results['model'] = model_name


        y_predict_prob_test = gs_search.best_estimator_.predict_proba(x_test)
        fpr, tpr, _ = roc_curve(y_test, [c[1] for c in y_predict_prob_test])
        roc_auc = roc_auc_score(y_test, [c[1] for c in y_predict_prob_test])
        pd.set_option('display.max_colwidth', None)
        best_result['roc_auc'] = roc_auc

        performance_teste = analyze_performance(gs_search.best_estimator_,  y_test,

        if model_name == "DecisionTree":
            coeficientes = gs_search.best_estimator_.named_steps['model'].feature_i
        else:
            coeficientes = gs_search.best_estimator_.named_steps['model'].coef_[0]

        dicionario_detalhes ={"model": model_name,
                              "params": best_result['params'] ,
                              "coef":coeficientes,
                              "metric": metric,
                              "fpr":[fpr] ,
                              "tpr":[tpr] ,
                              "perf_teste_0":performance_teste['0'],
                              "perf_teste_1": performance_teste['1'],
                              "perf_teste_avg": performance_teste['macro avg'],
                              "modelo_treinado":gs_search.best_estimator_ }

        detalhes_best_results.append(dicionario_detalhes)

        best_results_per_metric = pd.concat([best_results_per_metric, best_result],

    return  best_results_per_metric, detalhes_best_results
```

In [ ]:

```
pipe_rl = Pipeline([
    ('scaler', RobustScaler()),
    ('model', LogisticRegression(random_state=22))
])

model_name = "LogisticRegression"
kfold = 10

paramsGrid = {
    'model__penalty': ['l1', 'l2', 'elasticnet'],
    'model__solver': ['saga'],
    'model__C': np.random.uniform(0.01, 50, 20),
    'model__class_weight': ['balanced', None],
    'model__l1_ratio': [0,0.25,0.5,1]
}

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y
```

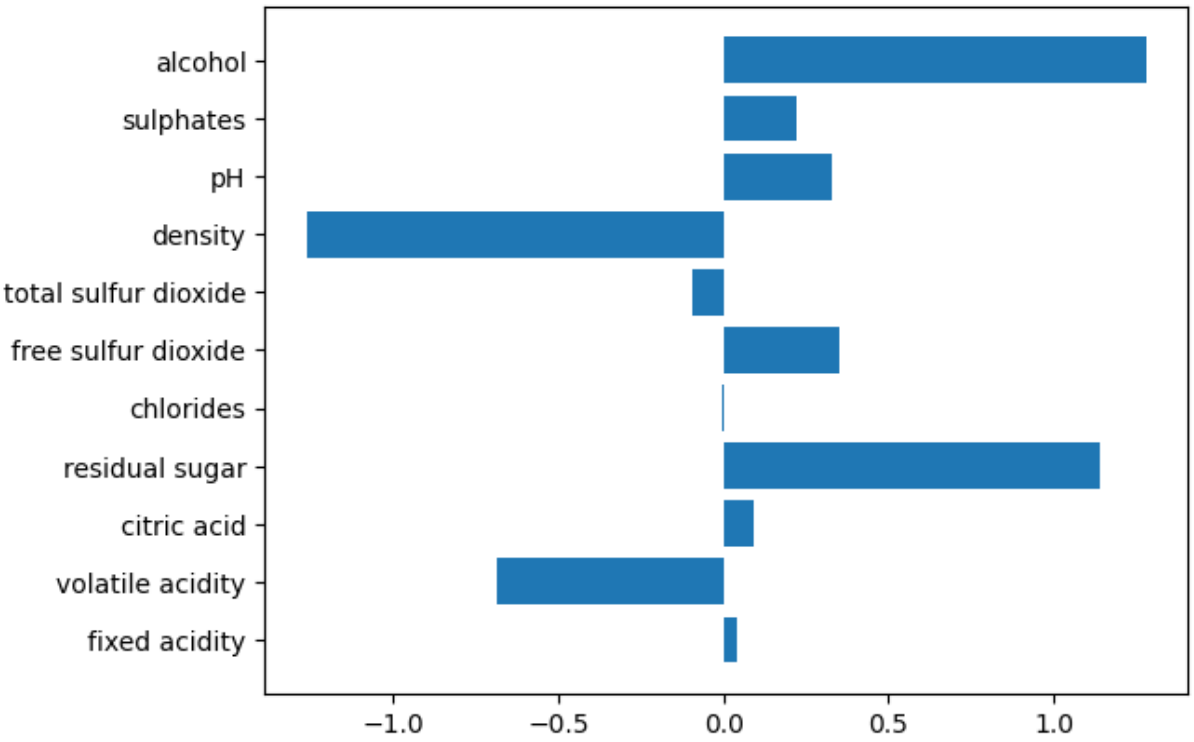
```
result_rl_GS_best_by_metric, detalhes_best_results_rl  = calculate_metrics_with_CV_
```

In [70]:
```
print(result_rl_GS_best_by_metric[["mean_test_score", "std_test_score", "metric", "
chaves_para_selecionar = ['perf_teste_0', 'perf_teste_1', 'perf_teste_avg']
for resultado in detalhes_best_results_rl :
    novo_dict = {chave: resultado[chave] for chave in chaves_para_selecionar if cha
    print(novo_dict)
```

```
   mean_test_score  std_test_score     metric               model  \
0         0.742724        0.024521   accuracy  LogisticRegression
1         0.857062        0.027900     recall  LogisticRegression
2         0.848724        0.019527  precision  LogisticRegression
3         0.814517        0.018556         f1  LogisticRegression


