Projeto EDA | Machine Learning I

- Anderson Miranda - ID: 1116003

- Curso: **Data Science - Santander Coders 2023 | 2º Semestre**
- **ADA Tech** em parceria com **Banco Santander**
- Módulo: **Machine Learning I**
- Facilitador: **Thiago Tavares**

Objetivo Geral:

Este projeto tem por objetivo criar um modelo de Machine Learning com dados reais, utilizando os recursos da disciplica de Machine Learning I, do Curso Data Science - Santander Coders, fazendo uso das bibliotecas: Scikit Learn, Seaborn, Matplotlib, Pandas e Numpy, possibilitando a classificação de um determinado recurso, com aplicação em negócios reais.

Sobre as Ferramentas:

- **Scikit Learn**: biblioteca de Machine Learning (Aprendizado de Máquina), com código aberto, para programadores em linguagem Python;
- **Matplotlib**: biblioteca para Data Visualization (Visualização de Dados), visando a criação de gráficos para programadores da linguagem Python;
- **Seaborn**: biblioteca para Data Visualization (Visualização de Dados), com alguns componetes mais simplificados, visando a criação de gráficos para programadores da linguagem Python, baseada no Matplotlib;
- **Pandas**: biblioteca para manipulação e análise de dados, em estruturas tabulares, para programadores da linguagem Python;
- **Numpy**: biblioteca de linguagem Python, para processamento de estrutura de dados de multiplas dimensões, utilizado para manipulação e análise de dados estatísticos;

Dos Dados:

Água potável

O acesso à água potável é essencial para a saúde, um direito humano básico e uma componente de uma política eficaz de protecção da saúde. Isto é importante como uma questão de saúde e desenvolvimento a nível nacional, regional e local. Em algumas regiões, foi demonstrado que os investimentos no abastecimento de água e no saneamento podem produzir um benefício económico líquido, uma vez que as reduções nos efeitos adversos para a

produzir um benefício econômico líquido, uma vez que as reduções nos efeitos adversos para a saúde e nos custos dos cuidados de saúde superam os custos da realização das intervenções.

Data Source:

https://www.kaggle.com/datasets/adityakadiwal/water-potability/data

Descrição dos datasets:

1. pH value: O pH é um parâmetro importante na avaliação do equilíbrio ácido-base da água. É também o indicador da condição ácida ou alcalina do estado da água. A WHO recomendou o limite máximo permitido de pH de 6,5 a 8,5. Os actuais intervalos de investigação foram de 6,52 a 6,83, que estão dentro dos padrões da WHO.
2. Hardness: A dureza é causada principalmente por sais de cálcio e magnésio. Esses sais são dissolvidos em depósitos geológicos através dos quais a água viaja. O período de tempo que a água fica em contato com o material produtor de dureza ajuda a determinar quanta dureza existe na água bruta. A dureza foi originalmente definida como a capacidade da água de precipitar sabão causada pelo Cálcio e pelo Magnésio.
3. Solids (Total de solidos dissolvidos): A água tem a capacidade de dissolver uma ampla gama de minerais ou sais inorgânicos e alguns orgânicos, como potássio, cálcio, sódio, bicarbonatos, cloretos, magnésio, sulfatos, etc. Esses minerais produziam sabor indesejado e cor diluída na aparência da água. Este é o parâmetro importante para o uso da água. A água com alto valor de TDS indica que a água é altamente mineralizada. O limite desejável para TDS é 500 mg/l o limite máximo é 1000 mg/l, prescrito para beber.
4. Chloramines: O cloro e a cloramina são os principais desinfetantes utilizados nos sistemas públicos de água. As cloraminas são mais comumente formadas quando a amônia é adicionada ao cloro para tratar a água potável. Níveis de cloro de até 4 miligramas por litro (mg/L ou 4 partes por milhão (ppm)) são considerados seguros na água potável.
5. Sulfate: Os sulfatos são substâncias naturais encontradas em minerais, solo e rochas. Eles estão presentes no ar ambiente, nas águas subterrâneas, nas plantas e nos alimentos. O principal uso comercial do sulfato é na indústria química. A concentração de sulfato na água do mar é de cerca de 2.700 miligramas por litro (mg/L). Varia de 3 a 30 mg/L na maioria das fontes de água doce, embora concentrações muito mais altas (1000 mg/L) sejam encontradas em algumas localizações geográficas.
6. Conductivity: A água pura não é um bom condutor de corrente elétrica, mas sim um bom isolante. O aumento na concentração de íons aumenta a condutividade elétrica da água. Geralmente, a quantidade de sólidos dissolvidos na água determina a condutividade elétrica. A condutividade elétrica (EC), na verdade, mede o processo iônico de uma solução que lhe permite transmitir corrente. De acordo com os padrões da WHO, o valor EC não deve exceder 400 µS/cm.

7. Organic_carbon: O carbono orgânico total (TOC) nas águas de origem provém da matéria orgânica natural em decomposição (NOM), bem como de fontes sintéticas. TOC é uma medida da quantidade total de carbono em compostos orgânicos em água pura. De acordo com a EPA dos EUA, < 2 mg/L como TOC em água tratada/potável e < 4 mg/L em água de origem usada para tratamento.
8. Trihalomethanes: THMs são produtos químicos que podem ser encontrados na água tratada com cloro. A concentração de THMs na água potável varia de acordo com o nível de matéria orgânica na água, a quantidade de cloro necessária para tratar a água e a temperatura da água que está sendo tratada. Níveis de THM de até 80 ppm são considerados seguros na água potável.
9. Turbidity: A turbidez da água depende da quantidade de matéria sólida presente no estado suspenso. É uma medida das propriedades de emissão de luz da água e o teste é usado para indicar a qualidade da descarga de resíduos em relação à matéria coloidal. O valor médio de turbidez obtido para Wondo Genet Campus (0,98 NTU) é inferior ao valor recomendado pela OMS de 5,00 NTU.
10. Potability: Indica se a água é segura para consumo humano, onde 1 significa Potável e 0 significa Não potável.

Objetivos Específico:

- Treinar um modelo KNNs;
- Treinar um modelo Random Forests;
- Otimizar hiperparâmetros dos modelos;
- Aplicar métricas de Avaliação de Modelos de Classificação;

Justificativa:

Essas informações ajudam a identificar a qualidade da água em diversos locais, usando, como base, dados já conhecidos e analizados em diversas outra fontes.

Referências:

1. BRUCE, P., & Bruce, A. (2019). Estatística prática para cientistas de dados: 50 conceitos essenciais. Alta Books.
2. DATACAMP. Understanding Logistic Regression in Python. DataCamp, 2023. Disponível em: https://www.datacamp.com/pt/tutorial/understanding-logistic-regression-python. Acesso em: 01 maio 2024.
3. DATASKLR. Select Classification Methods: K-Nearest Neighbors. DataSchool, 2023. Disponível em: https://medium.com/machine-learning-researcher/k-nearest-neighbors-in-

machine-learning-e794014abd2a. Acesso em: 01 maio 2024.

4. FÁVERO, L. P., Lopes E, B., & Prado, P. (2017). Manual de análise de dados: estatística e modelagem multivariada com Excel, SPSS e Stata. Elsevier.
5. KADIWAL, A. (2021). Water Quality [Data set]. In Drinking Water Potability. Acesso em 12 Jan 2024 de https://www.kaggle.com/datasets/adityakadiwal/water-potability/data.
6. KUNUMI. (2020, 10 Junho). Métricas de Avaliação em Machine Learning: Classificação. Kunumi Blog. Acesso em 17 Jan 2024 de https://medium.com/kunumi/m%C3%A9tricas-de-avalia%C3%A7%C3%A3o-em-machine-learning-classifica%C3%A7%C3%A3o-49340dcdb198.
7. MACHINE LEARNING MASTERY. Hyperparameters for Classification Machine Learning Algorithms. Machine Learning Mastery, 2023. Disponível em: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/. Acesso em: 01 maio 2024.
8. MATOS, G. (2023, December 5). K-Nearest Neighbors(KNN): Entendendo o seu funcionamento e o construindo do zero. Share! Por Ateliê de Software. Acesso em 18 Jan 2024 de https://share.atelie.software/k-nearest-neighbors-knn-entendo-o-seu-funcionamento-e-o-construindo-do-zero-a21b022acd6f.
9. SILVA, Jeaps. Quando utilizar StandardScaler, MinMaxScaler e RobustScaler em Machine Learning. Medium, 2018. Disponível em: https://medium.com/@jeapsilva/quando-utilizar-standardscaler-minmaxscaler-e-robustscaler-em-machine-learning-788de8b157ca. Acesso em: 01 maio 2024.
10. SOMOS TERA. Árvores de Decisão. Blog Somos Tera, 2023. Disponível em: https://blog.somostera.com/data-science/arvores-de-decisao. Acesso em: 01 maio 2024.
11. SKLEARN DEVELOPERS. sklearn.neighbors.KNeighborsClassifier. scikit-learn, 2023. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. Acesso em: 01 maio 2024.
12. TARAKANOVA, Sasha. KNN with Hyperparameter Tuning Using Optuna. Kaggle, 2021. Disponível em: https://www.kaggle.com/code/sashatarakanova/knn-with-hyperparameter-tuning-using-optuna. Acesso em: 01 maio 2024.

