# Relatório Prático - Inteligência Artificial (PPGI)

## Autor:


- ***Lucas Pereira de Lira***


## Professor

- ***Dr. Alexandre Rossi Paschoal***

## Especificações da Base de Dados

- ***Nome:***
  - Qualidade da água, Potabilidade da água potável
  - Water Quality, Drinking water potability
- ***Contexto:***
  - O acesso à água potável é essencial para a saúde, um direito humano básico e um componente de uma política eficaz de proteção à saúde. Isso é importante como uma questão de saúde e desenvolvimento em nível nacional, regional e local. Em algumas regiões, foi demonstrado que os investimentos em abastecimento de água e saneamento podem gerar um benefício econômico líquido, uma vez que as reduções nos efeitos adversos à saúde e nos custos de saúde superam os custos de empreender as intervenções.
- ***Estrurura:***
  - ***Observação***

    O arquivo water_potability.csv contém métricas de qualidade da água para 3276 corpos d'água diferentes.

  - ***1. PH (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 OMS recomendou o limite máximo permitido de pH de 6,5 a 8,5. Os intervalos de investigação atuais foram de 6,52–6,83, que estão na faixa dos padrões da OMS.

  - ***2. Dureza/Solidez (hardness)***

    A dureza é causada principalmente pelos sais de cálcio e magnésio. Esses sais são dissolvidos de depósitos geológicos pelos quais a água viaja. O tempo que a água está em contato com o material que produz dureza ajuda a determinar quanta dureza existe na água bruta. A dureza foi originalmente definida como a capacidade da água de precipitar o sabão causado pelo cálcio e magnésio.

  - ***3. Sólidos (total de sólidos dissolvidos - TDS)***

    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 produziram um sabor indesejável e uma cor diluída na aparência de 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 o TDS é de 500 mg / le o limite máximo é de 1000 mg / l, prescrito para beber.

- ***4. Cloraminas (chloramines)***

  O cloro e a cloramina são os principais desinfetantes usados nos sistemas públicos de água. As cloraminas são mais comumente formadas quando a amônia é adicionada ao cloro para tratar a água potável. Os níveis de cloro de até 4 miligramas por litro (mg / L ou 4 partes por milhão (ppm)) são considerados seguros na água potável.

- ***5. Sulfato (sulfate)***

  Os sulfatos são substâncias que ocorrem naturalmente e são encontradas em minerais, solo e rochas. Eles estão presentes no ar ambiente, águas subterrâneas, plantas e 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 dos suprimentos de água doce, embora concentrações muito mais altas (1000 mg / L) sejam encontradas em algumas localizações geográficas.

- ***6. Condutividade (conductivity)***

  Água pura não é um bom condutor de corrente elétrica, em vez disso, é 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) realmente mede o processo iônico de uma solução que permite a transmissão de corrente. De acordo com os padrões da OMS, o valor de CE não deve exceder 400 µS / cm.

- ***7. Carbono organico (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 US EPA, 2 mg/L como TOC em água tratada/potável e 4mg/Lit em água de origem que é usada para tratamento.

- ***8. Trihalometanos (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. Turbidez (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 turvação obtido para o Wondo Genet Campus (0,98 NTU) é inferior ao valor recomendado pela OMS de 5,00 NTU.

- ***10. Potabilidade (potability)***

  Indica se a água é segura para consumo humano, onde 1 significa potável e 0 significa não potável.

- ***Link de Acesso:*** ***https://www.kaggle.com/adityakadiwal/water-potability (https://www.kaggle.com/adityakadiwal/water-potability)***

# Objetivo

Desenvolver uma análise exploratória dos dados, de tal forma a abordar os seguintes conceitos: visualização e interpretação dos dados (Matplotlib e Seaborn), identificação de outliers, implementação de técnicas para o tratamento de dados faltantes, desenvolvimento de modelos de machine learning e implementação de métricas para a avaliação dos modelos.

# 1. Introdução

A fim de tornar o trecho de codificação uma área mais aberta à análise de dados e dos modelos desenvolvidos, utilizaremos este capítulo para introduzir os conceitos e técnicas relevantes ao entendimento do trabalho. Por se tratar de conceitos que podem ser utilizados em diferentes pontos do código, eles foram centralizados nesta área, de tal forma que o leitor possa consultá-los a qualquer momento.

## 1.1 Visualização dos Dados

Nesta seção, introduziremos o conjunto de técnicas de visualização utilizadas na analise das amostras de água.

### Scatter plot

Os gráficos de dispersão ou scatter plot são representações do relacionamento entre variáveis numéricas. Sua representação é baseada em pontos que são exibidos utilizando coordenadas cartesianas. Os dados são exibidos como um conjunto de pontos, onde cada ponto representa o valor de uma variável no eixo horizontal e o valor de outra variável no eixo vertical. Em resumo, são utilizados para verificar se existe relação de causa e efeito entre variáveis. Note que isso não significa que uma variável causa efeito na outra e sim se possui alguma relação e qual sua intensidade, podendo ser positiva, negativa ou neutra, linear ou não linear conforme apresentado na figura abaixo.