                                                params
0       {'model__solver': 'saga', 'model__penalty': 'l1', 'model__l1_ratio': 0, 'mod
el__class_weight': None, 'model__C': 42.790883544289635}
1     {'model__solver': 'saga', 'model__penalty': 'l1', 'model__l1_ratio': 0.5, 'mod
el__class_weight': None, 'model__C': 42.632518554818134}
2  {'model__solver': 'saga', 'model__penalty': 'l2', 'model__l1_ratio': 1, 'model__c
lass_weight': 'balanced', 'model__C': 6.179375303516394}
3    {'model__solver': 'saga', 'model__penalty': 'l2', 'model__l1_ratio': 0.25, 'mod
el__class_weight': None, 'model__C': 14.283118355151792}
{'perf_teste_0': {'precision': 0.6913183279742765, 'recall': 0.5308641975308642, 'f1
-score': 0.6005586592178771, 'support': 405}, 'perf_teste_1': {'precision': 0.784090
9090909091, 'recall': 0.8778625954198473, 'f1-score': 0.8283313325330133, 'support':
786}, 'perf_teste_avg': {'precision': 0.7377046185325928, 'recall': 0.70436339647535
57, 'f1-score': 0.7144449958754452, 'support': 1191}}
{'perf_teste_0': {'precision': 0.6913183279742765, 'recall': 0.5308641975308642, 'f1
-score': 0.6005586592178771, 'support': 405}, 'perf_teste_1': {'precision': 0.784090
9090909091, 'recall': 0.8778625954198473, 'f1-score': 0.8283313325330133, 'support':
786}, 'perf_teste_avg': {'precision': 0.7377046185325928, 'recall': 0.70436339647535
57, 'f1-score': 0.7144449958754452, 'support': 1191}}
{'perf_teste_0': {'precision': 0.5711538461538461, 'recall': 0.7333333333333333, 'f1
-score': 0.6421621621621622, 'support': 405}, 'perf_teste_1': {'precision': 0.839046
1997019374, 'recall': 0.7162849872773537, 'f1-score': 0.7728208647906657, 'support':
786}, 'perf_teste_avg': {'precision': 0.7051000229278918, 'recall': 0.72480916030534
35, 'f1-score': 0.7074915134764139, 'support': 1191}}
{'perf_teste_0': {'precision': 0.6913183279742765, 'recall': 0.5308641975308642, 'f1
-score': 0.6005586592178771, 'support': 405}, 'perf_teste_1': {'precision': 0.784090
9090909091, 'recall': 0.8778625954198473, 'f1-score': 0.8283313325330133, 'support':
786}, 'perf_teste_avg': {'precision': 0.7377046185325928, 'recall': 0.70436339647535
57, 'f1-score': 0.7144449958754452, 'support': 1191}}
```

Podemos observar que a performance no grupo de teste para a classificacao 0 (vinho ruim) foi bem abaixo para todas as metricas. A possivel causa para isso e dos dados estarem desbalanceados. Outra analise interessante que podemos aplicar e a de importancia das features do melhor modelo para f1:

In [71]:
```
coeficientes_best_f1_rl = [detalhe['coef'] for detalhe in detalhes_best_results_rl
plt.barh(y=numerical_columns, width=coeficientes_best_f1_rl[0])
plt.show()
```



Podemos notar que algumas features nao tem grande importancia para esse modelo, as mais proximas de 0: total sulfur dioxide, chlorides, citric acid e fixed acidity. Vamos realizar um teste retirando esses atributos e rodando o modelo novamente.

In [65]:
```
colunas_selection = ['alcohol', 'sulphates', 'pH', 'density', 'free sulfur dioxide'
X_feature_selection= dataset_vinho_branco_tratado[colunas_selection]
y = dataset_vinho_branco_tratado[["opinion"]]
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(X_feature_selection, y, test_si


result_rl_feature_selection, detalhes_best_results_rl_feature_selection  = calculat

#Ja percebemos que o modelo ficou muito mais eficiente, demorou a metade do tempo p
```

In [67]:
```
print(result_rl_feature_selection[["mean_test_score", "std_test_score", "metric", "
chaves_para_selecionar = ['perf_teste_0', 'perf_teste_1', 'perf_teste_avg']
for resultado in detalhes_best_results_rl_feature_selection :
    novo_dict = {chave: resultado[chave] for chave in chaves_para_selecionar if cha
    print(novo_dict)
```

```
   mean_test_score  std_test_score     metric              model  \
0         0.750282        0.021353   accuracy  LogisticRegression
1         0.863052        0.024746     recall  LogisticRegression
2         0.850549        0.021720  precision  LogisticRegression
3         0.819768        0.015531         f1  LogisticRegression


                                                 params
0       {'model__solver': 'saga', 'model__penalty': 'l2', 'model__l1_ratio': 0.25,
'model__class_weight': None, 'model__C': 3.4255700915370197}
1          {'model__solver': 'saga', 'model__penalty': 'l2', 'model__l1_ratio': 1,
'model__class_weight': None, 'model__C': 3.4255700915370197}
2  {'model__solver': 'saga', 'model__penalty': 'l1', 'model__l1_ratio': 0.5, 'model_
_class_weight': 'balanced', 'model__C': 3.6248402894583163}
3         {'model__solver': 'saga', 'model__penalty': 'l1', 'model__l1_ratio': 0.5,
'model__class_weight': None, 'model__C': 4.788508148883284}
{'perf_teste_0': {'precision': 0.6942675159235668, 'recall': 0.5382716049382716, 'f1
-score': 0.6063977746870653, 'support': 405}, 'perf_teste_1': {'precision': 0.786773
0900798175, 'recall': 0.8778625954198473, 'f1-score': 0.8298256163559832, 'support':
786}, 'perf_teste_avg': {'precision': 0.7405203030016922, 'recall': 0.70806710017905
95, 'f1-score': 0.7181116955215243, 'support': 1191}}
{'perf_teste_0': {'precision': 0.6942675159235668, 'recall': 0.5382716049382716, 'f1
-score': 0.6063977746870653, 'support': 405}, 'perf_teste_1': {'precision': 0.786773
0900798175, 'recall': 0.8778625954198473, 'f1-score': 0.8298256163559832, 'support':
786}, 'perf_teste_avg': {'precision': 0.7405203030016922, 'recall': 0.70806710017905
95, 'f1-score': 0.7181116955215243, 'support': 1191}}
{'perf_teste_0': {'precision': 0.581573896353167, 'recall': 0.7481481481481481, 'f1-
score': 0.654427645788337, 'support': 405}, 'perf_teste_1': {'precision': 0.84776119
40298507, 'recall': 0.72264631043257, 'f1-score': 0.7802197802197802, 'support': 78
6}, 'perf_teste_avg': {'precision': 0.7146675451915089, 'recall': 0.735397229290359,
'f1-score': 0.7173237130040586, 'support': 1191}}
{'perf_teste_0': {'precision': 0.6942675159235668, 'recall': 0.5382716049382716, 'f1
-score': 0.6063977746870653, 'support': 405}, 'perf_teste_1': {'precision': 0.786773
0900798175, 'recall': 0.8778625954198473, 'f1-score': 0.8298256163559832, 'support':
786}, 'perf_teste_avg': {'precision': 0.7405203030016922, 'recall': 0.70806710017905
95, 'f1-score': 0.7181116955215243, 'support': 1191}}
```

Com relacao aos resultados, a reducao das features nao afetou muito a performance.

4.c Treine um modelo de árvores de decisão usando um modelo de validação cruzada estratificada com k-folds (k=10) para realizar a classificação. Calcule para a base de teste:

i. a média e desvio da acurácia dos modelos obtidos

ii. a média e desvio da precisão dos modelos obtidos

iii. a média e desvio da recall dos modelos obtidos

iv. a média e desvio do f1-score dos modelos obtidos

In [72]:
```
from sklearn.tree import DecisionTreeClassifier