SUMÁRIO

6. Modelagem: Random Forest
7. Testando outros Modelos
8. Resultados
9. Conclusão

1. Instalação e Carregamento das Bibliotecas

```
# Importando libs básicas
import numpy as np
import pandas as pd

# Importando libs de Visualização de Dados
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Importando metodos comuns para os modelos
from sklearn.model_selection import train_test_split
```

```
# Importando métricas para validação do modelo
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score,
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Carregando **modelos**

```
# Modelo KNN
from sklearn.neighbors import KNeighborsClassifier
```

```
# Modelo Árvore de Decisão
from sklearn.tree import DecisionTreeClassifier
```

```
# Modelo Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
```

```
# Carregando o GridSearchCV
from sklearn.model_selection import GridSearchCV
```

2. EDA - Dataset

```
# Carrega dados
df = pd.read_csv('https://raw.githubusercontent.com/aluipio/ds_ada_santander_ml_i/main/
print('Dimensão dos dataset')
print(f'Linhas: {df.shape[0]}', f'- Colunas: {df.shape[1]}')
```

```
Dimensão dos dataset
Linhas: 3276 - Colunas: 10
```

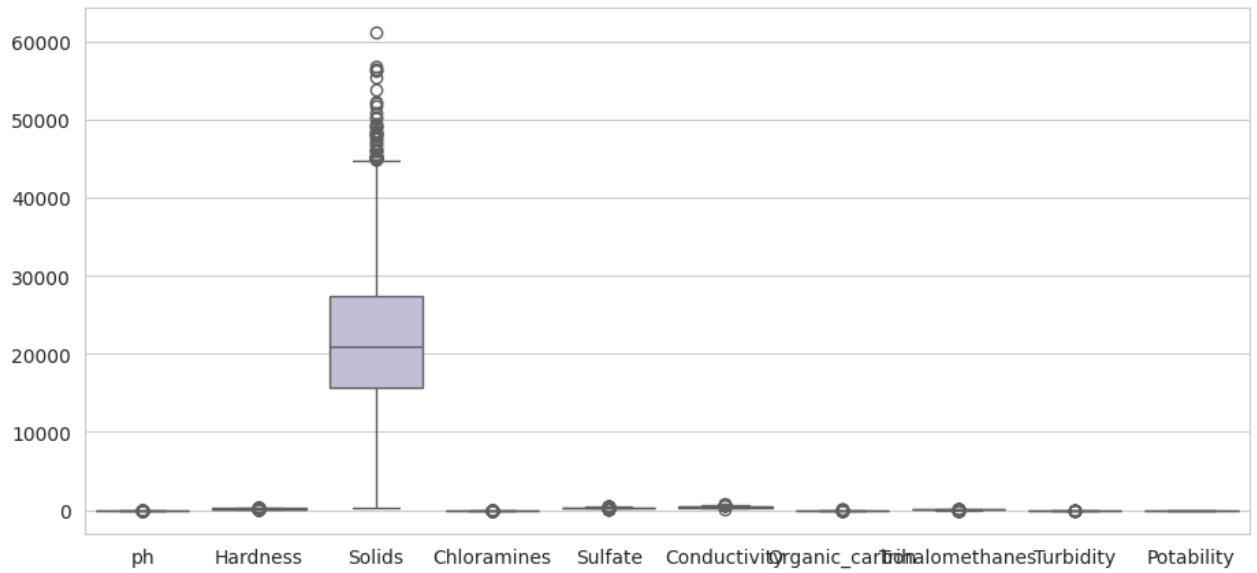
```
# Visualizando Dados
df.head()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.

```
# Visualizando a tipagem dos dados e a existência de dados nulos
df.info()
```

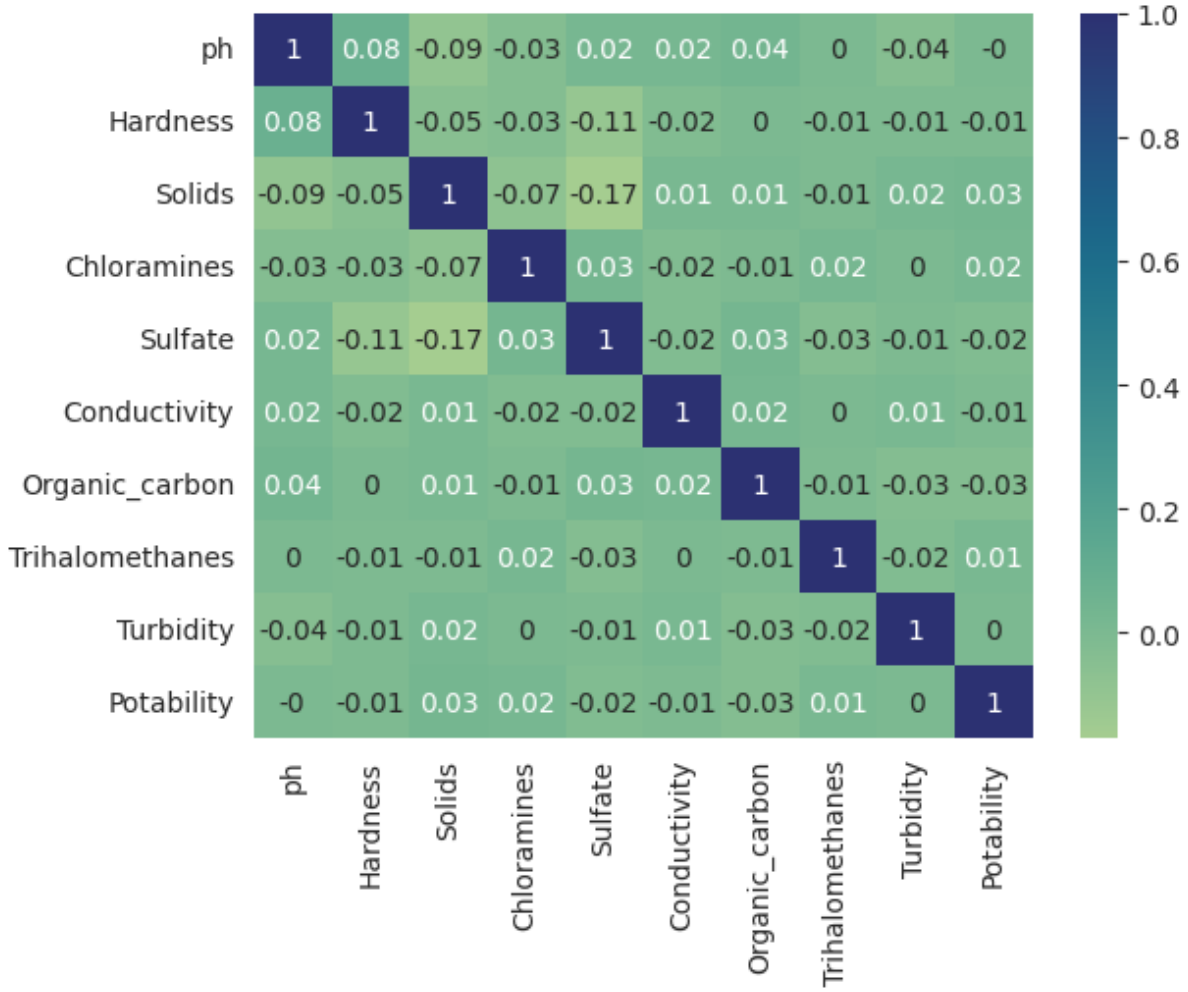
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

```
# Plotando gráfico boxplot
plt.figure(figsize =(11, 5))
sns.set_style("whitegrid")
ax = sns.boxplot(data=df, linewidth=1, palette = "Set3")
```



NOTA: Olhando as features através de um gráfico BoxPlot, pode-se observar que existem alguns dados que podem ser considerados outliers, contudo tratm-se de aferições reais de unidades de registro.

```
# Análise de correlação
corr = df.corr(method='pearson').round(2)

# Gráfico de Correlação
sns.heatmap(corr, annot=True, cmap="crest");
```



O gráfico acima, mostra a correlação de forma gráfica, onde quanto mais escuro o quadro, maior é a correlação entre as variáveis.

Quanto ao Coeficiente de correlação, considerando que:

- 0.9 para mais ou para menos indica uma correlação muito forte.
- 0.7 a 0.9 positivo ou negativo indica uma correlação forte.
- 0.5 a 0.7 positivo ou negativo indica uma correlação moderada.
- 0.3 a 0.5 positivo ou negativo indica uma correlação fraca.
- 0 a 0.3 positivo ou negativo indica uma correlação desprezível.

Considerando que todas estão entre -0.3 e 0.3, Pode-se observar que **NÃO existe uma CORRELAÇÃO POSITIVA ou NEGATIVA entre as features.** Sendo uma correlação desprezível.

```
# Quantificando os dados nulos
df.isna().sum()
```

```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

NOTA: A observação desses dados nulos gera um alerta importante para que seja realizado o devido tratamento dos dados após a divisão dos dados de Treino e dados de Teste.

```
# Verificando como está balanceado nossa variável de interesse.
df.Potability.value_counts()
```

```
Potability
0    1998
1    1278
Name: count, dtype: int64
```

```
# Distribuição da nossa variável de interesse
df.Potability.value_counts(normalize=True)
```

```
Potability
0    0.60989
1    0.39011
Name: proportion, dtype: float64
```

NOTA: Pela distribuições apresentada, pode-se considerar que os dados, conforme variável de interesse, estão balanceadas.

3. Preparação e Pré-processamento dos dados

```
# Separa os dados
X = df.drop(columns='Potability')
y = df.Potability
```

```
# Separação com estratificação dos dados conforme a feature em y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, stratify
```

Observação: Considerando que existem dados nulos no dataset, precisaremos trata essas features, antes de processar os modelos.

```
df.describe()
```

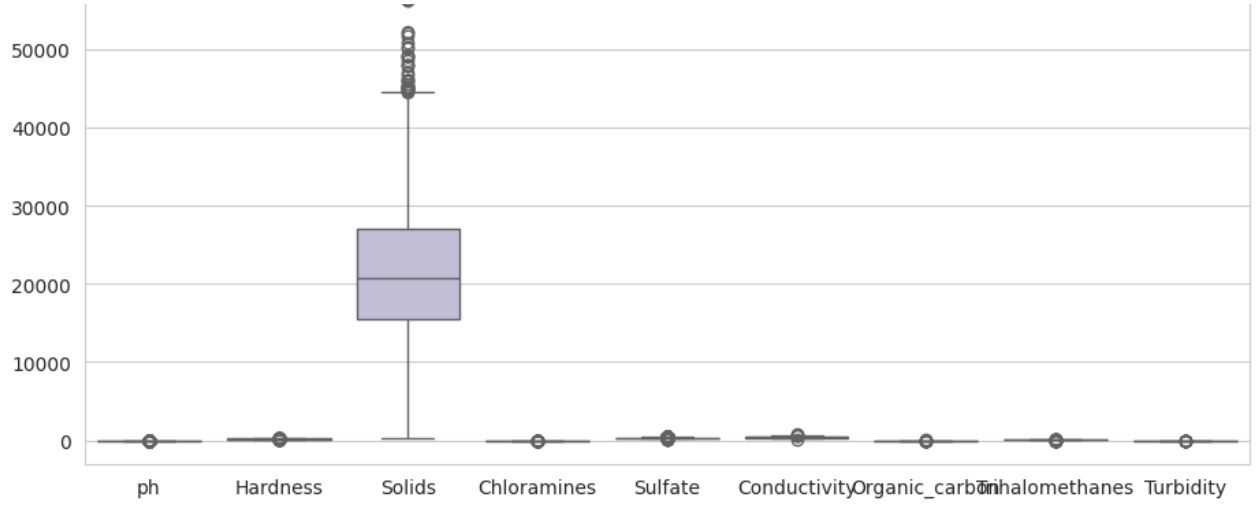
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620

NOTA: Considerando que a Média e Mediana estão muito próximas, podemos imputar a Média para o missing values.

Observação: desprezamos a possibilidade remover os registros com dados nulos, tendo em vista o pequeno número de registros em nosso dataset.