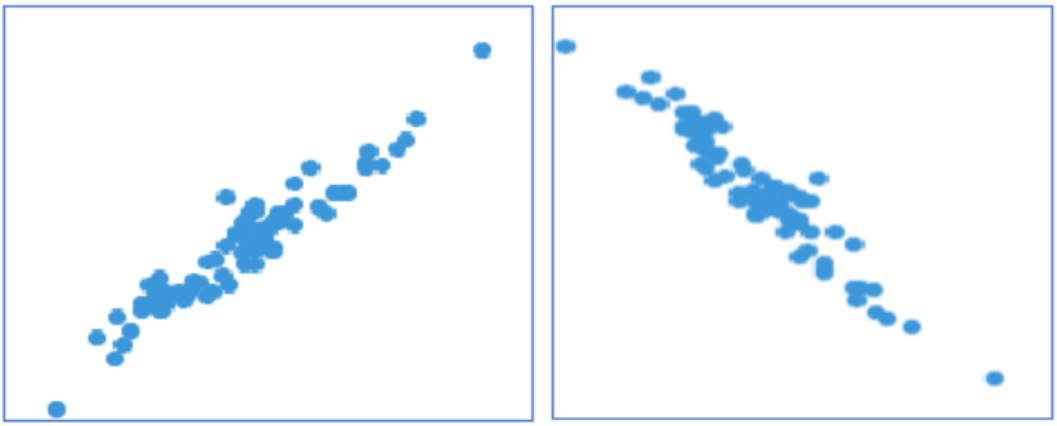


*Figura: Relação linear positiva (esquerda) e negativa (direita)*

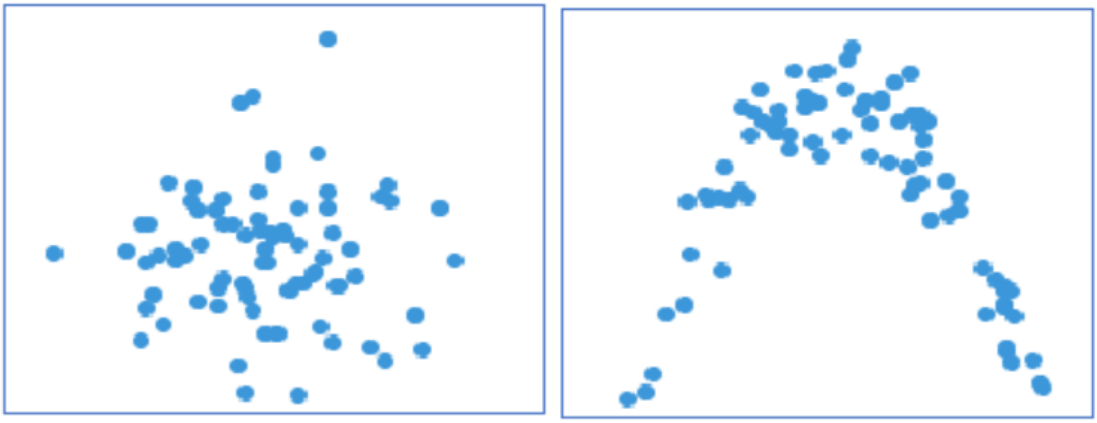


*Figura: Relação neutra (esquerda) e não linear (direita)*

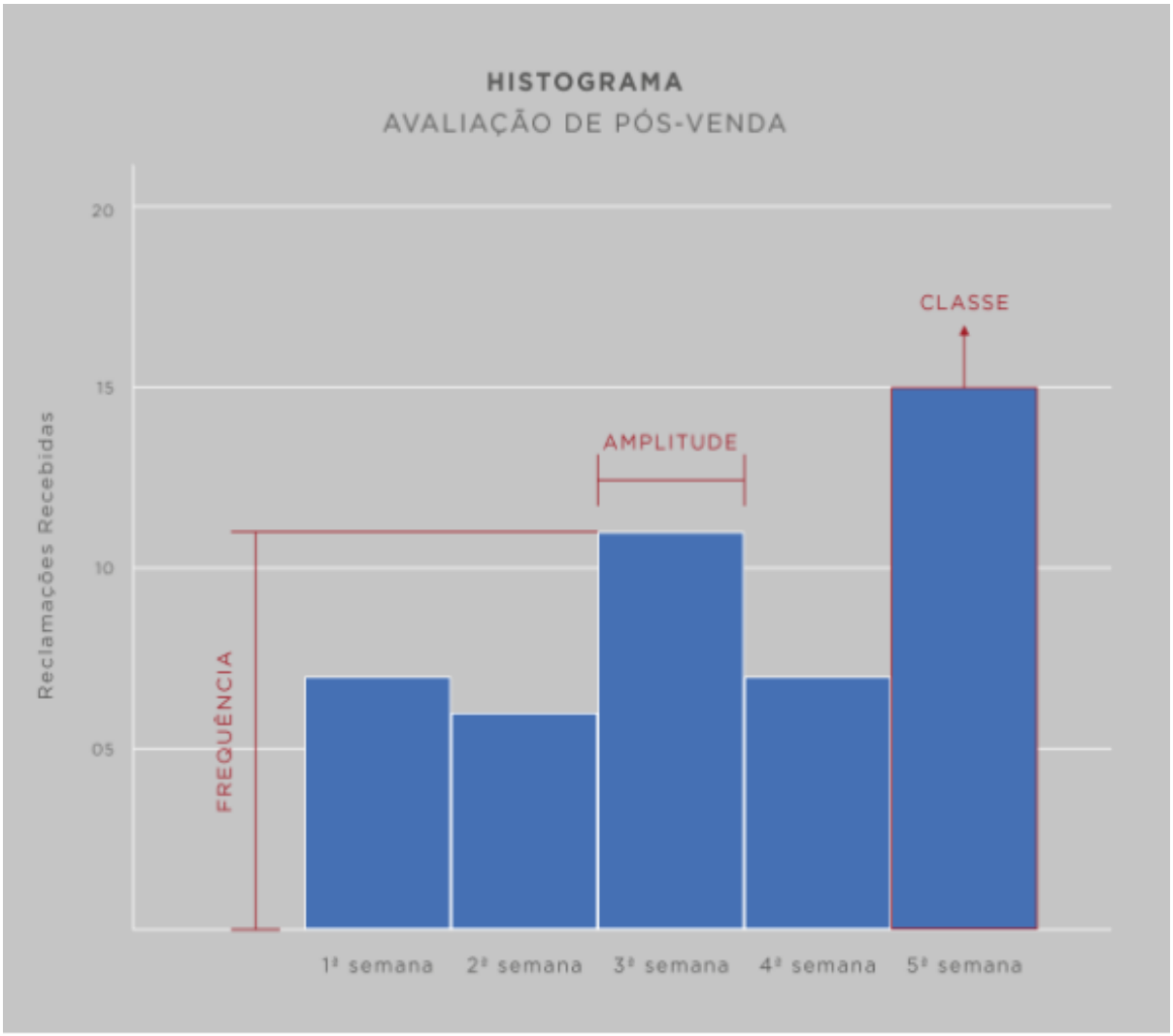
Além disso, esse tipo de gráfico também é útil para identificar pontos fora da curva (outliers) ou possíveis grupos nos dados;

## Histograma

Um histograma nada mais é que um gráfico (em forma de barras) que demonstra uma distribuição de frequências. No histograma, a base de cada uma das barras (eixo x) representa uma classe e a altura (eixo y) representa a quantidade ou frequência com que o valor de cada classe ocorre no conjunto de dados. É frequêntemente utilizado em processamento de imagens, entretanto, não se limita apenas a isso. Com ele temos recursos para responder questões do tipo:

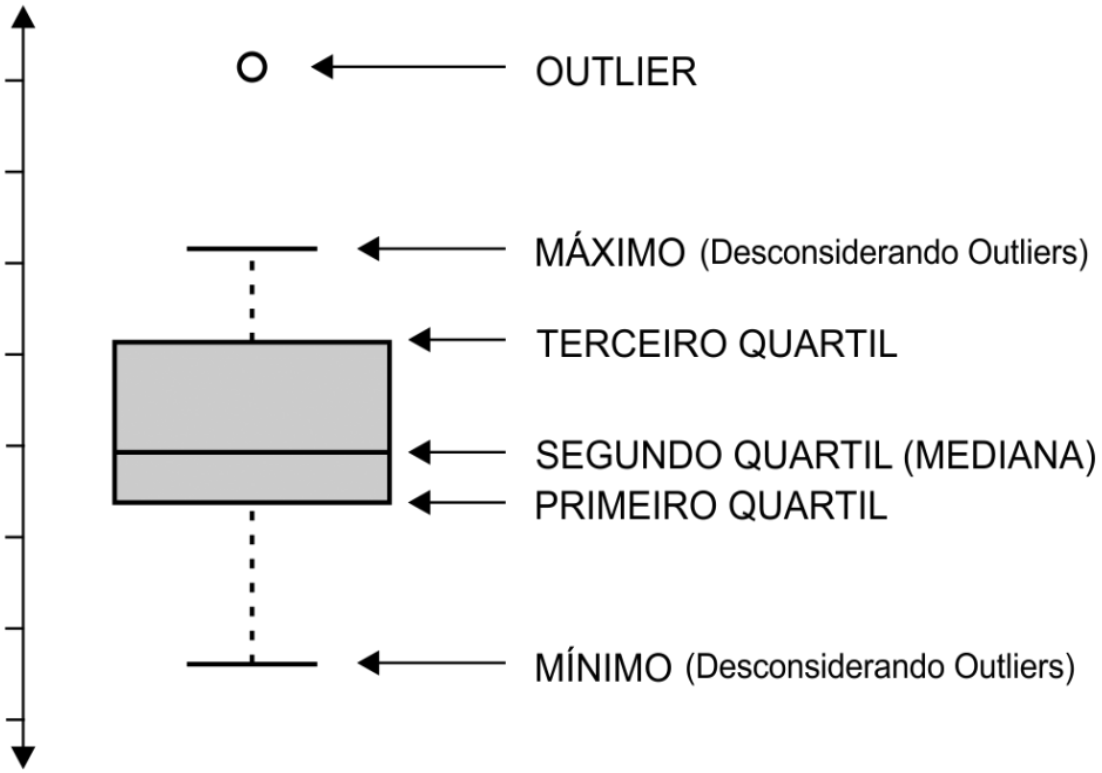
- Qual é o centro da distribuição?
- Onde é esperado que esteja a maioria das observações?
- A distribuição normalmente contém observações entre quais valores?
- Qual é o ponto de máximo e o ponto de mínimo?
- Será que devemos esperar a mesma frequência de pontos com valor alto e com valor baixo?
- Será que o processo é simétrico ou valores mais altos são mais raros?

Conforme imgem abaixo um histograma é formado por três elementos que podem nos ajudar a responder essas e outras questões, que são: classe, amplitude e frequência.



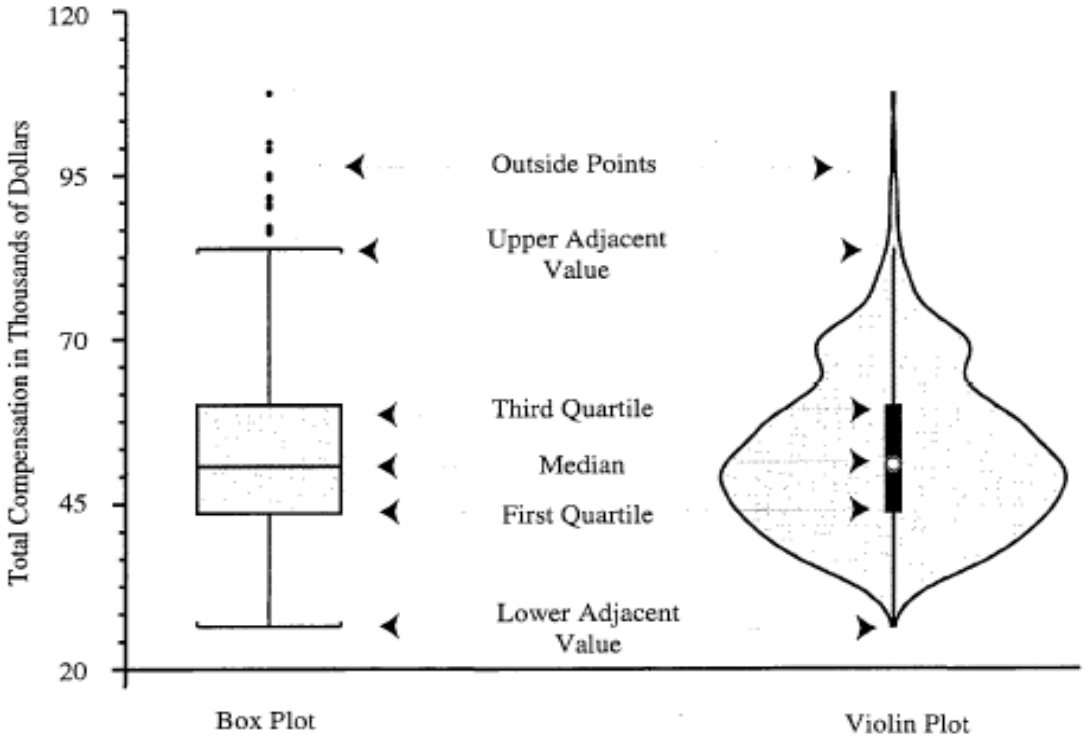

## Box plot

Box plot ou diagrama de caixa é um diagrama estatístico que representa, de maneira gráfica, a alteração dos dados de uma variável por meio de quartis. O box plot fornece informações quanto às seguintes características dos dados: localização, dispersão, assimetria, comprimento da cauda e outliers (dados discrepantes). Embora forneça todas essas informações, um dos pontos mais relevantes está na cauda da distribuição, na qual é possível identificar pontos fora da curva (outliers) que podem afetar diretamente as decisões a serem tomadas a partir da análise dos dados. Para exemplificar, a imagem abaixo apresenta em detalhes as divisões estatísticas do box plot.




## Violin plot

Violin plot são gráficos semelhantes aos box plot, entretanto, além das informações que o box plot nos trás o violin plot também mostra a densidade de probabilidade dos dados em valores diferentes, geralmente suavizados por um estimador de densidade de kernel. Considerando ambos os gráficos, o violin plot é consideravelmente mais informativo que o box plot. Enquanto o box plot trabalha com estatísticas resumidas, como média/mediana e intervalos interquartis, o violin plot apresenta a distribuição completa dos dados, essa diferença é útil quando a distribuição de dados é multimodal. Nesse caso, um violin plot mostra a presença de diferentes picos, sua posição e amplitude relativa. Como o box plot, o violin plot é utilizado para representar a comparação de uma distribuição variável (ou distribuição de amostra) em diferentes "categorias" (por exemplo, distribuição de temperatura comparada entre dia e noite ou distribuição de preços de automóveis comparados entre diferentes fabricantes de automóveis). A figura abaixo ilustra as diferenças e igualdades entre um box plot e um violin plot.