#Metricas para a Arvore de decisao
```

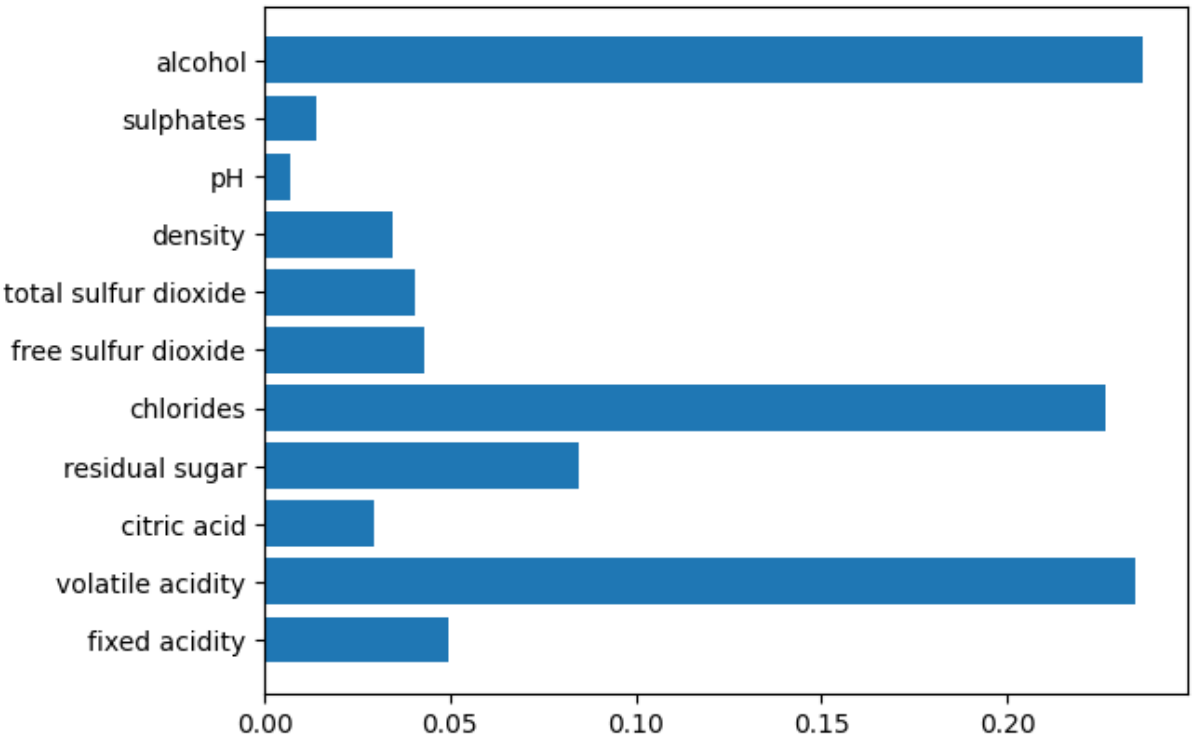
```
pipe_dt = Pipeline([
    ('scaler', RobustScaler()),
    ('model', DecisionTreeClassifier(random_state=22))
])

model_name = "DecisionTree"
kfold = 10

paramsGrid = {
    'model__criterion': ['gini', 'entropy', 'log_loss'],
    'model__max_depth': range(2, 20),
    'model__max_features': ['auto', 'sqrt', 'log2'],
    'model__class_weight': ['balanced', None]
}

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y

result_dt_GS_best_by_metric, detalhes_best_model_dt  = calculate_metrics_with_CV_fo
```

In [74]:

```
print(result_dt_GS_best_by_metric[["mean_test_score", "std_test_score", "metric", "
chaves_para_selecionar = ['perf_teste_0', 'perf_teste_1', 'perf_teste_avg']
for resultado in detalhes_best_model_dt :
    novo_dict = {chave: resultado[chave] for chave in chaves_para_selecionar if cha
    print(novo_dict)
```

```
   mean_test_score  std_test_score     metric         model  \
0         0.736959        0.028583   accuracy  DecisionTree
1         0.872885        0.058195     recall  DecisionTree
2         0.831989        0.031671  precision  DecisionTree
3         0.803938        0.020398         f1  DecisionTree


                                 params
0           {'model__max_features': 'log2', 'model__max_depth': 8, 'model__criterio
n': 'gini', 'model__class_weight': None}
1         {'model__max_features': 'log2', 'model__max_depth': 4, 'model__criterion':
'entropy', 'model__class_weight': None}
2  {'model__max_features': 'sqrt', 'model__max_depth': 8, 'model__criterion': 'entro
py', 'model__class_weight': 'balanced'}
3           {'model__max_features': 'log2', 'model__max_depth': 6, 'model__criterio
n': 'gini', 'model__class_weight': None}
{'perf_teste_0': {'precision': 0.6092896174863388, 'recall': 0.5506172839506173, 'f1
-score': 0.5784695201037614, 'support': 405}, 'perf_teste_1': {'precision': 0.779393
9393939394, 'recall': 0.8180661577608143, 'f1-score': 0.798261949099938, 'support':
786}, 'perf_teste_avg': {'precision': 0.6943417784401391, 'recall': 0.68434172085571
58, 'f1-score': 0.6883657346018497, 'support': 1191}}
{'perf_teste_0': {'precision': 0.5859649122807018, 'recall': 0.4123456790123457, 'f1
-score': 0.4840579710144928, 'support': 405}, 'perf_teste_1': {'precision': 0.737306
8432671082, 'recall': 0.8498727735368957, 'f1-score': 0.7895981087470451, 'support':
786}, 'perf_teste_avg': {'precision': 0.661635877773905, 'recall': 0.631109226274620
7, 'f1-score': 0.636828039880769, 'support': 1191}}
{'perf_teste_0': {'precision': 0.5734939759036145, 'recall': 0.5876543209876544, 'f1
-score': 0.5804878048780489, 'support': 405}, 'perf_teste_1': {'precision': 0.784793
8144329897, 'recall': 0.7748091603053435, 'f1-score': 0.7797695262483995, 'support':
786}, 'perf_teste_avg': {'precision': 0.6791438951683021, 'recall': 0.68123174064649
89, 'f1-score': 0.6801286655632242, 'support': 1191}}
{'perf_teste_0': {'precision': 0.5793650793650794, 'recall': 0.5407407407407407, 'f1
-score': 0.5593869731800766, 'support': 405}, 'perf_teste_1': {'precision': 0.771217
7121771218, 'recall': 0.7977099236641222, 'f1-score': 0.7842401500938087, 'support':
786}, 'perf_teste_avg': {'precision': 0.6752913957711006, 'recall': 0.66922533220243
15, 'f1-score': 0.6718135616369426, 'support': 1191}}
```

In [76]:
```
coeficientes_best_f1_dt = [detalhe['coef'] for detalhe in detalhes_best_model_dt if
plt.barh(y=numerical_columns, width=coeficientes_best_f1_dt[0])
plt.show()
```



In [77]:
```
colunas_selection = ['alcohol', 'density', 'total sulfur dioxide' ,'chlorides','fre
X_feature_selection= dataset_vinho_branco_tratado[colunas_selection]
y = dataset_vinho_branco_tratado[["opinion"]]
```

In [78]:
```
x_train, x_test, y_train, y_test = train_test_split(X_feature_selection, y, test_si

result_dt_feature_selection, detalhes_best_model_dt_feature_selection  = calculate_
```

In [79]:
```
print(result_dt_feature_selection[["mean_test_score", "std_test_score", "metric", "
chaves_para_selecionar = ['perf_teste_0', 'perf_teste_1', 'perf_teste_avg']
for resultado in detalhes_best_model_dt_feature_selection :
    novo_dict = {chave: resultado[chave] for chave in chaves_para_selecionar if cha
    print(novo_dict)
```

```
   mean_test_score  std_test_score     metric         model  \
0         0.740193        0.034951   accuracy  DecisionTree
1         0.829259        0.028161     recall  DecisionTree
2         0.845762        0.036823  precision  DecisionTree
3         0.808705        0.028736         f1  DecisionTree