```
# Utilizaremos
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X_train)

X_train_imputer = pd.DataFrame(imputer.transform(X_train), columns=X_train.columns)
X_test_imputer = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)
```

```
# Plotando gráfico boxplot
plt.figure(figsize =(11, 5))
sns.set_style("whitegrid")
ax = sns.boxplot(data=X_train_imputer, linewidth=1, palette = "Set3")
```

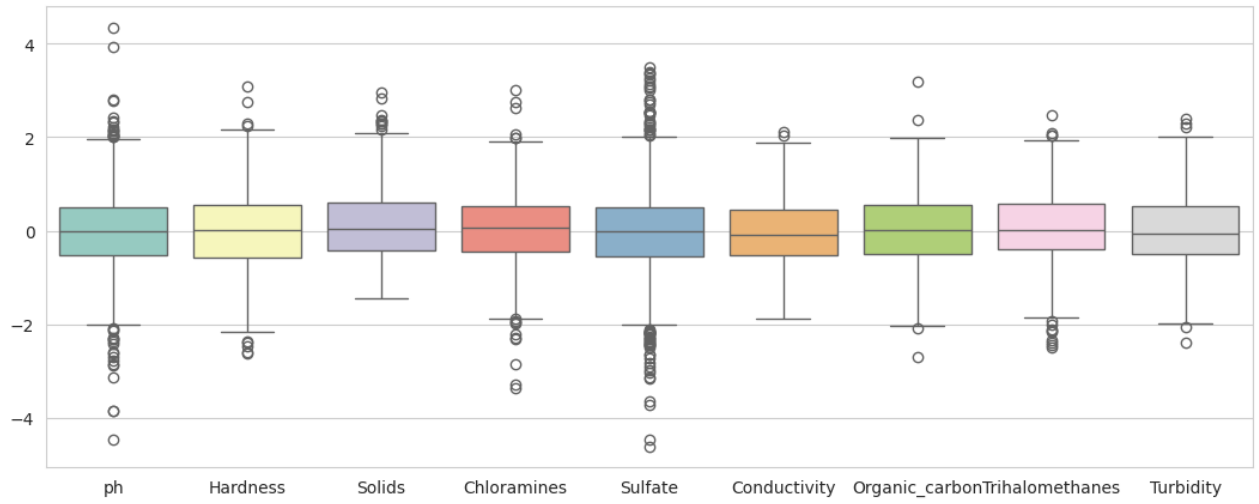


- **StandardScaler**: “padroniza as features removendo a média e a escala a uma variância a uma unidade”. Em outras palavras, isso significa que para cada feature, a média seria 0, e o Desvio Padrão seria 1. Ótimos em algoritmos como Linear Regression e Logistic Regression. Utilizar Quando estão com uma distribuição normal ou quando se deseja uma distribuição mais próxima da normal.
- **MinMaxScaler**: “dimensiona o conjunto de dados de modo que todos os valores de recursos estejam no intervalo [0, 1] ou no intervalo [-1,1] se houver valores negativos nos dados. Esta escala também comprime todos os inliers na faixa [0,0.005]”. É importante ressaltar que essa técnica funciona melhor se a distribuição dos dados não for normal e se o desvio padrão for pequeno, além disso, o MinMaxScaler não reduz de forma eficaz o impacto de outliers e também preserva a distribuição original. Quando se deseja reduzri o impacto dos outliers.
- **RobustScalcer**: é baseada em percentis. Devido a isso o diferencial desse método é o tratamento dos outliers. Porém, é importante notar que os outliers ainda estarão presentes nos dados reescalados, mas o seu impacto negativo será reduzido. Quando a distribuição dos seus dados não for normal.

Considerando que as features estão em escalas diferentes, que existem dados outliers no dataset, e buscando minimizar o impacto negativo deles na modelagem, optou-se, a princípio, usar o RobustScaler().

```
1 from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
2
3 # scaler = StandardScaler()
```

```
# scaler = StandardScaler()
# scaler = MinMaxScaler()
scaler = RobustScaler()

scaler.fit(X_train_imputer)

X_train_proc = pd.DataFrame(scaler.transform(X_train_imputer), columns=X_train_imputer.
X_test_proc = pd.DataFrame(scaler.transform(X_test_imputer), columns=X_test_imputer.col

# Plotando gráfico boxplot
plt.figure(figsize =(13, 5))
sns.set_style("whitegrid")
ax = sns.boxplot(data=X_test_proc, linewidth=1, palette = "Set3")
```



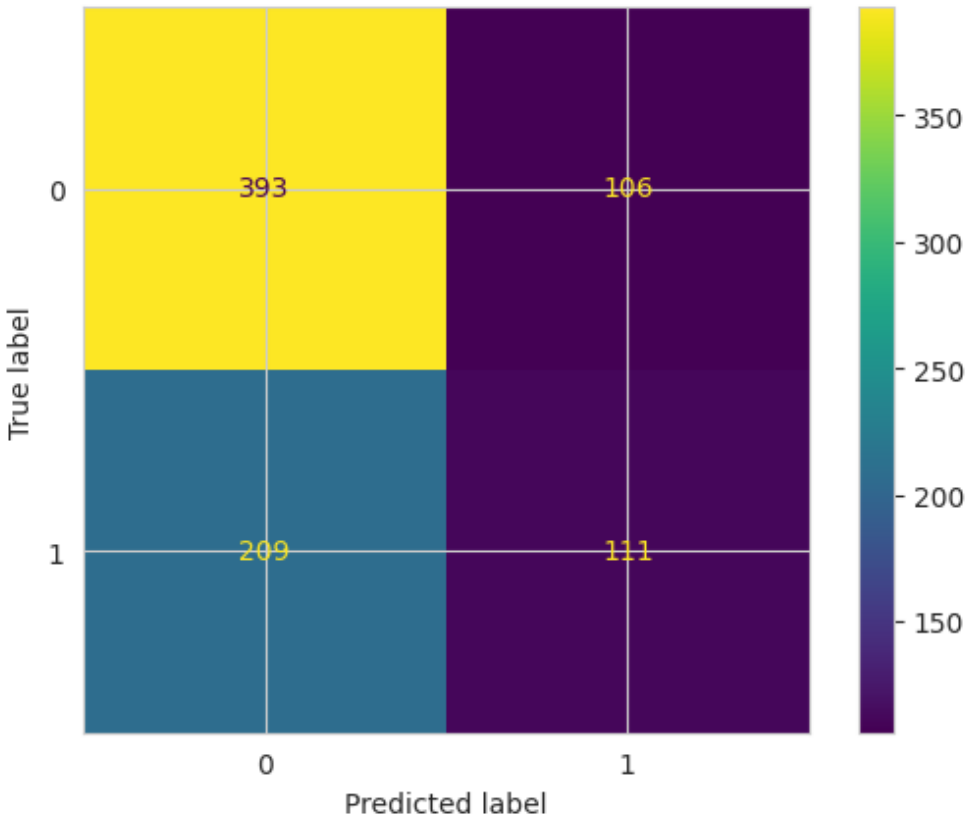

```
# Armazenar o resultado dos modelos
RESULTADOS = {}
```

4. Modelagem: K—Nearest Neighbors (KNN)

O K—Nearest Neighbors (KNN) é um dos muitos algoritmos de aprendizagem supervisionada usado no campo de data mining e machine learning, ele é um classificador onde o aprendizado é baseado “no quão similar” é um dado do outro. O treinamento é formado por vetores de n dimensões. (José, 2018)

```
# Definindo o valor de vizinhos
classifier = KNeighborsClassifier(n_neighbors=5)

# Treinando o modelo, com dados de treinamento
```

```
classifier.fit(X_train_proc, y_train)

#### Prevendo os valores de Y para os dados de teste (X_test)
y_test_pred = classifier.predict(X_test_proc)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```




```
print(classification_report(y_test, y_test_pred))

y_train_pred = classifier.predict(X_train_proc)
y_test_pred = classifier.predict(X_test_proc)

roc_auc_train = roc_auc_score(y_train, y_train_pred)
roc_auc_test = roc_auc_score(y_test, y_test_pred)

print(f"AUC_train = {roc_auc_train}, AUC_test = {roc_auc_test}")
```

```
              precision    recall  f1-score   support

           0       0.65      0.79      0.71       499
           1       0.51      0.35      0.41       320

    accuracy                           0.62       819
   macro avg       0.58      0.57      0.56       819
weighted avg       0.60      0.62      0.60       819

AUC_train = 0.7340916909115471, AUC_test = 0.5672250751503006
```

NOTA: Considerando no primeiro treinamento, que pela curva AUC_train o modelo sofreu overfiting, estando mais ajustado aos dados de treinamento e não se adequando aos dados de

overfitting, estando mais ajustado aos dados de treinamento e não se adequando aos dados de teste. Será preciso testar outras possibilidades, que são inumeras, até encontrar os melhores parametros para o nosso modelo. Usaremos o GridSearchCV para encontrar esse valores ótimos.

O GridSearchCV testará as possibilidades estabelecidas para o modelo em questão, conforme parametros apresentados.