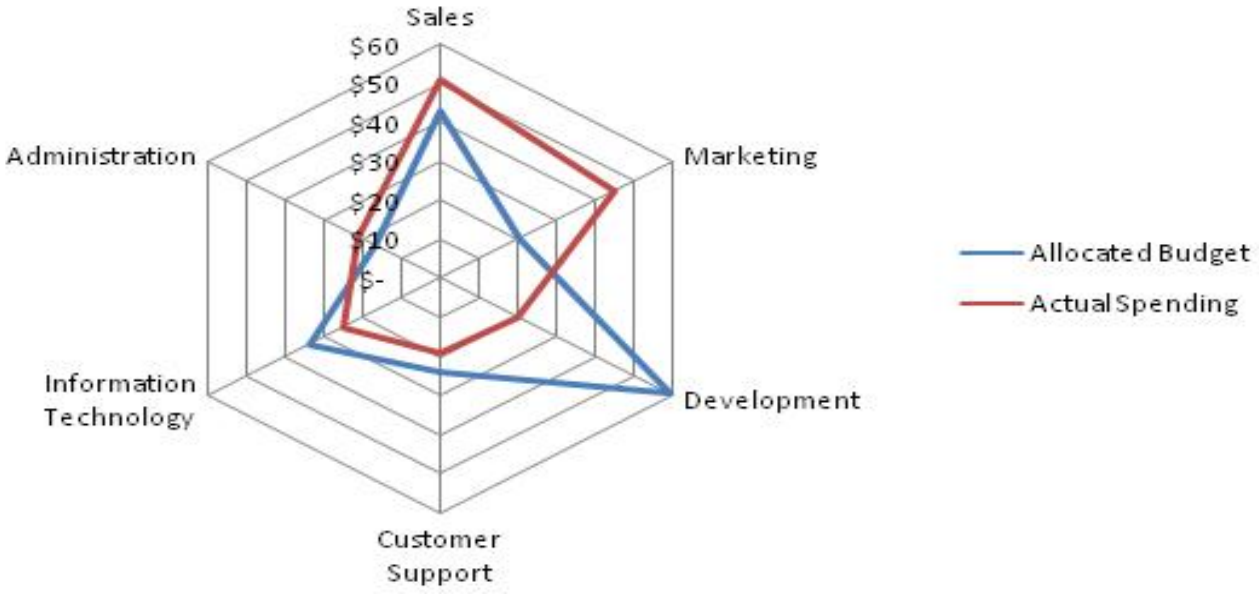



## Heatmap

Os heatmaps ou mapas de calor são representações gráficas que utilizam o espectro de cores, das mais "quentes" (vermelhas) as mais "frias" (azuis), para identificar áreas com maior e menor atividade, levando em consideração o contexto ao qual está sendo aplicado. O exemplo mais simples seria a representação de calor onde o heatmap poderia mostrar as áreas com menor e maior intensidade de calor. Outra aplicação comum é na observação do comportamento dos usuários ao acessar uma página da web, onde empresas utilizam mapas de calor para identificar possíveis pontos no design da página que possam estar prejudicando a experiência dos usuários.

## Radar plot

Radar plot ou também conhecido como gráfico de teia ou gráfico de aranha é um modelo visual utilizado para apresentar dados multivariáveis na forma de um gráfico bidimensional de três ou mais variáveis quantitativas representadas em eixos que partem do mesmo ponto. A exemplo, a imagem a seguir ilustra a estrutura de um radar plot.



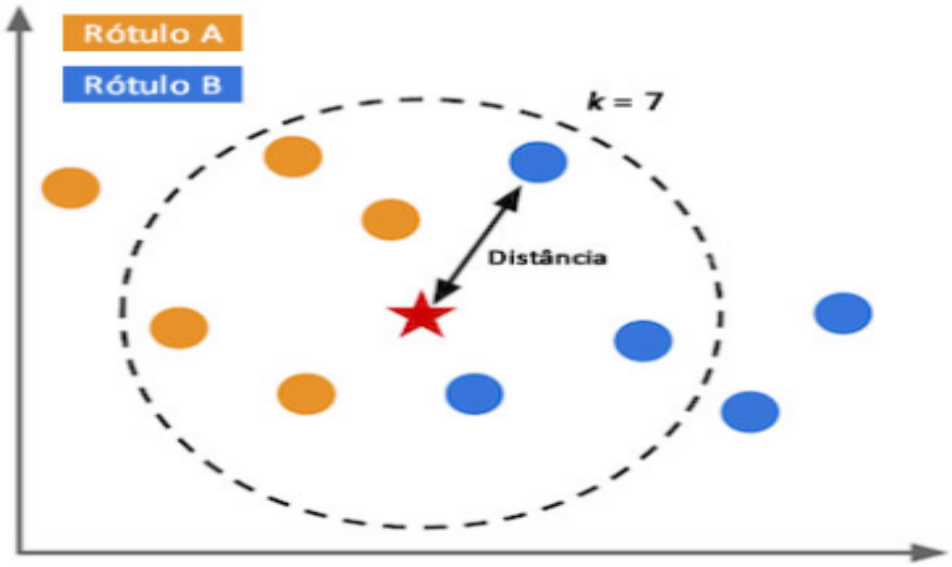

Conforme podemos observar na imagem acima o gráfico consiste em uma sequência de raios equiangulares, onde cada raio representa uma variável. Nesse caso os comprimentos dos raios são proporcionais à magnitude da variável. Um exemplo prático para a utilização desse tipo de gráfico é no controle de melhoria da qualidade para apresentar as métricas de desempenho de qualquer programa em curso, que inclusive é a aplicação utilizada para este gráfico neste trabalho. Além disso, eles também são utilizados no esporte, onde servem para representar os pontos fortes e fracos dos jogadores.

## 1.2 Classificadores

Nesta seção, introduziremos o conjunto de algoritmos abordados ao longo deste trabalho.

### KNN

KNN ou K vizinhos mais próximos (do inglês: K-nearest neighboors – KNN) é um algoritmo de classificação baseado em distância. Sendo assim a idéia é determinar o rótulo de classificação da amostra baseando-se nas amostras vizinhas advindas de um conjunto de treinamento. Para exemplificar utilizaremos o exemplo do Dr. André Pacheco. Ao analisar a figura abaixo nos deparamos com um problemas de classificação com dois rótulos de classe e com k = 7. No exemplo, são aferidas as distâncias de uma nova amostra, representada por uma estrela, às demais amostras de treinamento, representadas pelas bolinhas azuis e amarelas. A variável k representa a quatidade de vizinhos mais próximos que serão utilizados para averiguar de qual classe a nova amostra pertence. Com isso, das sete amostras de treinamento mais próximas da nova amostra, 4 são do rótulo A e 3 do rótulo B. Portanto, como existem mais vizinhos do rótulo A, a nova amostra receberá o mesmo rótulo deles, ou seja, A.



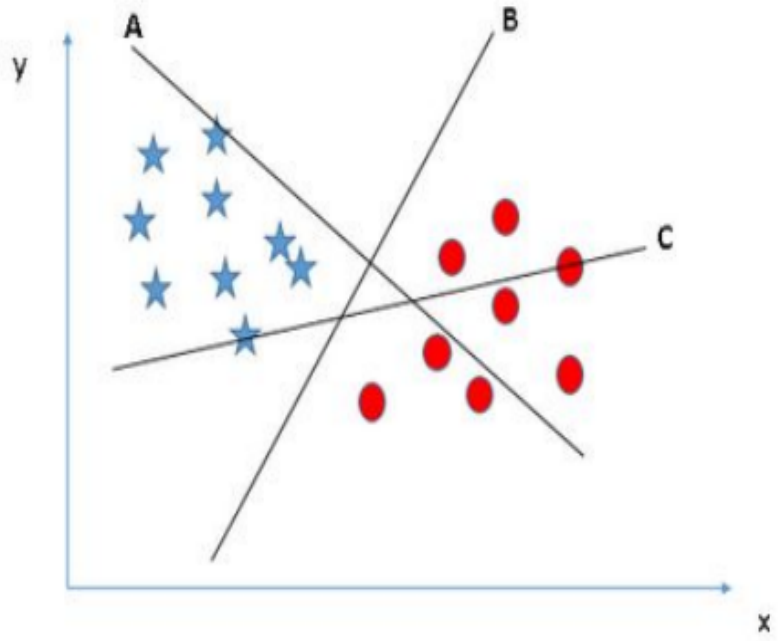

Note que a escolha do método para o cálculo da distância e o valor de k são de estrema importância para um bom desempenho do algoritmo, além disso, esses parâmetros irão depender do problema ao qual o algoritmo será aplicado e das características do conjunto de dados.