                              params
0     {'model__max_features': 'auto', 'model__max_depth': 7, 'model__criterion': 'en
tropy', 'model__class_weight': None}
1        {'model__max_features': 'auto', 'model__max_depth': 7, 'model__criterion':
'gini', 'model__class_weight': None}
2  {'model__max_features': 'auto', 'model__max_depth': 2, 'model__criterion': 'gin
i', 'model__class_weight': 'balanced'}
3     {'model__max_features': 'auto', 'model__max_depth': 7, 'model__criterion': 'en
tropy', 'model__class_weight': None}
{'perf_teste_0': {'precision': 0.6208791208791209, 'recall': 0.5580246913580247, 'f1
-score': 0.58777633289987, 'support': 405}, 'perf_teste_1': {'precision': 0.78355501
81378477, 'recall': 0.8244274809160306, 'f1-score': 0.8034717916924985, 'support': 7
86}, 'perf_teste_avg': {'precision': 0.7022170695084843, 'recall': 0.691226086137027
6, 'f1-score': 0.6956240622961842, 'support': 1191}}
{'perf_teste_0': {'precision': 0.5960099750623441, 'recall': 0.5901234567901235, 'f1
-score': 0.5930521091811415, 'support': 405}, 'perf_teste_1': {'precision': 0.789873
417721519, 'recall': 0.7938931297709924, 'f1-score': 0.7918781725888325, 'support':
786}, 'perf_teste_avg': {'precision': 0.6929416963919315, 'recall': 0.69200829328055
79, 'f1-score': 0.692465140884987, 'support': 1191}}
{'perf_teste_0': {'precision': 0.42408376963350786, 'recall': 0.8, 'f1-score': 0.554
3199315654406, 'support': 405}, 'perf_teste_1': {'precision': 0.810304449648712, 're
call': 0.4402035623409669, 'f1-score': 0.5704863973619125, 'support': 786}, 'perf_te
ste_avg': {'precision': 0.6171941096411099, 'recall': 0.6201017811704834, 'f1-scor
e': 0.5624031644636766, 'support': 1191}}
{'perf_teste_0': {'precision': 0.6208791208791209, 'recall': 0.5580246913580247, 'f1
-score': 0.58777633289987, 'support': 405}, 'perf_teste_1': {'precision': 0.78355501
81378477, 'recall': 0.8244274809160306, 'f1-score': 0.8034717916924985, 'support': 7
86}, 'perf_teste_avg': {'precision': 0.7022170695084843, 'recall': 0.691226086137027
6, 'f1-score': 0.6956240622961842, 'support': 1191}}
```

4.d Treine um modelo de SVM usando um modelo de validação cruzada estratificada com k-folds (k=10) para realizar a classificação. Calcule para a base de teste:

i. a média e desvio da acurácia dos modelos obtidos

ii. a média e desvio da precisão dos modelos obtidos

iii. a média e desvio da recall dos modelos obtidos

iv. a média e desvio do f1-score dos modelos obtidos

In [81]:
```
from sklearn.svm import SVC

#Metricas para SVM
```

```
pipe_svm = Pipeline([
    ('scaler', RobustScaler()),
    ('model', SVC(random_state=22, probability=True))
])

model_name = "SVM"
kfold = 10

#result_df_svm, roc_auc_svm = calculate_metrics_for_model(model,model_name,X, y, kf

paramsGrid = {
    'model__kernel': [ 'linear'],
    'model__C': np.random.uniform(0.01, 10, 50),
    'model__gamma': np.random.uniform(0.001, 1, 50),
    'model__degree': [2, 3, 4],
    'model__class_weight': ['balanced', None]
}


x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y

result_svm_GS_best_by_metric, detalhes_best_model_svm  = calculate_metrics_with_CV_
```

In [82]:
```
print(result_svm_GS_best_by_metric[["mean_test_score", "std_test_score", "metric",
```

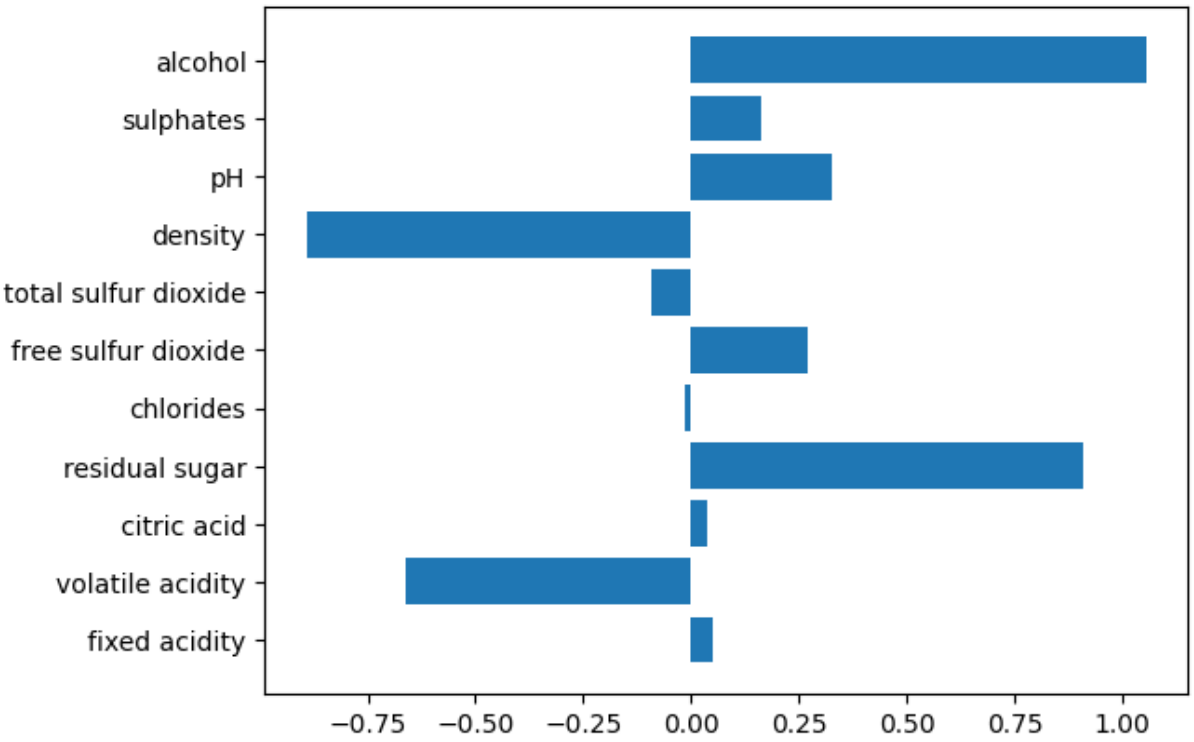
```
   mean_test_score  std_test_score     metric model  \
0         0.749556        0.026046   accuracy   SVM
1         0.872336        0.020588     recall   SVM
2         0.855207        0.020945  precision   SVM
3         0.819894        0.019361         f1   SVM