Ele apresentará ajudará a encontrar os melhores valores para otimizar os hiperparametros.

```
# Listando os scores disponíveis para usar como metricas no modelo.
import sklearn
sklearn.metrics.get_scorer_names()
```

```
['accuracy',
 'adjusted_mutual_info_score',
 'adjusted_rand_score',
 'average_precision',
 'balanced_accuracy',
 'completeness_score',
 'explained_variance',
 'f1',
 'f1_macro',
 'f1_micro',
 'f1_samples',
 'f1_weighted',
 'fowlkes_mallows_score',
 'homogeneity_score',
 'jaccard',
 'jaccard_macro',
 'jaccard_micro',
 'jaccard_samples',
 'jaccard_weighted',
 'matthews_corrcoef',
 'max_error',
 'mutual_info_score',
 'neg_brier_score',
 'neg_log_loss',
 'neg_mean_absolute_error',
 'neg_mean_absolute_percentage_error',
 'neg_mean_gamma_deviance',
 'neg_mean_poisson_deviance',
 'neg_mean_squared_error',
 'neg_mean_squared_log_error',
 'neg_median_absolute_error',
 'neg_negative_likelihood_ratio',
 'neg_root_mean_squared_error',
 'normalized_mutual_info_score',
 'positive_likelihood_ratio',
 'precision',
 'precision_macro',
 'precision_micro',
 'precision_samples',
 'precision_weighted',
 'r2',
 'rand_score',
 'recall',
```

```
    'recall_macro',
    'recall_micro',
    'recall_samples',
    'recall_weighted',
    'roc_auc',
    'roc_auc_ovo',
    'roc_auc_ovo_weighted',
    'roc_auc_ovr',
    'roc_auc_ovr_weighted',
    'top_k_accuracy',
    'v_measure_score']
```

Será utilizado o Accuracy, uma vez que buscará o modelo com maior precisão (precision) e sensibilidade (recall), simultaneamente.

```
# Carregando a biblioteca GridSearchCV
from sklearn.model_selection import GridSearchCV

# Criando modelo
classifier = KNeighborsClassifier(algorithm='auto', p=1)

# Definindo parâmetros
relacao_parametros = {
    'leaf_size': (1, 3, 6, 9),
    'metric': ('minkowski', 'euclidean'),
    'n_neighbors': (10, 20, 25, 30, 40),
    'weights': ('uniform', 'distance')
}

# Aplicando o algoritmo com parâmetros definidos anteriormente
clf = GridSearchCV(
    estimator = classifier,
    param_grid = relacao_parametros,
    scoring = 'accuracy',
    n_jobs = -1,
    cv = 5,
    return_train_score = True
)

# Treinando o modelo
search = clf.fit(X_train_proc, y_train)

# Capturando os resultados e os índices dos melhores parâmetros
results_GridSearchCV = search.cv_results_
indice_melhores_parametros = search.best_index_

# Apresentando a média de score de treino e teste produzida
print(f"mean_train_score {results_GridSearchCV['mean_train_score'][indice_melhores_pa
print(f"mean_test_score {results_GridSearchCV['mean_test_score'][indice_melhores_para

# Apresentação dos parâmetros
print(search.best_params_ )
print('Best Score - KNN:', search.best_score_ )

```

```
# Resultado do 1º ciclo
# ----------------------
# relacao_parametros = {
#     'leaf_size': (1, 10, 20, 30, 40),
#     'metric': ('minkowski', 'euclidean'),
#     'n_neighbors': (40, 50, 60, 70, 80, 90),
#     'weights': ('uniform', 'distance')
# }
# ----------------------
# mean_train_score 1.00
# mean_test_score 0.66
# {'leaf_size': 1, 'metric': 'euclidean', 'n_neighbors': 40, 'weights': 'distance'}
# Best Score - KNN: 0.6597370556190287


# Resultado do 2º ciclo (Regulado GridSearch para os valores em torno do 1º ciclo)
# ----------------------
# relacao_parametros = {
#     'leaf_size': (1, 3, 6, 9),
#     'metric': ('minkowski', 'euclidean'),
#     'n_neighbors': (10, 20, 25, 30, 40),
#     'weights': ('uniform', 'distance')
# }
# ----------------------
# mean_train_score 1.00
# mean_test_score 0.66
# {'leaf_size': 1, 'metric': 'euclidean', 'n_neighbors': 20, 'weights': 'distance'}
# Best Score - KNN: 0.66055006374911

# Modelo reajustado para um segundo ciclo
```

```
mean_train_score 1.00
mean_test_score 0.66
{'leaf_size': 1, 'metric': 'euclidean', 'n_neighbors': 20, 'weights': 'distance'}
Best Score - KNN: 0.66055006374911
```

```
print('BEST MODEL - K-NEAREST NEIGHBORS ')

y_train_pred = search.predict(X_train_proc)
y_test_pred = search.predict(X_test_proc)

roc_auc_train = roc_auc_score(y_train, y_train_pred)
roc_auc_test = roc_auc_score(y_test, y_test_pred)

print(classification_report(y_test, y_test_pred))
print(f"Accuracy = {accuracy_score(y_test, y_test_pred)}")
print(f"AUC_train = {roc_auc_train}, AUC_test = {roc_auc_test}")
```

```
BEST MODEL - K-NEAREST NEIGHBORS
              precision    recall  f1-score   support

           0       0.65      0.89      0.75       499
           1       0.60      0.25      0.36       320

    accuracy                           0.64       819
   macro avg       0.63      0.57      0.55       819
weighted avg       0.63      0.64      0.60       819
```

```
Accuracy = 0.6422466422466423
AUC_train = 1.0, AUC_test = 0.5724542835671343
```

NOTA: Observa-se um grande Overfiting. Buscando encontrar o melhores resultados, será testado o OPTUNA

```
# Instalar o Optuna
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-3.6.1-py3-none-any.whl (380 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 380.1/380.1 kB 5.4 MB/s eta 0:00:00
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.13.1-py3-none-any.whl (233 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 233.4/233.4 kB 11.5 MB/s eta 0:00:00
Collecting colorlog (from optuna)
  Downloading colorlog-6.8.2-py3-none-any.whl (11 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: sqlalchemy>=1.3.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (fr
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.3-py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.8/78.8 kB 8.6 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dis
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-p
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.3 alembic-1.13.1 colorlog-6.8.2 optuna-3.6.1
```

```
import optuna
from optuna.samplers import TPESampler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
```

```
def objective(trial):
    # -- Instanciar o Scaler
    scalers = trial.suggest_categorical("scalers", ['minmax', 'standard', 'robust'])

    if scalers == "minmax":
        scaler = MinMaxScaler()
    elif scalers == "standard":
        scaler = StandardScaler()
    else:
        scaler = RobustScaler()

    # -- Tune estimator algorithm
    leaf_size = trial.suggest_int("leaf_size", 1, 10)
    n_neighbors = trial.suggest_int("n_neighbors", 10, 50)
    weights = trial.suggest_categorical("weights", ['uniform', 'distance'])
    metric = trial.suggest_categorical("metric", ['euclidean', 'manhattan', 'minkowsk

```

```
    cls_knn = KNeighborsClassifier(n_neighbors=n_neighbors, weights=weights, metric=m

    # -- Make a pipeline
    pipeline = make_pipeline(scaler, cls_knn)

    # -- Cross-validate the features reduced by dimensionality reduction methods
    kfold = StratifiedKFold(n_splits=10)
    score = cross_val_score(pipeline, X_train_imputer, y_train, scoring='accuracy', c
    score = score.mean()
    return score

sampler = TPESampler(seed=42)
study = optuna.create_study(direction="maximize", sampler=sampler)
study.optimize(objective, n_trials=300)
```

```
[I 2024-05-01 17:24:33,239] A new study created in memory with name: no-name-3591527
[I 2024-05-01 17:24:34,854] Trial 0 finished with value: 0.6438609590177534 and para
[I 2024-05-01 17:24:37,039] Trial 1 finished with value: 0.6446640119462419 and para
[I 2024-05-01 17:24:39,557] Trial 2 finished with value: 0.6357424921187987 and para
[I 2024-05-01 17:24:40,092] Trial 3 finished with value: 0.6332951717272275 and para
[I 2024-05-01 17:24:40,605] Trial 4 finished with value: 0.6357474697195952 and para
[I 2024-05-01 17:24:41,080] Trial 5 finished with value: 0.6381765389082463 and para
[I 2024-05-01 17:24:41,913] Trial 6 finished with value: 0.6597312095569935 and para
[I 2024-05-01 17:24:42,546] Trial 7 finished with value: 0.6312626514020242 and para
[I 2024-05-01 17:24:44,672] Trial 8 finished with value: 0.6324838227974118 and para
[I 2024-05-01 17:24:45,178] Trial 9 finished with value: 0.6658586361373817 and para
[I 2024-05-01 17:24:45,694] Trial 10 finished with value: 0.6601592832254852 and par
[I 2024-05-01 17:24:46,225] Trial 11 finished with value: 0.6601592832254852 and par
[I 2024-05-01 17:24:46,752] Trial 12 finished with value: 0.6658586361373817 and par
[I 2024-05-01 17:24:47,269] Trial 13 finished with value: 0.6646275095404015 and par
[I 2024-05-01 17:24:47,765] Trial 14 finished with value: 0.6605558320889331 and par
[I 2024-05-01 17:24:48,282] Trial 15 finished with value: 0.6646275095404015 and par
[I 2024-05-01 17:24:48,721] Trial 16 finished with value: 0.6519844035175046 and par
[I 2024-05-01 17:24:49,820] Trial 17 finished with value: 0.6605574912891987 and par
[I 2024-05-01 17:24:50,344] Trial 18 finished with value: 0.6613688402190144 and par
[I 2024-05-01 17:24:50,886] Trial 19 finished with value: 0.660565787290526 and para
[I 2024-05-01 17:24:51,664] Trial 20 finished with value: 0.6536436037829766 and par
[I 2024-05-01 17:24:52,180] Trial 21 finished with value: 0.6605608096897295 and par
[I 2024-05-01 17:24:52,710] Trial 22 finished with value: 0.6613771362203418 and par
[I 2024-05-01 17:24:53,216] Trial 23 finished with value: 0.6625900116144019 and par
[I 2024-05-01 17:24:53,743] Trial 24 finished with value: 0.6658586361373817 and par
[I 2024-05-01 17:24:54,236] Trial 25 finished with value: 0.6617786626845861 and par
[I 2024-05-01 17:24:54,805] Trial 26 finished with value: 0.6617770034843206 and par
[I 2024-05-01 17:24:55,536] Trial 27 finished with value: 0.6593330014932801 and par
[I 2024-05-01 17:24:56,290] Trial 28 finished with value: 0.6613771362203418 and par
[I 2024-05-01 17:24:57,146] Trial 29 finished with value: 0.6556844201095073 and par
[I 2024-05-01 17:24:57,695] Trial 30 finished with value: 0.656084287373486 and para
[I 2024-05-01 17:24:58,211] Trial 31 finished with value: 0.6642276422764227 and par
[I 2024-05-01 17:24:58,741] Trial 32 finished with value: 0.6593462750954041 and par
[I 2024-05-01 17:24:59,249] Trial 33 finished with value: 0.6613771362203418 and par
[I 2024-05-01 17:24:59,789] Trial 34 finished with value: 0.6605608096897295 and par
[I 2024-05-01 17:25:00,467] Trial 35 finished with value: 0.6390011614401858 and par
[I 2024-05-01 17:25:00,864] Trial 36 finished with value: 0.6536485813837729 and par
[I 2024-05-01 17:25:01,729] Trial 37 finished with value: 0.6613738178198107 and par
[I 2024-05-01 17:25:02,632] Trial 38 finished with value: 0.6328886676621868 and par
[I 2024-05-01 17:25:03,287] Trial 39 finished with value: 0.6593479342956694 and par
[I 2024-05-01 17:25:03,651] Trial 40 finished with value: 0.6556611913057906 and par
[I 2024-05-01 17:25:04,779] Trial 41 finished with value: 0.6646275095404015 and par
[I 2024-05-01 17:25:05,301] Trial 42 finished with value: 0.6646275095404015 and par
```