## Random Forest

Random Forest ou floresta aleatória como o nome já diz é um algoritmo de machine learning que irá criar muitas árvores de decisão, de maneira aleatória, formando assim uma floresta, onde cada árvore contribuirá com o resultado final. Esse algoritmo é geralmente utilizado na classificação e regressão, entretanto, não se limita somente a estas aplicações. Para tarefas de classificação, a saída da floresta aleatória é a classe selecionada pela maioria das árvores. Enquanto que, para tarefas de regressão, a média ou previsão média das árvores individuais é retornada. Florestas de decisão aleatórias corrigem o hábito de sobreajuste das árvores de decisão em seu conjunto de treinamento e geralmente superam árvores de decisão, entretanto, sua precisão é inferior à das árvores com aumento de gradiente. No entanto, as características dos dados podem afetar seu desempenho.

## SVM - Support Vector Machine

Consiste em um algoritmo de aprendizado de máquina supervisionado que pode ser utilizado tanto para classificação quanto para a regressão, com utilização mais comum em problemas de classificação. Nesse algoritmo, colocamos cada item de dados como um ponto no espaço n-dimensional (onde n é o número de recursos que você tem), com o valor de cada recurso sendo o valor de uma determinada coordenada. Então, nós executamos a classificação encontrando o hiperplano que melhor diferencia as duas classes. Considerando a imagem abaixo, onde temos três hiperplanos (A, B e C) o algoritmo busca identificar qual o melhor hiperplano para diferenciar os dados. Com isso em mente, o algoritmo pode calcular as distâncias entre os pontos de dados mais próximos e o hiperplano, de tal forma a maximizar as distâncias e assim selecionar o melhor hiperplano. Vale lembrar que antes de maximizar a margem entre o hiperplano e os pontos de dados o SVM faz uma seleção para verificar se o hiperplano classifica as classes com precisão (sem nenhuma falha).




## Gradient Tree Boosting

O modelo de conjunto de árvore na equação a seguir, inclui funções como parâmetros e não pode ser otimizado usando métodos tradicionais de otimização no espaço euclidiano. Em vez disso, o modelo é treinado de maneira aditiva. Formalmente, seja Yi^(t) a previsão da i-ésima instância na t-ésima iteração, precisaremos adicionar ft para minimizar o seguinte objetivo [2].

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

Isso significa que adicionamos avidamente o ft que mais melhora nosso modelo de acordo com a equação abaixo. A aproximação de segunda ordem pode ser usada para otimizar rapidamente o objetivo no cenário geral [3].

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

Onde

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

e

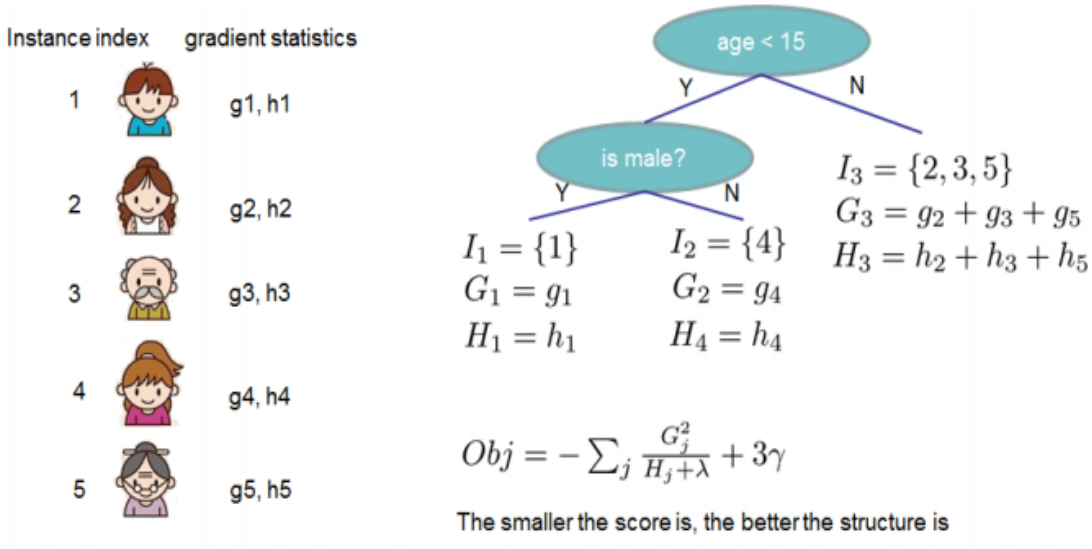
$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

são estatísticas de gradiente de primeira e segunda ordem na função de perda. Podemos remover os termos constantes para obter o seguinte objetivo simplificado na etapa t.

**Equação 3**

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

**Figura 3**




A figura acima apresenta o cálculo da pontuação da estrutura. Precisamos apenas somar as estatísticas de gradiente e gradiente de segunda ordem em cada folha e, em seguida, aplicar a fórmula de pontuação para obter a pontuação de qualidade.

Defina Ij = {i | q (xi) = j} como o conjunto de instâncias da folha j. Podemos reescrever a Equação 3, expandindo Ω da seguinte forma

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

Para uma estrutura fixa q(x), podemos calcular o peso ótimo da folha j por

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

e calcular o valor ideal correspondente por

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

A Equação 6 pode ser usada como uma função de pontuação para medir a qualidade de uma estrutura de árvore q. Essa pontuação é como a pontuação de impureza para avaliar árvores de decisão, exceto que é derivada para uma gama mais ampla de funções de objetivo. A Figura 3 ilustra como essa pontuação pode ser calculada.

Normalmente é impossível enumerar todas as estruturas de árvore possíveis q. Em vez disso, é usado um algoritmo ganancioso que começa em uma única folha e adiciona ramos à árvore de forma iterativa. Suponha que IL e IR sejam os conjuntos de instâncias dos nós esquerdo e direito após a divisão. Deixando I = IL ∪ IR, então a redução da perda após a divisão é dada por

$$\mathcal{L}_{split} = \frac{1}{2} \left[ \frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

Esta fórmula é geralmente usada na prática para avaliar os candidatos divididos.