                                                                 params
0          {'model__kernel': 'linear', 'model__gamma': 0.1882929625754767, 'model__d
egree': 4, 'model__class_weight': None, 'model__C': 9.440754944583237}
1        {'model__kernel': 'linear', 'model__gamma': 0.5589136857194006, 'model__deg
ree': 3, 'model__class_weight': None, 'model__C': 0.11582493338684478}
2  {'model__kernel': 'linear', 'model__gamma': 0.16016915815651608, 'model__degree':
3, 'model__class_weight': 'balanced', 'model__C': 0.9072779686570348}
3          {'model__kernel': 'linear', 'model__gamma': 0.4612700690867606, 'model__d
egree': 4, 'model__class_weight': None, 'model__C': 7.679272171466217}
```

In [83]:
```
print(result_svm_GS_best_by_metric[["mean_test_score", "std_test_score", "metric",
chaves_para_selecionar = ['perf_teste_0', 'perf_teste_1', 'perf_teste_avg']
for resultado in detalhes_best_model_svm :
    novo_dict = {chave: resultado[chave] for chave in chaves_para_selecionar if cha
    print(novo_dict)
```

```
   mean_test_score  std_test_score     metric model  \
0         0.749556        0.026046   accuracy   SVM
1         0.872336        0.020588     recall   SVM
2         0.855207        0.020945  precision   SVM
3         0.819894        0.019361         f1   SVM


                                                              params
0          {'model__kernel': 'linear', 'model__gamma': 0.1882929625754767, 'model__d
egree': 4, 'model__class_weight': None, 'model__C': 9.440754944583237}
1        {'model__kernel': 'linear', 'model__gamma': 0.5589136857194006, 'model__deg
ree': 3, 'model__class_weight': None, 'model__C': 0.11582493338684478}
2  {'model__kernel': 'linear', 'model__gamma': 0.16016915815651608, 'model__degree':
3, 'model__class_weight': 'balanced', 'model__C': 0.9072779686570348}
3          {'model__kernel': 'linear', 'model__gamma': 0.4612700690867606, 'model__d
egree': 4, 'model__class_weight': None, 'model__C': 7.679272171466217}
{'perf_teste_0': {'precision': 0.685064935064935, 'recall': 0.5209876543209877, 'f1-
score': 0.5918653576437587, 'support': 405}, 'perf_teste_1': {'precision': 0.7802944
507361268, 'recall': 0.8765903307888041, 'f1-score': 0.8256440982624327, 'support':
786}, 'perf_teste_avg': {'precision': 0.7326796929005309, 'recall': 0.69878899255489
59, 'f1-score': 0.7087547279530957, 'support': 1191}}
{'perf_teste_0': {'precision': 0.6912751677852349, 'recall': 0.508641975308642, 'f1-
score': 0.5860597439544808, 'support': 405}, 'perf_teste_1': {'precision': 0.7771556
550951848, 'recall': 0.8829516539440203, 'f1-score': 0.8266825491363907, 'support':
786}, 'perf_teste_avg': {'precision': 0.7342154114402099, 'recall': 0.69579681462633
11, 'f1-score': 0.7063711465454358, 'support': 1191}}
{'perf_teste_0': {'precision': 0.5600739371534196, 'recall': 0.7481481481481481, 'f1
-score': 0.6405919661733614, 'support': 405}, 'perf_teste_1': {'precision': 0.843076
9230769231, 'recall': 0.6972010178117048, 'f1-score': 0.7632311977715877, 'support':
786}, 'perf_teste_avg': {'precision': 0.7015754301151713, 'recall': 0.72267458297992
65, 'f1-score': 0.7019115819724746, 'support': 1191}}
{'perf_teste_0': {'precision': 0.685064935064935, 'recall': 0.5209876543209877, 'f1-
score': 0.5918653576437587, 'support': 405}, 'perf_teste_1': {'precision': 0.7802944
507361268, 'recall': 0.8765903307888041, 'f1-score': 0.8256440982624327, 'support':
786}, 'perf_teste_avg': {'precision': 0.7326796929005309, 'recall': 0.69878899255489
59, 'f1-score': 0.7087547279530957, 'support': 1191}}
```

In [84]:

```
coeficientes_best_f1_svm = [detalhe['coef'] for detalhe in detalhes_best_model_svm
plt.barh(y=numerical_columns, width=coeficientes_best_f1_svm[0])
plt.show()
```



In [85]:
```
colunas_selection = ['alcohol', 'pH', 'density','free sulfur dioxide', 'residual su
X_feature_selection= dataset_vinho_branco_tratado[colunas_selection]
y = dataset_vinho_branco_tratado[["opinion"]]
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(X_feature_selection, y, test_si

result_svm_feature_selection, detalhes_best_model_svm_feature_selection  = calculat
```

In [ ]:
```
print(result_svm_feature_selection[["mean_test_score", "std_test_score", "metric",
chaves_para_selecionar = ['perf_teste_0', 'perf_teste_1', 'perf_teste_avg']
for resultado in detalhes_best_model_svm_feature_selection :
    novo_dict = {chave: resultado[chave] for chave in chaves_para_selecionar if cha
    print(novo_dict)
```

PARTE 5 - COMPARAÇÃO DOS MODELOS

Observando o resultado geral das performances na base de treino como uma tabela de resultados, podemos ver que com excecao do melhor resultado do SVM considerando a metrica de recall, todos os demais algoritmos obtiveram um resultado muito proximo na media. Para o problema em questao, a metrica que faria mais sentido seria a F1, pois ha desbalanceamento das classes e para esse problema em questao nao estamos focados em resolver o recall ou acuracia.

In [86]:
```
pd.set_option('display.max_colwidth', None)

final_best_results= pd.concat([result_svm_GS_best_by_metric, result_dt_GS_best_by_m

results_best = final_best_results[["mean_test_score", "std_test_score", "metric", "
```