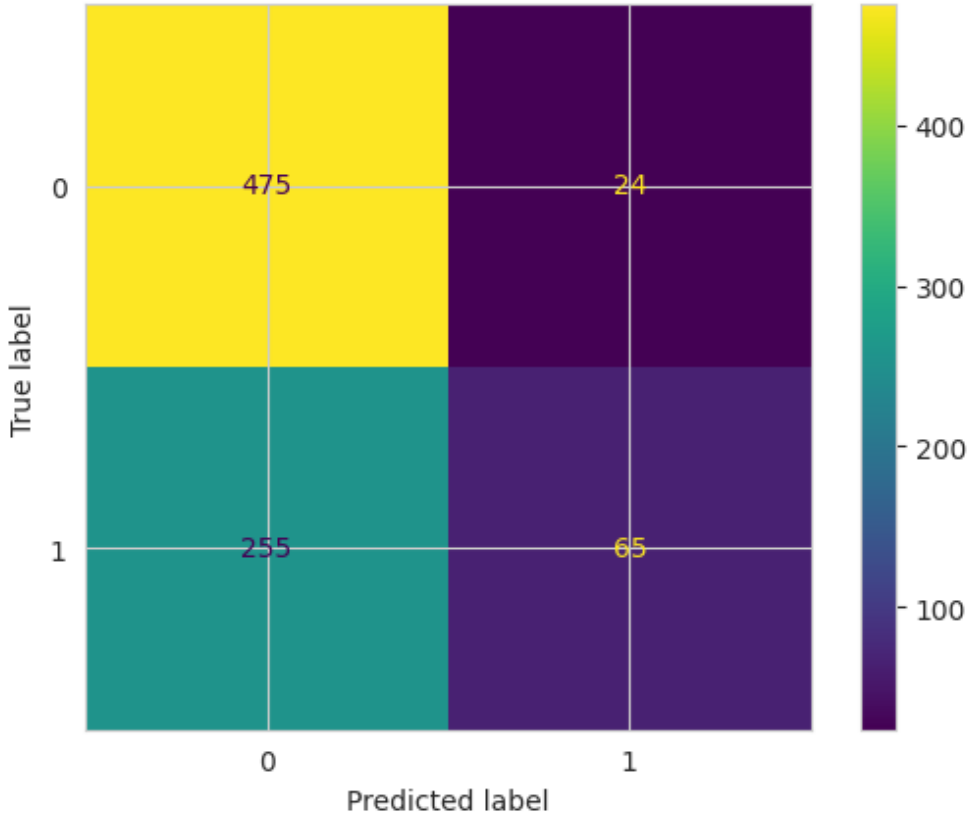
```
[I 2024-05-01 17:25:05,815] Trial 43 finished with value: 0.6642276422764227 and par
[I 2024-05-01 17:25:06,245] Trial 44 finished with value: 0.6345146839223494 and par
[I 2024-05-01 17:25:06,759] Trial 45 finished with value: 0.6605608096897295 and par
[I 2024-05-01 17:25:07,442] Trial 46 finished with value: 0.6609689729550356 and par
[I 2024-05-01 17:25:08,344] Trial 47 finished with value: 0.6638161606105857 and par
[I 2024-05-01 17:25:08,856] Trial 48 finished with value: 0.6471096731375477 and par
[I 2024-05-01 17:25:09,474] Trial 49 finished with value: 0.6519844035175046 and par
[I 2024-05-01 17:25:10,067] Trial 50 finished with value: 0.6328870084619214 and par
[I 2024-05-01 17:25:10,576] Trial 51 finished with value: 0.6625883524141364 and par
[I 2024-05-01 17:25:11,092] Trial 52 finished with value: 0.6646275095404015 and par
[I 2024-05-01 17:25:11,487] Trial 53 finished with value: 0.6658586361373817 and par
[I 2024-05-01 17:25:11,882] Trial 54 finished with value: 0.6658586361373817 and par
[I 2024-05-01 17:25:12,293] Trial 55 finished with value: 0.6585299485647917 and par
[I 2024-05-01 17:25:12,673] Trial 56 finished with value: 0.6658586361373817 and par
```

NOTA: pelo resultado do Optuna, pode-se observar que o melhor resultado: Trial 9 finished with value: 0.6658586361373817 and parameters:

```
{
    'scalers': 'robust',
    'leaf_size': 6,
    'n_neighbors': 41,
    'weights': 'distance',
    'metric': 'euclidean'
}
```

Best is trial 9 with value: 0.6658586361373817.

```
# Processando Dados
scaler = RobustScaler()

scaler.fit(X_train_imputer)
X_train_proc = pd.DataFrame(scaler.transform(X_train_imputer), columns=X_train_impute
X_test_proc = pd.DataFrame(scaler.transform(X_test_imputer), columns=X_test_imputer.c

# Testando modelos
classifier = KNeighborsClassifier(
    leaf_size= 6,
    n_neighbors= 41,
    weights= 'distance',
    metric= 'euclidean'
)

# Treinando o modelo, com dados de treinamento
classifier.fit(X_train_proc, y_train)

#### Prevendo os valores de Y para os dados de teste (X_test)
y_test_pred = classifier.predict(X_test_proc)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```




```
# Verificando os resultados
y_train_pred = classifier.predict(X_train_proc)
y_test_pred = classifier.predict(X_test_proc)

roc_auc_train = roc_auc_score(y_train, y_train_pred)
roc_auc_test = roc_auc_score(y_test, y_test_pred)

print(classification_report(y_test, y_test_pred))
print(f"Accuracy = {accuracy_score(y_test, y_test_pred)}.")
print(f"AUC_train = {roc_auc_train}, AUC_test = {roc_auc_test}.")

RESULTADOS['KNN'] = [classifier, accuracy_score(y_test, y_test_pred), roc_auc_train,
```

```
              precision    recall  f1-score   support

           0       0.65      0.95      0.77       499
           1       0.73      0.20      0.32       320

    accuracy                           0.66       819
   macro avg       0.69      0.58      0.55       819
weighted avg       0.68      0.66      0.60       819

Accuracy = 0.6593406593406593.
AUC_train = 1.0, AUC_test = 0.5775144038076152.
```

5. Modelagem: Árvore de Decisão

Uma Árvore de Decisão é um algoritmo de aprendizado de máquina supervisionado que é utilizado para classificação e para regressão. Isto é, pode ser usado para prever categorias discretas (sim ou não, por exemplo) e para prever valores numéricos (o valor do lucro em reais)

discretas (sim ou não, por exemplo) e para prever valores numéricos (o valor do lucro em reais).

Será utilizado o optuna para testar os dados.

```
def objective_dtree(trial):
    # -- Tune estimator algorithm
    max_depth = trial.suggest_int("max_depth", 1, 20)
    min_samples_split = trial.suggest_categorical("min_samples_split", [16, 32, 64, 1
    min_samples_leaf = trial.suggest_categorical("min_samples_leaf", [4, 8, 16, 32])
    criterion = trial.suggest_categorical("criterion", ['gini', 'entropy'])

    cls_knn = DecisionTreeClassifier(
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        criterion=criterion
    )

    # -- Make a pipeline
    pipeline = make_pipeline(cls_knn)

    # -- Cross-validate the features reduced by dimensionality reduction methods
    kfold = StratifiedKFold(n_splits=10)
    score = cross_val_score(pipeline, X_train_imputer, y_train, scoring='accuracy', c
    score = score.mean()
    return score

sampler = TPESampler(seed=42)
study = optuna.create_study(direction="maximize", sampler=sampler)
study.optimize(objective_dtree, n_trials=200)
```

```
[I 2024-05-01 17:48:00,458] A new study created in memory with name: no-name-3e582e7
[I 2024-05-01 17:48:00,787] Trial 0 finished with value: 0.6304546208727393 and para
[I 2024-05-01 17:48:01,142] Trial 1 finished with value: 0.6113190642110504 and para
[I 2024-05-01 17:48:01,401] Trial 2 finished with value: 0.6275858636137382 and para
[I 2024-05-01 17:48:01,702] Trial 3 finished with value: 0.6377750124440019 and para
[I 2024-05-01 17:48:02,089] Trial 4 finished with value: 0.6300530944084951 and para
[I 2024-05-01 17:48:02,342] Trial 5 finished with value: 0.6341148166583708 and para
[I 2024-05-01 17:48:02,428] Trial 6 finished with value: 0.6125452132072342 and para
[I 2024-05-01 17:48:02,650] Trial 7 finished with value: 0.6320623859299818 and para
[I 2024-05-01 17:48:02,974] Trial 8 finished with value: 0.6166135722581715 and para
[I 2024-05-01 17:48:03,167] Trial 9 finished with value: 0.627181018748963 and param
[I 2024-05-01 17:48:03,626] Trial 10 finished with value: 0.638991206238593 and para
[I 2024-05-01 17:48:04,100] Trial 11 finished with value: 0.636147336983574 and para
[I 2024-05-01 17:48:04,539] Trial 12 finished with value: 0.6414468226314917 and par
[I 2024-05-01 17:48:04,982] Trial 13 finished with value: 0.6369620043139207 and par
[I 2024-05-01 17:48:05,473] Trial 14 finished with value: 0.6300447984071678 and par
[I 2024-05-01 17:48:05,911] Trial 15 finished with value: 0.6414468226314917 and par
[I 2024-05-01 17:48:06,357] Trial 16 finished with value: 0.6414468226314917 and par
[I 2024-05-01 17:48:06,780] Trial 17 finished with value: 0.6410353409656545 and par
[I 2024-05-01 17:48:07,260] Trial 18 finished with value: 0.6345246391239422 and par
[I 2024-05-01 17:48:07,701] Trial 19 finished with value: 0.6410353409656545 and par
[I 2024-05-01 17:48:08,043] Trial 20 finished with value: 0.6174182843869255 and par
[I 2024-05-01 17:48:08,487] Trial 21 finished with value: 0.6414468226314917 and par
[I 2024-05-01 17:48:08,902] Trial 22 finished with value: 0.6410353409656545 and par
[I 2024-05-01 17:48:09,348] Trial 23 finished with value: 0.6369620043139207 and par
[I 2024-05-01 17:48:09,749] Trial 24 finished with value: 0.6491869918699187 and par
```