## 1.3 Métricas de Avaliação

Nesta seção, introduziremos o conjunto de métrica de avaliação utilizadas ao longo deste trabalho.

### Matriz de Confusão

De forma simples a matriz de confusão é uma tabela onde são registrados todos as quatro tipos de classificação do modelo de machine learning. A exemplo, imagine um modelo capaz de realizar uma classificação binária, nesse caso a matriz de confusão seria a seguinte.

| Valor real / Valor previsto | Positivo | Negativo |
|---|---|---|
| Positivo | Verdadeiro Positivo (TP) | Falso Positivo (FP) |
| Negativo | Falso Negativo (FN) | Verdadeiro negativo (TN) |

Como podemos visualizar na imagem acima, as colunas representam o valor real, e como comentado anteriormente, por se tratar de uma classificação binária, nosso resultado só pode ser um entre dois estados, onde neste exemplo seria positivo ou negativo. Com isso em mente, as linhas da matriz representam os valores previstos, ou seja, o estado apontado por nosso modelo de machine learning para classificar uma determinada amostra dos dados. Sendo assim, a matriz de confusão nada mais é do que uma matriz que contabiliza a quantidade de vezes em que o modelo classificou um amostra corretamente e incorretamente, apontando qual classe deveria ser atribuída e enquadrando a classificação entre os quatro tipos especificados, sendo eles:

- ***Verdadeiro Positivo (TP):*** são observações cujo valor real é positivo e o valor previsto é positivo, isto é, o modelo acertou;
- ***Falso Positivo (FP):*** são observações cujo valor real é negativo e o valor previsto é positivo, isto é, o modelo errou;
- ***Falso Negativo (FN):*** são casos em que o resultado correto é positivo, entretanto, o resultado obtido é negativo, isto é, o modelo errou;
- ***Verdadeiro Negativo (TN):*** são casos em que o resultado correto é negativo e o resultado obtido é negativo, isto é, o modelo acertou;

Note que na diagonal principal (os quadrantes (1, 1) e (2, 2)) podemos ver os acertos do modelo e na diagonal secundária (os quadrantes (2, 1) e (1, 2)) podemos ver os erros do modelo. Tais observações são úteis para a efetiva avaliação do modelo, além de servir como base para o cálculo de diversas metrícas de avaliação como as comentadas a seguir.

## Acurácia

Esta é a métrica mais simples. É basicamente o número de acertos (positivos) dividido pelo número total de exemplos. Ela deve ser usada em datasets com a mesma proporção de exemplos para cada classe, e quando as penalidades de acerto e erro para cada classe forem as mesmas. Em problemas com classes desproporcionais, ela causa uma falsa impressão de bom desempenho. Por exemplo, num dataset em que 80% dos exemplos pertençam a uma classe, só de classificar todos os exemplos naquela classe já se atinge uma precisão de 80%, mesmo que todos os exemplos da outra classe estejam classificados incorretamente.

- ***acuracy*** = (TP + TN) / (TP + TN + FP + FN)

## Sensibilidade

Também chamada de recall, é a proporção de casos positivos que foram identificados corretamente. Para calculá-la, toma-se o número de observações que o modelo classificou como positivos corretamente (verdadeiros positivos) e divide-se pelo número total de observações com rótulo positivo (verdadeiros positivos

+ falsos negativos).

- ***sensitivity*** = TP / (TP + FN)

## Especificidade

É a proporção de casos negativos que foram identificados corretamente. Para calculá-la, toma-se o número de observações que o modelo classificou como negativos corretamente (verdadeiros negativos) e divide-se pelo número total de observações com rótulo negativo (verdadeiros negativos + falsos positivos).

- ***specificity*** = TN / (TN + FP)

## Precisão

A precisão é a métrica que indica a quantidade de verdadeiros positivos sobre a soma de todos os valores positivos, ou seja, daqueles que classifiquei como corretos, quantos efetivamente estavam corretos?

- ***precision*** = ((TP / (TP + FP)) + (TN / (TN + FN))) / 2

## Recall

O recall é uma métrica que calcula quantos positivos reais o nosso modelo é capaz de classificar corretamente, rotulando-o como verdadeiro positivo (VP).

- ***recall*** = ((TP / (TP + FN)) + (TN / (TN + FP))) / 2

## F1-Score

O F1-Score é uma média harmônica entre precisão (que, apesar de ter o mesmo nome, não é a mesma coisa) e recall. Ela é muito boa quando você possui um dataset com classes desproporcionais, e o seu modelo não emite probabilidades. Isso não significa que não possa ser usada com modelos que emitem probabilidades, tudo depende do objetivo de sua tarefa de machine learning. Em geral, quanto maior o F1 score, melhor.

- ***f1_score*** = 2 * (precision * recall) / (precision + recall)

## Cohen’s kappa

Kappa é uma medida de avaliação boa para lidar com problemas de desbalanceamento de classes e é definido pela seguinte fórmula:

- ***kappa*** = 1 - (1 - po) / (1 - pe)

Onde po é a concordância observada e pe é a concordância esperada. Basicamente, ele informa o quão melhor o seu classificador está se saindo em relação ao desempenho de um classificador que simplesmente adivinha aleatoriamente de acordo com a frequência de cada classe. O valor resultante é sempre menor ou

igual a 1. Valores menores que 0 indicam que o modelo é inútil. Não existe uma maneira padronizada de interpretar seus valores. Landis e Koch (1977) fornecem uma maneira de caracterizar valores. De acordo com seu esquema, um valor menor que 0 indica nenhuma concordância, 0-0,20 como leve, 0,21-0,40 como razoável, 0,41-0,60 como moderado, 0,61-0,80 como substancial e 0,81-1 como concordância quase perfeita.

### Curva ROC

Trata-se de uma representação gráfica que ilustra o desempenho de um sistema de classificação à medida que seu limiar de discriminação varia. A curva ROC é obtida pela representação da razão, RPV = verdadeiros positivos / total de positivos versus a razão RPF = falsos positivos / total de negativos, para vários valores do limiar de classificação. O RPV é também conhecido como sensibilidade (ou taxa de verdadeiros positivos), e RPF = 1-especificidade ou taxa de falsos positivos. A especificidade é conhecida como taxa de verdadeiros negativos (RVN). A análise ROC fornece ferramentas para selecionar modelos possivelmente ideais (modelos ótimos) e descartar modelos não tão ótimos, independentemente (e antes de especificar) o contexto de custos ou a distribuição de classe. A análise ROC está relacionada de forma direta e natural com a análise de custo/benefício do diagnóstico.

# 2. Visualização e Interpretação de Dados

Basicamente, utilizamos a visualização de dados para facilitar a interpretação das informações para que seja possível realizar inferências e tomar decisões rápidas e seguras, além de simplificar o processo de identificação de padrões e tendências. A seguir são apresentadas diferentes formas de visualização, seguida da análise dos dados.