```
results_best
```

Out[86]:

	mean_test_score	std_test_score	metric	model	params
0	0.749556	0.026046	accuracy	SVM	{'model__kernel': 'linear', 'model__gamma': 0.1882929625754767, 'model__degree': 4, 'model__class_weight': None, 'model__C': 9.440754944583237}
1	0.872336	0.020588	recall	SVM	{'model__kernel': 'linear', 'model__gamma': 0.5589136857194006, 'model__degree': 3, 'model__class_weight': None, 'model__C': 0.11582493338684478}
2	0.855207	0.020945	precision	SVM	{'model__kernel': 'linear', 'model__gamma': 0.16016915815651608, 'model__degree': 3, 'model__class_weight': 'balanced', 'model__C': 0.9072779686570348}
3	0.819894	0.019361	f1	SVM	{'model__kernel': 'linear', 'model__gamma': 0.4612700690867606, 'model__degree': 4, 'model__class_weight': None, 'model__C': 7.679272171466217}
4	0.736959	0.028583	accuracy	DecisionTree	{'model__max_features': 'log2', 'model__max_depth': 8, 'model__criterion': 'gini', 'model__class_weight': None}
5	0.872885	0.058195	recall	DecisionTree	{'model__max_features': 'log2', 'model__max_depth': 4, 'model__criterion': 'entropy', 'model__class_weight': None}
6	0.831989	0.031671	precision	DecisionTree	{'model__max_features': 'sqrt', 'model__max_depth': 8, 'model__criterion': 'entropy', 'model__class_weight': 'balanced'}

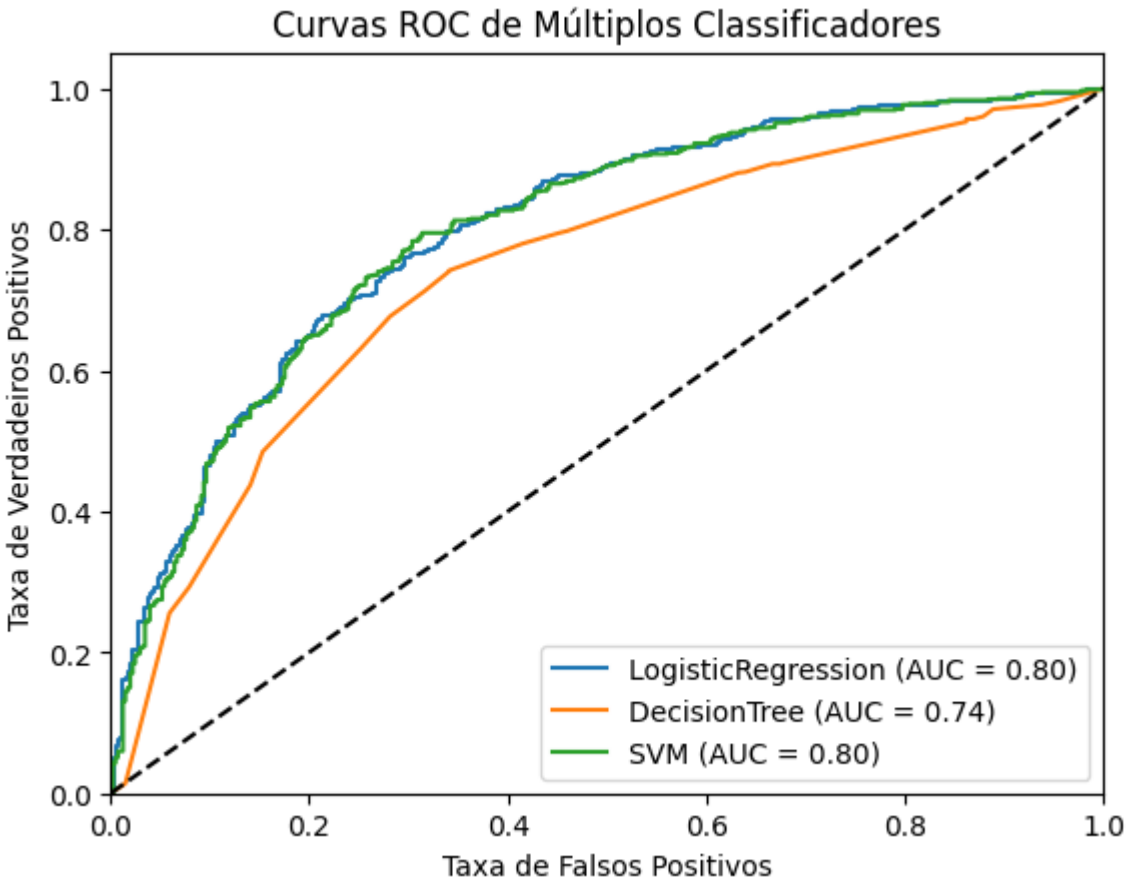
	mean_test_score	std_test_score	metric	model	params
7	0.803938	0.020398	f1	DecisionTree	{'model__max_features': 'log2', 'model__max_depth': 6, 'model__criterion': 'gini', 'model__class_weight': None}
8	0.742724	0.024521	accuracy	LogisticRegression	{'model__solver': 'saga', 'model__penalty': 'l1', 'model__l1_ratio': 0, 'model__class_weight': None, 'model__C': 42.790883544289635}
9	0.857062	0.027900	recall	LogisticRegression	{'model__solver': 'saga', 'model__penalty': 'l1', 'model__l1_ratio': 0.5, 'model__class_weight': None, 'model__C': 42.632518554818134}
10	0.848724	0.019527	precision	LogisticRegression	{'model__solver': 'saga', 'model__penalty': 'l2', 'model__l1_ratio': 1, 'model__class_weight': 'balanced', 'model__C': 6.179375303516394}
11	0.814517	0.018556	f1	LogisticRegression	{'model__solver': 'saga', 'model__penalty': 'l2', 'model__l1_ratio': 0.25, 'model__class_weight': None, 'model__C': 14.283118355151792}

Observando a comparacao das curvas ROC de todos os melhores modelos com metrica f1:

In [96]:
```
final_detalhes_modelos_f1 = []

rl_ = list(map( lambda x:{'fpr': x['fpr'],'tpr':x['tpr'],'model': x['model']} ,filt

dt_ = list(map( lambda x:{'fpr': x['fpr'],'tpr':x['tpr'],'model': x['model']} ,filt

svm_ = list(map( lambda x:{'fpr': x['fpr'],'tpr':x['tpr'],'model': x['model']} ,fil

final_detalhes_modelos_f1.append(rl_)
final_detalhes_modelos_f1.append(dt_)
final_detalhes_modelos_f1.append(svm_)
```

In [104…
```
from sklearn.metrics import auc

plt.figure()
```

```
for item in final_detalhes_modelos_f1:
    plt.plot(item['fpr'][0], item['tpr'][0], label=f'{item["model"]} (AUC = {auc(it

# Configurações do gráfico
plt.plot([0, 1], [0, 1], 'k--')  # Linha diagonal
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taxa de Falsos Positivos')
plt.ylabel('Taxa de Verdadeiros Positivos')
plt.title('Curvas ROC de Múltiplos Classificadores')
plt.legend(loc='lower right')

# Mostrar o gráfico
plt.show()
```



Olhando para as curvas ROC podemos ver que RL e SVM tiveram um resultado muito proximos.

In [106…

```
filtered_df_f1 = results_best.query('metric == "f1"')
filtered_df_f1
```

Out[106]:

	mean_test_score	std_test_score	metric	model	params
3	0.819894	0.019361	f1	SVM	{'model__kernel': 'linear', 'model__gamma': 0.4612700690867606, 'model__degree': 4, 'model__class_weight': None, 'model__C': 7.679272171466217}
7	0.803938	0.020398	f1	DecisionTree	{'model__max_features': 'log2', 'model__max_depth': 6, 'model__criterion': 'gini', 'model__class_weight': None}
11	0.814517	0.018556	f1	LogisticRegression	{'model__solver': 'saga', 'model__penalty': 'l2', 'model__l1_ratio': 0.25, 'model__class_weight': None, 'model__C': 14.283118355151792}

Se fossemos parar os treinos apenas executando o Random Grid Search com alguns parametros sendo testados, comparando todas os resultados desses testes, optando pelo F1 como metrica, vimos que a regressao logistica teve media de 0.8145 e o SVM uma media de 0.8198. Analisando a curva ROC anteriormente plotadas podemos observar que a curva ROC para os dois modelos tambem sao muito similares. Como o SVM e um algoritmo muito parrudo e sua performance e inferior ao de RL, acredito que a melhor opcao para esse problema em questao seria o RL com metrica F1 e os parametros: {'model__solver': 'saga', 'model__penalty': 'l2', 'model__l1_ratio': 0.25, 'model__class_weight': None, 'model__C': 14.283118355151792}.

Podemos melhorar a performance desse modelo escolhido inicialmente, a partir da analise mais detalhada de como os parametros afetam a efiencia do modelo.

In [108…

```
pipe_rl = Pipeline([
    ('scaler', RobustScaler()),
    ('model', LogisticRegression(random_state=22, ))
])

model_name = "LogisticRegression"
kfold = 10

#focando nos parametros q tiveram mais sucesso , variando apenas o C
paramsGrid = {
    'model__penalty': ['l2'],
    'model__solver': [ 'saga'],
    'model__C': np.arange(2, 30, 1),
    'model__l1_ratio':[0.25],
    'model__class_weight': [None]
```