```
[I 2024-05-01 17:48:10,118] Trial 25 finished with value: 0.6202770864443339 and par
[I 2024-05-01 17:48:10,537] Trial 26 finished with value: 0.6373552347768375 and par
[I 2024-05-01 17:48:11,115] Trial 27 finished with value: 0.6491869918699187 and par
[I 2024-05-01 17:48:11,460] Trial 28 finished with value: 0.643051269288203 and para
[I 2024-05-01 17:48:11,721] Trial 29 finished with value: 0.6255467064874731 and par
[I 2024-05-01 17:48:12,164] Trial 30 finished with value: 0.6243487638958022 and par
[I 2024-05-01 17:48:12,665] Trial 31 finished with value: 0.6214932802389248 and par
[I 2024-05-01 17:48:12,904] Trial 32 finished with value: 0.643051269288203 and para
[I 2024-05-01 17:48:13,082] Trial 33 finished with value: 0.6182296333167413 and par
[I 2024-05-01 17:48:13,349] Trial 34 finished with value: 0.6422299651567944 and par
[I 2024-05-01 17:48:13,566] Trial 35 finished with value: 0.6353210552513688 and par
[I 2024-05-01 17:48:13,836] Trial 36 finished with value: 0.6288252862120458 and par
[I 2024-05-01 17:48:13,948] Trial 37 finished with value: 0.6125452132072342 and par
[I 2024-05-01 17:48:14,194] Trial 38 finished with value: 0.6341148166583708 and par
[I 2024-05-01 17:48:14,478] Trial 39 finished with value: 0.6389829102372657 and par
[I 2024-05-01 17:48:14,626] Trial 40 finished with value: 0.6202687904430063 and par
[I 2024-05-01 17:48:14,893] Trial 41 finished with value: 0.6422299651567944 and par
[I 2024-05-01 17:48:15,134] Trial 42 finished with value: 0.643051269288203 and para
[I 2024-05-01 17:48:15,357] Trial 43 finished with value: 0.6353210552513688 and par
[I 2024-05-01 17:48:15,650] Trial 44 finished with value: 0.640210718433715 and para
[I 2024-05-01 17:48:15,893] Trial 45 finished with value: 0.6410121121619379 and par
[I 2024-05-01 17:48:16,384] Trial 46 finished with value: 0.6385913389746142 and par
[I 2024-05-01 17:48:16,924] Trial 47 finished with value: 0.6166119130579061 and par
[I 2024-05-01 17:48:17,264] Trial 48 finished with value: 0.643051269288203 and para
[I 2024-05-01 17:48:17,605] Trial 49 finished with value: 0.6361456777833083 and par
[I 2024-05-01 17:48:17,833] Trial 50 finished with value: 0.6259598473535755 and par
[I 2024-05-01 17:48:18,193] Trial 51 finished with value: 0.643051269288203 and para
[I 2024-05-01 17:48:18,457] Trial 52 finished with value: 0.6353210552513688 and par
[I 2024-05-01 17:48:18,768] Trial 53 finished with value: 0.6402223328355732 and par
[I 2024-05-01 17:48:19,105] Trial 54 finished with value: 0.6308478513356561 and par
[I 2024-05-01 17:48:19,289] Trial 55 finished with value: 0.6255467064874731 and par
[I 2024-05-01 17:48:19,537] Trial 56 finished with value: 0.6377683756429401 and par
```

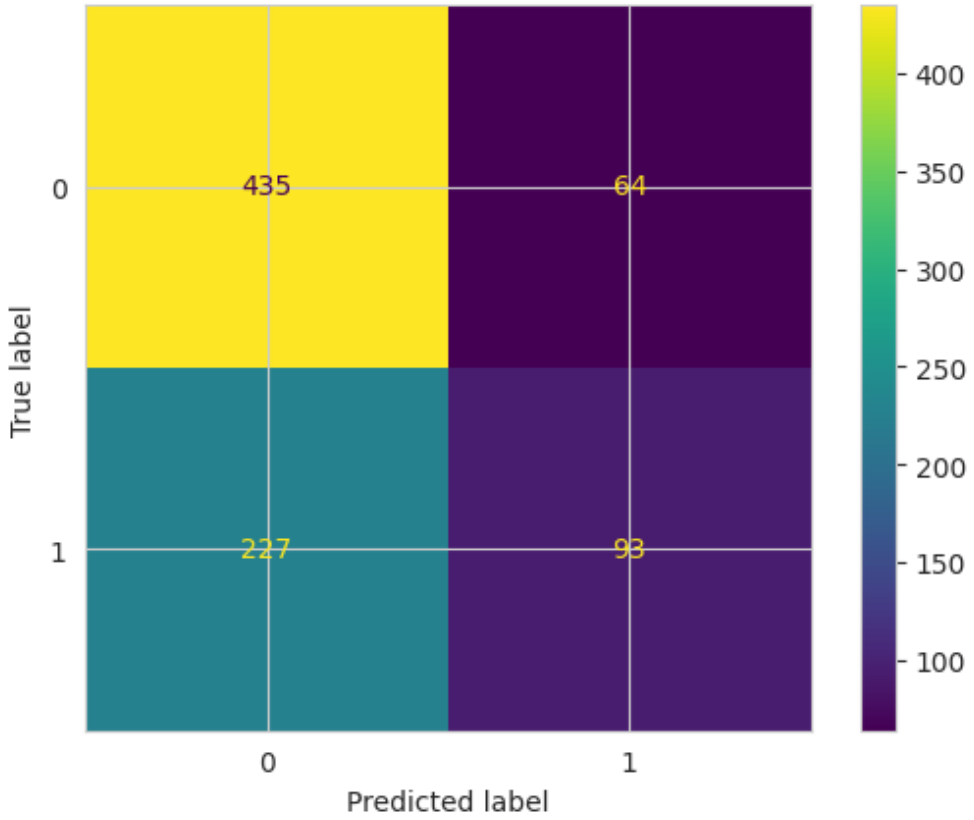
NOTA: pelo resultado do Optuna, pode-se observar que o melhor resultado: Trial 24 finished with value: 0.6491869918699187 and parameters:

```
{
    'max_depth': 12,
    'min_samples_split': 64,
    'min_samples_leaf': 4,
    'criterion': 'entropy'
}
```

Best is trial 24 with value: 0.6491869918699187.

```
# Testando modelos
classifier = DecisionTreeClassifier(
    max_depth = 12,
    min_samples_split = 64,
    min_samples_leaf = 4,
    criterion = 'entropy'
)

# Treinando o modelo, com dados de treinamento
```

```
classifier.fit(X_train_imputer, y_train)

#### Prevendo os valores de Y para os dados de teste (X_test)
y_test_pred = classifier.predict(X_test_imputer)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```




```
# Verificando os resultados
y_train_pred = classifier.predict(X_train_imputer)
y_test_pred = classifier.predict(X_test_imputer)

roc_auc_train = roc_auc_score(y_train, y_train_pred)
roc_auc_test = roc_auc_score(y_test, y_test_pred)

print(classification_report(y_test, y_test_pred))
print(f"Accuracy DTree = {accuracy_score(y_test, y_test_pred)}.")
print(f"AUC_train = {roc_auc_train}, AUC_test = {roc_auc_test}.")

RESULTADOS['DTree'] = [classifier, accuracy_score(y_test, y_test_pred), roc_auc_train
```

```
              precision    recall  f1-score   support

           0       0.66      0.87      0.75       499
           1       0.59      0.29      0.39       320

    accuracy                           0.64       819
   macro avg       0.62      0.58      0.57       819
weighted avg       0.63      0.64      0.61       819

Accuracy DTree = 0.6446886446886447.
AUC_train = 0.6494541942366588, AUC_test = 0.58118424348697
```

```
AUC_train = 0.8494941942500588, AUC_test = 0.9811842454860974.
```

NOTA: Faremos uma comparação entre os modelos ao final do estudo.