```
}

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y


result_rl_final_best_by_metric, detalhes_rl_final_best_by_metric  = calculate_metri

#Vemos que os resultados nao melhoraram muito. Se essa eficiencia nao e suficiente
# o ideal seria tentar focar nas features e na padronizacao dos dados. Talvez aplic
# ou ainda aplicar uma normalizacao dos dados.
```

Analisando melhor os inputs, vamos verificar se existem outliers:

In [109…
```
from scipy import stats

z_scores = np.abs(stats.zscore(dataset_vinho_branco_tratado[numerical_columns]))

# Identificar linhas que NÃO são outliers
dataset_vinho_branco_sem_outlier = dataset_vinho_branco_tratado[(z_scores < 3).all(

X2= dataset_vinho_branco_sem_outlier[numerical_columns]
Y2 = dataset_vinho_branco_sem_outlier[["opinion"]]
```

Vamos rodar o um GridSearch utilizando os novos dados sem outliers:

In [116…
```
pipe_rl = Pipeline([
    ('scaler', RobustScaler()),
    ('model', LogisticRegression(random_state=22, ))
])

model_name = "LogisticRegression"
kfold = 10


paramsGrid = {
    'model__penalty': ['l2'],
    'model__solver': [ 'saga'],
    'model__C': np.arange(2, 30, 1),
    'model__l1_ratio':[0.25],
    'model__class_weight': [None]
}
x_train, x_test, y_train, y_test = train_test_split(X2, Y2, test_size=0.3, stratify


rl_sem_outlier,detalhe_rl_sem_outlier  = calculate_metrics_with_CV_for_model(pipe_r
```

In [117…
```
print(rl_sem_outlier[["mean_test_score", "std_test_score", "metric", "model", "para
chaves_para_selecionar = ['perf_teste_0', 'perf_teste_1', 'perf_teste_avg']
for resultado in detalhe_rl_sem_outlier :
    novo_dict = {chave: resultado[chave] for chave in chaves_para_selecionar if cha
    print(novo_dict)
```

```
   mean_test_score  std_test_score metric               model  \
0         0.827111        0.015504     f1  LogisticRegression

                                  params
0  {'model__C': 3, 'model__class_weight': None, 'model__l1_ratio': 0.25, 'model__pen
alty': 'l2', 'model__solver': 'saga'}
{'perf_teste_0': {'precision': 0.623574144486692, 'recall': 0.4619718309859155, 'f1-
score': 0.5307443365695792, 'support': 355}, 'perf_teste_1': {'precision': 0.7698795
180722892, 'recall': 0.8658536585365854, 'f1-score': 0.8150510204081632, 'support':
738}, 'perf_teste_avg': {'precision': 0.6967268312794905, 'recall': 0.66391274476125
04, 'f1-score': 0.6728976784888712, 'support': 1093}}
```

Conseguimos melhorar um pouco o modelo, com uma media de score f1 de 0.82711 no grupo de treino e uma media de 0.6728 para o grupo de teste.

PARTE 6 - ESCOLHA DO MODELO

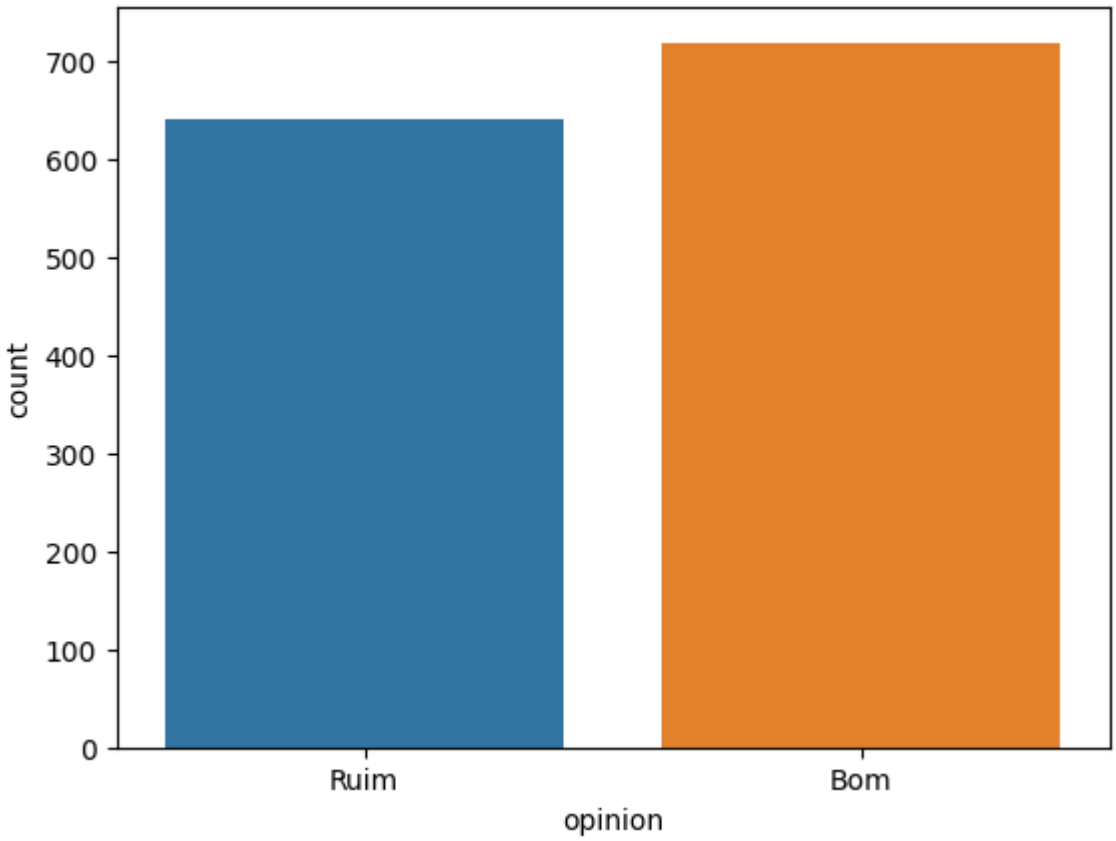
Com a escolha do melhor modelo, use os dados de vinho tinto, presentes na base original e faça a inferência (não é para treinar novamente!!!) para saber quantos vinhos são bons ou ruins. Utilize o mesmo critério utilizado com os vinhos brancos, para comparar o desempenho do modelo. Ele funciona da mesma forma para essa nova base? Justifique.

O modelo escolhido e o de Regressao Logistica , utilizando os parametros: {'model__C': 3, 'model__class_weight': None, 'model__l1_ratio': 0.25, 'model__penalty': 'l2', 'model__solver': 'saga'} , pois ele conseguiu alcancar um desempenho muito bom comparado com os demais olhando a metrica de f1 (tanto no grupo de teste quanto no de treino) alem disso, e um modelo com menor complexidade, ele pode ser processado utilizando muito menos recursos e tempo (comparando com o SVM que teve um desempenho de metrica um pouco superior).

In [137…

```
dataset_original = pd.read_csv('winequalityN.csv', sep=',', decimal='.')

dataset_filtrado_red = dataset_original[dataset_original['type'] == "red"]

dataset_filtrado_red['opinion'] = np.where(dataset_filtrado_red['quality'] > 5 , 1,

dataset_filtrado_red.drop(columns='quality', inplace=True)

dataset_filtrado_red[dataset_filtrado.columns] = dataset_filtrado_red[dataset_filtr


dataset_vinho_tinto_tratado = dataset_filtrado_red.drop_duplicates()


ax  = sns.countplot(x=dataset_vinho_tinto_tratado["opinion"])
new_labels = ['Ruim', 'Bom']
ax.set_xticklabels(new_labels)
plt.show()
```

```
X_red= dataset_vinho_tinto_tratado[numerical_columns]
y_red = dataset_vinho_tinto_tratado[["opinion"]]
```



Podemos ver que a base de dados para o vinho tinto e mais balanceada do que a de vinho branco. Utilizando o modelo treinado anteriormente para vinho branco, vamos inferir a qualidade do vinho para essa base :

In [144…

```
X_red_tratado = RobustScaler().fit_transform(X_red)

predict_vinho_tinto = detalhe_rl_sem_outlier[0]['modelo_treinado'].predict(X_red_tr
predict_proba_vinho_tinto = detalhe_rl_sem_outlier[0]['modelo_treinado'].predict_pr

fpr_tinto, tpr_tinto, _ = roc_curve(y_red, [c[1] for c in predict_proba_vinho_tinto
roc_auc_tinto = roc_auc_score(y_red, [c[1] for c in predict_proba_vinho_tinto])

performance_teste_tinto = classification_report(y_red, predict_vinho_tinto)

print(performance_teste_tinto)
```

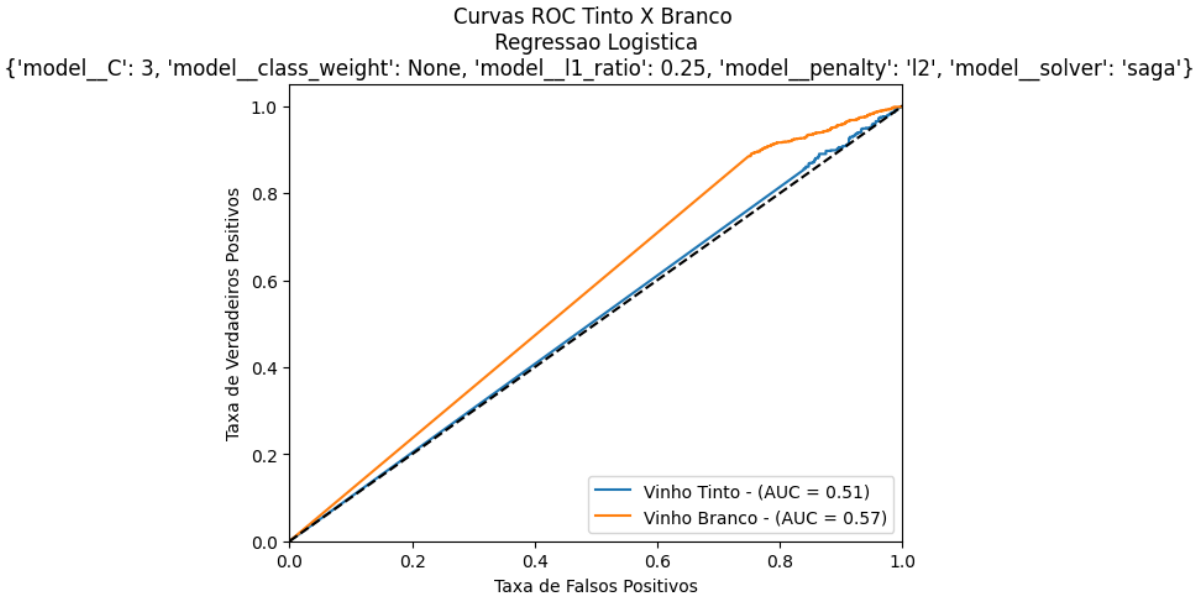
```
              precision    recall  f1-score   support

           0       0.51      0.14      0.21       640
           1       0.53      0.88      0.67       719

    accuracy                           0.53      1359
   macro avg       0.52      0.51      0.44      1359
weighted avg       0.52      0.53      0.45      1359
```

Bem o modelo ter tido um desempenho inferior na base de vinho branco pode ser explicado pelo fato de que talvez as caracteristicas dos vinhos brancos sao diferentes na hora de se classificar em bom ou ruim .

In [143…
```
X_branco_tratado = RobustScaler().fit_transform(X)
predict_vinho_branco = detalhe_rl_sem_outlier[0]['modelo_treinado'].predict(X_branc
predict_proba_vinho_branco = detalhe_rl_sem_outlier[0]['modelo_treinado'].predict_p

fpr_branco, tpr_branco, _ = roc_curve(y, [c[1] for c in predict_proba_vinho_branco]
roc_auc_branco = roc_auc_score(y, [c[1] for c in predict_proba_vinho_branco])

performance_teste_branco = classification_report(y, predict_vinho_branco)

print(performance_teste_branco)
```

```
              precision    recall  f1-score   support

           0       0.55      0.22      0.31      1351
           1       0.69      0.91      0.79      2619

    accuracy                           0.67      3970
   macro avg       0.62      0.56      0.55      3970
weighted avg       0.64      0.67      0.62      3970
```

In [159…
```
from sklearn.metrics import auc

plt.figure()


plt.plot(fpr_tinto, tpr_tinto, label=f'Vinho Tinto - (AUC = {roc_auc_tinto:.2f})')
plt.plot(fpr_branco, tpr_branco, label=f'Vinho Branco - (AUC = {roc_auc_branco:.2f}


# Configurações do gráfico
plt.plot([0, 1], [0, 1], 'k--')  # Linha diagonal
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taxa de Falsos Positivos')
plt.ylabel('Taxa de Verdadeiros Positivos')
plt.title(f'Curvas ROC Tinto X Branco \n Regressao Logistica \n {dict(detalhe_rl_se
plt.legend(loc='lower right')

# Mostrar o gráfico
plt.show()
```



Olhando as metricas para o modelo sendo aplicado na base completa do vinho branco, estranhamente as metricas nao estao boas tambem. Alguns fatores podem ser a resposta para isso:

- O modelo teve overfitting, capturando bem apenas os dados de treinamento e nao sendo generico suficiente para diferentes combinacoes de dados.
- Tratamento de dados nulos. Eu tratei os dados nulos aplicando a media , porem pode nao ter sido uma boa opcao. Seria uma boa retornar o modelo CRISP, aplicar uma logica diferente nos dados nulos e avaliar os modelos novamente.
- Talvez a quantidade de dados apresentados nao foi suficiente para execucao de um bom modelo, talvez o problema e mais complexo do que parece.
- Machine learning e um eterno esforco de ida e volta , tanto que foi criado a metodologia CRISP, que e iterativa, incentivando a revisitar as fases anteriores com base nos insights obtidos durante o processo.

PARTE 7 - LINK PARA GITHUB

https://github.com/rachelreuters/posGraduacaoIA/blob/main/Classificacao/TRABALHOPD/TRAB/