6. Modelagem: Random Forest

```
def objective_rf(trial):
    # -- Tune estimator algorithm
        # n_estimators: Número de Árvores na Floresta
    n_estimators = trial.suggest_int("n_estimators", 50, 150)
        # criterion: Função de Separação
    criterion = trial.suggest_categorical("criterion", ['gini', 'entropy', 'log_loss'
        # max_depth: Profundidade da Ávore
    max_depth = trial.suggest_categorical("max_depth", [2, 4, 8, 16])
        # min_samples_split: Número mínimo para separação de folhas
    min_samples_split = trial.suggest_categorical("min_samples_split", [2, 4, 8, 16])
        # min_samples_leaf: Número mínimo de amostras em cada folha
    min_samples_leaf = trial.suggest_categorical("min_samples_leaf", [1, 2, 4, 8])
        # max_features: Número máximo de features
    max_features = trial.suggest_categorical("max_features", ['sqrt', 'log2'])

    cls_rf = RandomForestClassifier(
        n_estimators = n_estimators,
        criterion = criterion,
        max_depth = max_depth,
        min_samples_split = min_samples_split,
        min_samples_leaf = min_samples_leaf,
        max_features = max_features
    )

    # -- Make a pipeline
    pipeline = make_pipeline(cls_rf)

    # -- Cross-validate the features reduced by dimensionality reduction methods
    kfold = StratifiedKFold(n_splits=10)
    score = cross_val_score(pipeline, X_train_imputer, y_train, scoring='accuracy', c
    score = score.mean()
    return score

sampler = TPESampler(seed=42)
study = optuna.create_study(direction="maximize", sampler=sampler)
study.optimize(objective_rf, n_trials=200)

# [I 2024-05-01 18:37:16,046] Trial 33 finished with value: 0.6646275095404016 and pa
```

```
[I 2024-05-01 18:48:58,112] A new study created in memory with name: no-name-2a088ea
[I 2024-05-01 18:49:05,560] Trial 0 finished with value: 0.6694790111166418 and para
[I 2024-05-01 18:49:15,661] Trial 1 finished with value: 0.6662435705989713 and para
[I 2024-05-01 18:49:25,238] Trial 2 finished with value: 0.6662435705989713 and para
[I 2024-05-01 18:49:32,172] Trial 3 finished with value: 0.6556711465073835 and para
[I 2024-05-01 18:49:34,350] Trial 4 finished with value: 0.6149759415961507 and para
[I 2024-05-01 18:49:36,562] Trial 5 finished with value: 0.6202721088435375 and para
[I 2024-05-01 18:49:39,162] Trial 6 finished with value: 0.6166019578563133 and para
[I 2024-05-01 18:49:43,206] Trial 7 finished with value: 0.6157922681267629 and para
```

```
[I 2024-05-01 18:49:50,192] Trial 8 finished with value: 0.6581168076986892 and para
[I 2024-05-01 18:50:03,480] Trial 9 finished with value: 0.6690957358553178 and para
[I 2024-05-01 18:50:11,829] Trial 10 finished with value: 0.6341048614567778 and par
[I 2024-05-01 18:50:25,280] Trial 11 finished with value: 0.6703135888501743 and par
[I 2024-05-01 18:50:37,090] Trial 12 finished with value: 0.6776389580222333 and par
[I 2024-05-01 18:50:50,106] Trial 13 finished with value: 0.6658354073336652 and par
[I 2024-05-01 18:50:56,658] Trial 14 finished with value: 0.6369586859133898 and par
[I 2024-05-01 18:51:11,719] Trial 15 finished with value: 0.6695121951219513 and par
[I 2024-05-01 18:51:24,726] Trial 16 finished with value: 0.6658271113323379 and par
[I 2024-05-01 18:51:37,658] Trial 17 finished with value: 0.6698987887838062 and par
[I 2024-05-01 18:51:49,808] Trial 18 finished with value: 0.6544449975111997 and par
[I 2024-05-01 18:51:56,779] Trial 19 finished with value: 0.63655550024888 and param
[I 2024-05-01 18:52:07,507] Trial 20 finished with value: 0.6674713787954205 and par
[I 2024-05-01 18:52:19,886] Trial 21 finished with value: 0.6715331010452961 and par
[I 2024-05-01 18:52:32,746] Trial 22 finished with value: 0.6686792765886842 and par
[I 2024-05-01 18:52:48,653] Trial 23 finished with value: 0.6654239256678282 and par
[I 2024-05-01 18:53:00,242] Trial 24 finished with value: 0.6715314418450307 and par
[I 2024-05-01 18:53:11,573] Trial 25 finished with value: 0.6662485481997676 and par
[I 2024-05-01 18:53:19,567] Trial 26 finished with value: 0.6658437033349927 and par
[I 2024-05-01 18:53:25,604] Trial 27 finished with value: 0.6369570267131243 and par
[I 2024-05-01 18:53:30,853] Trial 28 finished with value: 0.6560710137713621 and par
[I 2024-05-01 18:53:41,459] Trial 29 finished with value: 0.6739771030363364 and par
[I 2024-05-01 18:53:49,644] Trial 30 finished with value: 0.6703185664509707 and par
[I 2024-05-01 18:54:01,040] Trial 31 finished with value: 0.6690857806537249 and par
[I 2024-05-01 18:54:10,760] Trial 32 finished with value: 0.6764128090260495 and par
[I 2024-05-01 18:54:20,201] Trial 33 finished with value: 0.6703052928488468 and par
[I 2024-05-01 18:54:27,408] Trial 34 finished with value: 0.6666550522648084 and par
[I 2024-05-01 18:54:38,950] Trial 35 finished with value: 0.6711265969802555 and par
[I 2024-05-01 18:54:46,453] Trial 36 finished with value: 0.6694922847187655 and par
[I 2024-05-01 18:54:50,235] Trial 37 finished with value: 0.6157922681267629 and par
[I 2024-05-01 18:54:54,912] Trial 38 finished with value: 0.6605491952878711 and par
[I 2024-05-01 18:54:57,823] Trial 39 finished with value: 0.61456943753111 and param
[I 2024-05-01 18:55:06,854] Trial 40 finished with value: 0.6577185996349759 and par
[I 2024-05-01 18:55:19,236] Trial 41 finished with value: 0.6719396051103368 and par
[I 2024-05-01 18:55:30,286] Trial 42 finished with value: 0.6784453293512527 and par
[I 2024-05-01 18:55:41,067] Trial 43 finished with value: 0.6682711133233782 and par
[I 2024-05-01 18:55:49,045] Trial 44 finished with value: 0.6686660029865605 and par
[I 2024-05-01 18:55:57,000] Trial 45 finished with value: 0.6699004479840716 and par
[I 2024-05-01 18:55:59,697] Trial 46 finished with value: 0.6149792599966815 and par
[I 2024-05-01 18:56:04,981] Trial 47 finished with value: 0.6393943919031027 and par
[I 2024-05-01 18:56:15,382] Trial 48 finished with value: 0.6670582379293181 and par
[I 2024-05-01 18:56:23,371] Trial 49 finished with value: 0.6556744649079145 and par
[I 2024-05-01 18:56:37,493] Trial 50 finished with value: 0.6703119296499088 and par
[I 2024-05-01 18:56:50,128] Trial 51 finished with value: 0.6727708644433383 and par
[I 2024-05-01 18:57:03,458] Trial 52 finished with value: 0.6670516011282561 and par
[I 2024-05-01 18:57:19,117] Trial 53 finished with value: 0.6707300481168077 and par
[I 2024-05-01 18:57:35,051] Trial 54 finished with value: 0.6711315745810519 and par
[I 2024-05-01 18:57:51,267] Trial 55 finished with value: 0.6707217521154802 and par
[I 2024-05-01 18:58:00,905] Trial 56 finished with value: 0.666238592998175 and para
```

Resultados do Random Forest

[I 2024-05-01 18:49:05,560] Trial 0 finished with value: 0.6694790111166418 and parameters: {'n_estimators': 87, 'criterion': 'gini', 'max_depth': 16, 'min_samples_split': 16, 'min_samples_leaf': 1, 'max_features': 'log2'}. Best is trial 0 with value: 0.6694790111166418.

[I 2024-05-01 18:50:37,090] Trial 12 finished with value: 0.6776389580222333 and parameters: {'n_estimators': 97, 'criterion': 'log_loss', 'max_depth': 16, 'min_samples_split': 16,

parameters: {'n_estimators': 97, 'criterion': 'log_loss', 'max_depth': 16, 'min_samples_split': 16, 'min_samples_leaf': 1, 'max_features': 'log2'}. Best is trial 12 with value: 0.6776389580222333.

[I 2024-05-01 18:55:30,286] Trial 42 finished with value: 0.6784453293512527 and parameters: {'n_estimators': 91, 'criterion': 'log_loss', 'max_depth': 16, 'min_samples_split': 16, 'min_samples_leaf': 1, 'max_features': 'log2'}. Best is trial 42 with value: 0.6784453293512527.

```
# Processando Dados
scaler_ss = StandardScaler()
scaler_ss.fit(X_train)
X_train_ss = scaler_ss.transform(X_train_imputer)
X_test_ss = scaler_ss.transform(X_test_imputer)

# Testando modelos
classifier = RandomForestClassifier(
    n_estimators = 91,
    criterion = 'log_loss',
    max_depth = 16,
    min_samples_split = 16,
    min_samples_leaf = 1,
    max_features = 'log2'
)

# Treinando o modelo, com dados de treinamento
classifier.fit(X_train_ss, y_train)

#### Prevendo os valores de Y para os dados de teste (X_test)
y_test_pred = classifier.predict(X_test_ss)

# Verificando os resultados
y_train_pred = classifier.predict(X_train_ss)
y_test_pred = classifier.predict(X_test_ss)

roc_auc_train = roc_auc_score(y_train, y_train_pred)
roc_auc_test = roc_auc_score(y_test, y_test_pred)

print(classification_report(y_test, y_test_pred))
print(f"Accuracy RF = {accuracy_score(y_test, y_test_pred)}.")
print(f"AUC_train = {roc_auc_train}, AUC_test = {roc_auc_test}.")
```

```
                precision    recall  f1-score   support

             0       0.66      0.93      0.77       499
             1       0.69      0.26      0.38       320

      accuracy                           0.67       819
     macro avg       0.68      0.59      0.58       819
  weighted avg       0.67      0.67      0.62       819

  Accuracy RF = 0.6666666666666666.
  AUC_train = 0.8404217982482407, AUC_test = 0.5941758517034068.
```

NOTA: Aplicando os parametros estipulados pelo Optuna, percebemos que a Curva AUC do treino está elevada, em relação ao AUC de teste, o que aponta para Overfiting.

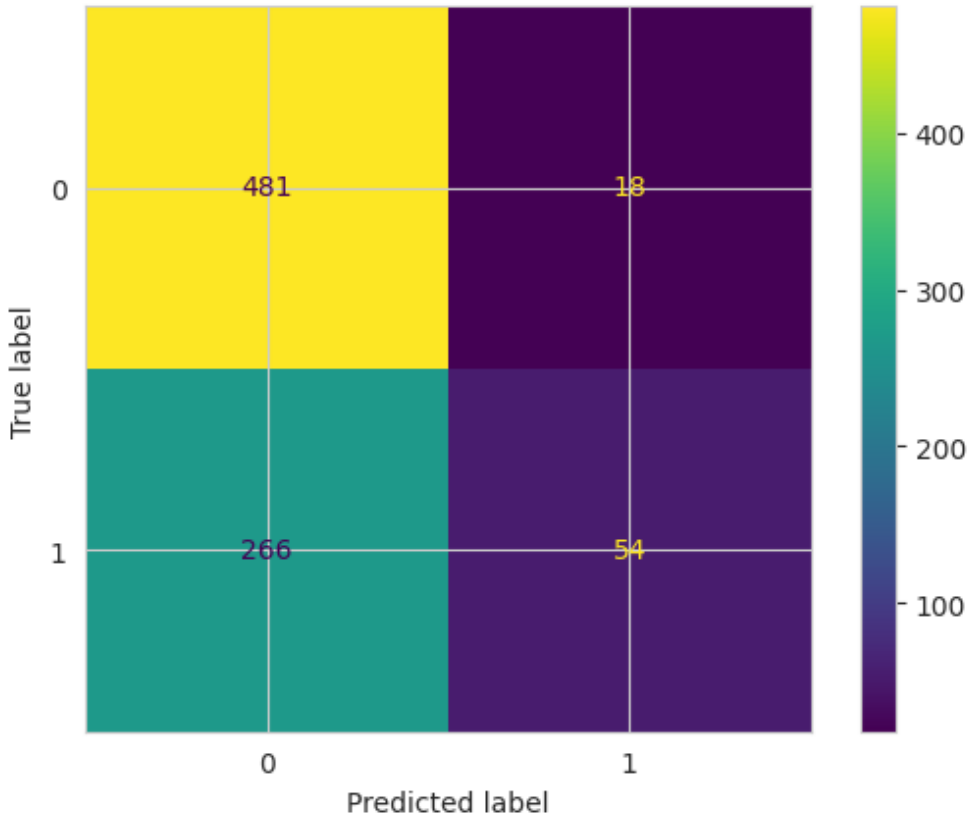
Como forma de garantir um resultado melhor, testou-se manualmente pequenas variações próximo aos valores retornados pelo Optuna para melhorar o modelo.

```
# Testando modelos
classifier = RandomForestClassifier(
    n_estimators = 91,
    criterion = 'log_loss',
    max_depth = 8,
    min_samples_split = 16,
    min_samples_leaf = 1,
    max_features = 'log2'
)

# Treinando o modelo, com dados de treinamento
classifier.fit(X_train_ss, y_train)

#### Prevendo os valores de Y para os dados de teste (X_test)
y_test_pred = classifier.predict(X_test_ss)

# Verificando os resultados
y_train_pred = classifier.predict(X_train_ss)
y_test_pred = classifier.predict(X_test_ss)

roc_auc_train = roc_auc_score(y_train, y_train_pred)
roc_auc_test = roc_auc_score(y_test, y_test_pred)

print(classification_report(y_test, y_test_pred))
print(f"Accuracy RF = {accuracy_score(y_test, y_test_pred)}.")
print(f"AUC_train = {roc_auc_train}, AUC_test = {roc_auc_test}.")
```

```
                  precision    recall  f1-score   support

               0       0.64      0.96      0.77       499
               1       0.75      0.17      0.28       320

        accuracy                           0.65       819
       macro avg       0.70      0.57      0.52       819
    weighted avg       0.69      0.65      0.58       819

    Accuracy RF = 0.6532356532356532.
    AUC_train = 0.6594016748813754, AUC_test = 0.5663389278557114.
```

NOTA: após testadas as variações, percebeu-se que a redução do max_depth de 16 para 8, reduziria a curva AUC de train de 0.82 para 0.64, mantendo a Acurácia em 0.65 e ficando uma diferença uma diferença de 0.08 entre AUC_train e AUC_test, diferente da diferença de 0.24, da modelagem anterior.

```
# # Create the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```




```
# Salvar resultado
RESULTADOS['RF'] = [classifier, accuracy_score(y_test, y_test_pred), roc_auc_train, r
```

7. Testando outros Modelos

```
# Libraries
from warnings import filterwarnings

# Modelling Libraries
from sklearn.linear_model import LogisticRegression,RidgeClassifier,SGDClassifier,Pas
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC,LinearSVC,NuSVC
from sklearn.neighbors import KNeighborsClassifier,NearestCentroid
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoosti
from sklearn.naive_bayes import GaussianNB,BernoulliNB
from sklearn.ensemble import VotingClassifier

# Evaluation & CV Libraries
from sklearn.metrics import precision_score,accuracy_score
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV,RepeatedStratifie
```

```
# Ignorar alertas
filterwarnings('ignore')

# Lista Modelos
models =[
    ('ADA',AdaBoostClassifier()),
    ("BNB", BernoulliNB())
```

```
    ("BNB", BernoulliNB()),
    ("DTC", DecisionTreeClassifier(
        max_depth = 12,
        min_samples_split = 64,
        min_samples_leaf = 4,
        criterion = 'entropy'
    )),
    ("GNB", GaussianNB()),
    ('KNN',KNeighborsClassifier(
        leaf_size= 6,
        n_neighbors= 41,
        weights= 'distance',
        metric= 'euclidean'
    )),
    ("LR", LogisticRegression(max_iter=1000)),
    ("NC",NearestCentroid()),
    ("NuSVC", NuSVC()),
    ('PAC',PassiveAggressiveClassifier()),
    ("Perc", Perceptron()),
    ("Ridge", RidgeClassifier()),
    ("SGDC", SGDClassifier()),
    ("SVC", SVC()),
    ('RF',RandomForestClassifier(
        n_estimators = 91,
        criterion = 'log_loss',
        max_depth = 8,
        min_samples_split = 16,
        min_samples_leaf = 1,
        max_features = 'log2'
    )),
    ('XGB',GradientBoostingClassifier())
]

# Testa modelos
resultado_final = []
for nome, model in models:
    model.fit(X_train_ss, y_train)
    model_results = model.predict(X_test_ss)
    accuracy = accuracy_score(y_test, y_test_pred_model)
    precision = precision_score(y_test, model_results, average='macro')
    recall = recall_score(y_test, y_test_pred_model)

    y_train_pred_model = model.predict(X_train_ss)
    y_test_pred_model = model.predict(X_test_ss)

    roc_auc_train_model = roc_auc_score(y_train, y_train_pred_model)
    roc_auc_test_model = roc_auc_score(y_test, y_test_pred_model)

    resultado_final.append((nome, accuracy, precision, recall, roc_auc_train_model, r

resultado_final.sort(key=lambda k:k[1], reverse=True)
```

```
df_algoritmos = pd.DataFrame(resultado_final)
df_algoritmos.columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'ROC_AUC_Train',
df_algoritmos['Variação'] = df_algoritmos['ROC_AUC_Train'] - df_algoritmos['ROC_AUC_T
df_algoritmos
```

```
df_algoritmos
```

	Model	Accuracy	Precision	Recall	ROC_AUC_Train	ROC_AUC_Test	Variação
0	RF	0.666667	0.700334	0.253125	0.655372	0.565778	0.089593
1	ADA	0.658120	0.566296	0.237500	0.585784	0.540356	0.045428
2	XGB	0.653236	0.662880	0.165625	0.675389	0.579553	0.095836
3	GNB	0.644689	0.558887	0.290625	0.558044	0.530505	0.027539
4	PAC	0.644689	0.507737	0.393750	0.502941	0.508110	-0.005169
5	LR	0.638584	0.304640	0.140625	0.500000	0.500000	0.000000
6	BNB	0.609280	0.304640	0.225000	0.500000	0.500000	0.000000
7	DTC	0.609280	0.624728	0.000000	0.649454	0.581184	0.068270
8	NC	0.609280	0.515208	0.000000	0.517005	0.515922	0.001083
9	SGDC	0.609280	0.483497	0.000000	0.520855	0.482922	0.037933
10	KNN	0.606838	0.658306	0.181250	1.000000	0.549270	0.450730
11	Ridge	0.526252	0.304640	0.509375	0.500000	0.500000	0.000000
12	NuSVC	0.509158	0.619760	0.546875	0.838056	0.599681	0.238375
13	Perc	0.503053	0.522147	0.531250	0.503074	0.523225	-0.020150
14	SVC	0.496947	0.682430	0.418750	0.674620	0.592494	0.082126

NOTA: Neste comparação entre modelos de classificação, o Decision Tree Classifier (DTC), com os parametros ajustados pelo optuna, teve um desempenho mais satisfatório. Mesmo com uma acurácia menor, o modelo apresentou a menor diferença entre curva ROC AUC Train e Test, o que demonstra mais próximo do ideal, com menor Overfiting.

Os demais outro modelos podem apresentar uma performance melhor que os que foram testados (Decision Tree Classifier, Random Forest Classifier e KNN) de forma mais acurada, se for refinar a análise otimizando os hiperparâmetros.

8. Resultados Testados

```
df_resultados = pd.DataFrame(RESULTADOS).T
df_resultados.columns = ['Parametros', 'Acurácia', 'ROC AUC Train', 'ROC AUC Test']
df_resultados['Diff_ROC'] = df_resultados['ROC AUC Train'] - df_resultados['ROC AUC T
df_resultados.sort_values(by="Diff_ROC")
```

	Parametros	Acurácia	ROC AUC Train	ROC AUC Test	Diff_ROC
DTree	DecisionTreeClassifier(criterion='entropy', ma...	0.644689	0.649454	0.581184	0.06827

RF	(DecisionTreeClassifier(criterion='log_loss', ...	0.653236	0.659402	0.566339	0.093063

Analisando os resultados obtidos entre os modelos otimizados, pode-se observar que:

1. KNN - apesar da várias possibilidades testadas, houve overfiting sempre presente. Mesmo com uma acurácia maior que os demais, o modelo está muito ajustado aos dados de treino, o que pode comprometer os resultados durante o uso do modelo.

2. RF - Random Forest: teve um bom desempenho, por apesar de uma acuracia elevada, outro modelos apresentou uma diferença menor entre as curvas ROC AUC de Train e Test.

3. DTree - Decision Tree: teve um bom desempenho, mesmo com uma pequena acurácia, o modelo apresentou a menor diferença entre curva ROC AUC Train e Test, o que demonstra mais próximo do ideal.

9. Conclusão

Certamente, como proposta de trabalho avaliativo, os resultados obtidos são suficientes para demostrânção e aplicação de conhecimento. Entretanto, em mum trabalho real seria necessário maior aprofundamentos e exploração, esgotando outras possibilidades, modelos e algoritmos, as diversas opções existentes para classificação, principalmente binária.

Para maior certeza, dentre os diversos algoritmos, poderia-se testar os hiperparâmetros de outros modelos, tais como:

```
AdaBoostClassifier();
BernoulliNB();
GaussianNB();
GradientBoostingClassifier();
Logistic Regression();
NearestCentroid();
NuSVC();
PassiveAggressiveClassifier();
Perceptron();
RidgeClassifier();
SGDClassifier();
SVC();
```

Este projeto visou realizar uma aplicação prática das técnicas de **Machine Learning I** em uma base de dados real, fazendo uso da linguagem de programação **Python**, através do Jupyter Notebook e/ou Colab.

Todas as técnicas aplicadas nesta análise foram algumas dentre as diversas aprendidas no módulo de **Machine Learning I**, podendo ser aplicadas diversas outras técnicas existentes, e testados outros modelos além dos que foram abordados no cronograma do curso, explorando outras possiblidades dentro do conhecimento de Aprendizado de Maquina Supervisionado.

Como aplicação prática na realidade do Analista e Cientista de Dados, foi explorado um dataset exportado do Kaggle, contendo aproximadamente 3200 registros de testes realizados para certificar a Potabilidade da Água.

Durante todo o processo, passou-se por três etapas:

- **Preparação e Carregamento**: onde instalamos e carregamos nosso recursos, construímos nosso conjunto de dados (carregando os dados e explorando as variáveis) e agrupamos nosso dados;
- **Processamento e Análise**: onde foi realizada uma primeira análise de estruturas e tipos de dados armazenados, bem como apreciamos a dimensão e como estão organizados os dados e suas particularidades;
- **Testamos e Deduzimos**: nessa última etapa aplicamos a modelagem e buscamos gerar alguns insights do conjunto de dados, explorando o conjunto no todo e em partes, testando possibiilidades, e gerando informações possivelmente relevantes para uma análise na vida real. Destaca-se, que esta análise tem intuito prático e didático, sendo possível estender muito mais toda a análise se caso estivéssemos tratando de uma análise real dentro de uma necessidade de negócio. Buscando, assim, mais informações relevantes para melhores decisões.

Ficam disponíveis todos os arquivos: código e links, disponíveis e abertos para quem estiver interessado em aprender, corrigir ou aperfeiçoar seus trabalhos.

Obrigado.