Como em todo projeto prático de análise de dados, devemos ter em mente quais recursos iremos utilizar para nos ajudar a obter um bom resultado com nossos experimentos. A seguir apresentamos algumas das bibliotecas utilizadas neste capítulo:

- ***Pandas***

  O Pandas é uma biblioteca do Python utilizada para análise e manipulação de dados e nos permite trabalhar de forma rápida e eficiente com arquivos do tipo csv, excel, txt, etc, fornecendo várias maneiras de estruturar os dados.

- ***Matplotlib***

  O matplotlib é uma biblioteca com recursos para a geração de gráficos 2D a partir de arrays no python. Gráficos comuns podem ser criados com alta qualidade a partir de comandos simples o que facilita a vida do desenvolvedor.

- ***Seaborn***

  A biblioteca Seaborn atua em cima do matplotlib e ajuda a melhorar o visual dos gráficos, dando uma aparência mais bem acabada.

In [1]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="darkgrid")
```

Em seguida vamos carregar nosso conjunto de dados. Por se tratar de um arquivo com extensão .csv, podemos utilizar o método "read_csv" do pandas para realizar essa tarefa.

É interessante lembrar que ao carregar o conjunto de dados, dependendo da forma em que os dados são organizados, o pandas pode retornar uma de duas estruturas, entre elas temos, o DataFrame e o TextParser, sendo o DataFrame o mais comum, principalmente quando estamos trabalhando com arquivos com extensão .csv.

In [2]:

```
file = pd.read_csv('./datasets/water.csv')
```

Com o DataFrame pronto, vamos explorar algumas características dos nossos dados. A seguir utilizaremos o método dtypes do nosso DataFrame para identificar com quais tipos de dados iremos trabalhando.

In [3]:

```
file.dtypes
```

Out[3]:

```
ph                float64
Hardness          float64
Solids            float64
Chloramines       float64
Sulfate           float64
Conductivity      float64
Organic_carbon    float64
Trihalomethanes   float64
Turbidity         float64
Potability          int64
dtype: object
```

Note que o dataset escolhido possui apenas valores numéricos. Devido a essa característica, ou seja, sem a presença de dados em formato de texto, nossa análise se torna de certa forma um pouco mais simples, pois não há necessidade necessidade de qualquer processamento para a conversão de valores não numéricos para numéricos, que por sua vez é o único tipo de dado processado pelos modelos da biblioteca Scikit Learn.

Caso tenha interesse procure por **Feature engineering**.

A seguir podemos visualizar parte dos dados presentes em nosso dataframe.

In [4]:

```
file
```

Out[4]:

| | ph | Hardness | Solids | Chloramines | Sulfate | Conductivity | Organic_carb |
|---|---|---|---|---|---|---|---|
| **0** | NaN | 204.890455 | 20791.318981 | 7.300212 | 368.516441 | 564.308654 | 10.3797 |
| **1** | 3.716080 | 129.422921 | 18630.057858 | 6.635246 | NaN | 592.885359 | 15.1800 |
| **2** | 8.099124 | 224.236259 | 19909.541732 | 9.275884 | NaN | 418.606213 | 16.8686 |
| **3** | 8.316766 | 214.373394 | 22018.417441 | 8.059332 | 356.886136 | 363.266516 | 18.4365 |
| **4** | 9.092223 | 181.101509 | 17978.986339 | 6.546600 | 310.135738 | 398.410813 | 11.5582 |
| **...** | ... | ... | ... | ... | ... | ... | |
| **3271** | 4.668102 | 193.681735 | 47580.991603 | 7.166639 | 359.948574 | 526.424171 | 13.8944 |
| **3272** | 7.808856 | 193.553212 | 17329.802160 | 8.061362 | NaN | 392.449580 | 19.9032 |
| **3273** | 9.419510 | 175.762646 | 33155.578218 | 7.350233 | NaN | 432.044783 | 11.0390 |
| **3274** | 5.126763 | 230.603758 | 11983.869376 | 6.303357 | NaN | 402.883113 | 11.1689 |
| **3275** | 7.874671 | 195.102299 | 17404.177061 | 7.509306 | NaN | 327.459760 | 16.1403 |

3276 rows × 10 columns

Ao visualizar os dados podemos perceber que a coluna Potability é uma coluna de dados categóricos e conforme descrito na especificação da base de dados pode receber o valor 1 (um) ou 0 (zero), classificando a amostra da água como potável ou não potável.

De forma resumida, existem basicamente dois tipos de classificação, a binária e a multiclasse. Uma classificação é considerada binária quando seu objetivo é classificar uma amostra em apenas duas classes distintas, como no caso deste trabalho, onde cada amostra de água pode ser classificada como 0 (zero) ou 1 (um). Em contrapartida, temos a classificação multiclasse que ocorre quando temos mais de duas classes alvo, como, "gato", "cão" e "papagaio".

Independente do tipo de classificação, existem algumas etapas a serem seguidas para realizar uma boa análise exploratória. Detre elas podemos citar:

- Realizar um exame gráfico da natureza das variáveis e uma análise descritiva que permita quantificar alguns aspectos gráficos dos dados;
- Realizar um exame gráfico das relações entre as variáveis analisadas e uma análise descritiva que quantifique o grau de inter-relação entre elas;
- Identificar os possíveis casos atípicos (outliers);
- Avaliar, se for necesário, a presença de dados ausentes (missing).

Uma observação que podemos fazer nesse ponto é que ao visualizar os dados podemos tentar identificar valores ausentes ou outliers. Embora possível, na maioria dos casos essa atividade torna-se inviável devido ao tamanho do conjunto de dados utilizado. No exemplo acima, mesmo sendo um grande conjunto de dados, é possível afirmar que existem valores faltantes (NaN) entre os dados, entretanto, não conseguimos inferir qual a quantidade ou em quais colunas isso é mais frequente. Em contrapartida, se considerarmos outliers a identificação torna-se inviável.

Neste momento, façamos uma cópia do nosso dataframe. A partir daqui, utilizaremos a cópia como entrada para os métodos de visualização apresentados a seguir.

Decidimos utilizar o conjunto de dados em sua forma original para que seja possível explorar os recursos gráficos de forma mais completa, abordando o máximo de aspectos possíveis.

In [5]:

```
df = file.copy()
```

## 2.1 Scatter Plot e Histograma

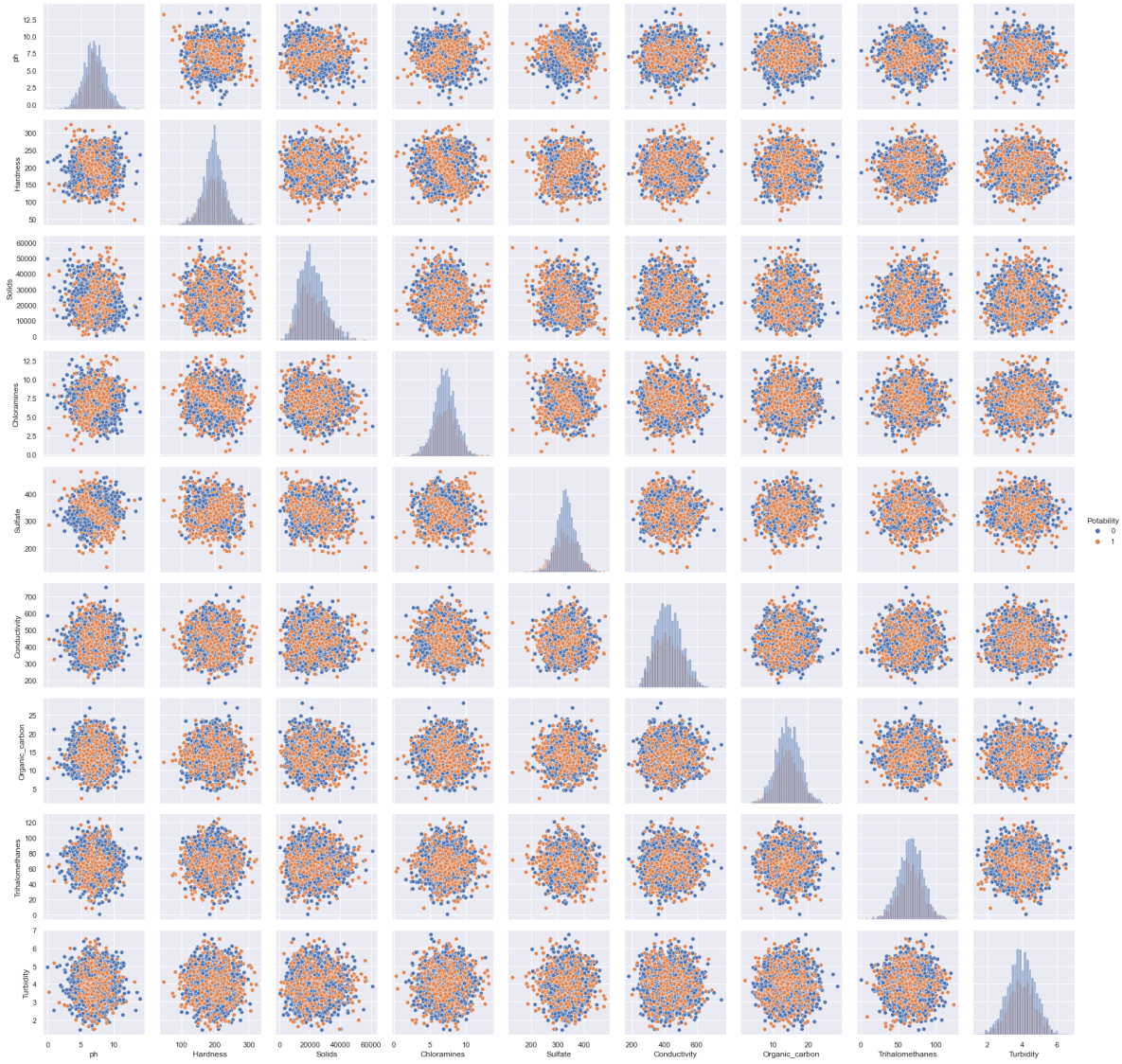
Em nosso primeiro exemplo, utilizaremos dois tipos de gráfico para representar os dados, o primeiro deles é o scatter plot que ocupa maior parte das posições da matriz de gráficos apresentada abaixo, com exceção apenas da diagonal principal da matriz, a qual é preenchida com histogramas. Em geral, considerando esta situação, ambos os gráficos estão sendo utilizados para representar as relações e diferenças entre as variáveis numéricas presentes no dataset. Notem que em nosso exemplo utilizamos o atributo Potability a fim de realizar uma análise mais abrangente.

In [6]:

```
g = sns.PairGrid(df, hue='Potability')
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
g.add_legend()
```

Out[6]:

```
<seaborn.axisgrid.PairGrid at 0x1808edf3040>
```




Como podemos ver nos gráficos acima, as representações dos dados em cor laranja são referentes ao valor 1 (potável), enquanto que as representações em azul representam o valor 0 (não potável).

Uma das primeiras observações que podemos fazer, referente ao scatter plot, é com relação a sua distribuição. Notem que ela não apresenta uma relação linear (positiva ou negativa) e nem mesmo uma relação não linear. Sendo assim, podemos inferir que a relação entre as variáveis deste dataset é neutra.

Além disso, podemos notar que existem valores discrepantes em nosso conjunto de dados, podendo ser valores nulos ou até mesmo valores muito acima ou muito abaixo do normal. Isso é identificado pelos pontos que estão um pouco mais distantes do grupo de pontos central em cada scatter plot.

Com relação aos histogramas podemos dizer que a distribuição dos dados para cada atributo é bem semelhante quando consideramos ambas as classes (1 e 0), entretanto, os valores para as variáveis

referentes a classe 0 (não potável) são mais frequentes. Visualmente falando podemos inferir que há um desbalanceamento de classes em nosso conjunto de dados, sendo a classe 0 (não potável) a classe predominante.

Para comprovar o que queremos dizer vamos fazer o seguinte:

Primeiro vamos verificar qual o total de amostras do nosso conjunto. Como podemos ver, ele é composto por 3276 amostras com 10 atributos/colunas por amostra.

In [7]:

```
df.shape
```

Out[7]:

```
(3276, 10)
```

Em seguida, vamos utilizar um Pie chart para verificar a quantidade de amostras para cada classe.

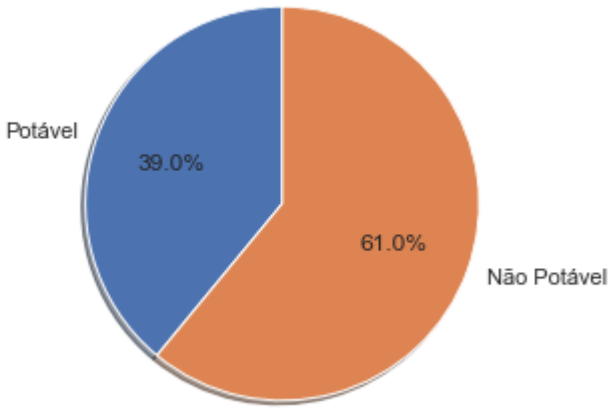
In [8]:

```
labels = 'Potável', 'Não Potável'
sizes = [len(df[df['Potability'] == 1]), len(df[df['Potability'] == 0])]

fig1, ax1 = plt.subplots()

ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')

plt.show()
```




Como podemos perceber, o conjunto de dados é composto por 61% (1.998 classes) de valor 0 (não potável) e 39% (1.278 classes) de valor 1 (potável). Com isso, podemos confirmar o desbalanceamento entre classes em nosso conjunto de dados.

Notem que essa característica deve ser levada em consideração ao preparar os dados, pois, se desbalanceados, podem enviesar o modelo e torná-lo tendencioso a uma classe específica. Vale lembrar que não é somente o desbalanceamento entre classes que pode tornar um modelo enviesado, esse assunto é muito mais complexo que isso e pode ser causado por características quase que imperceptíveis dos dados, o que os tornam tão complexos.

Embora tenhamos utilizado o histograma para supor o desbalanceamento entre classes, as vezes pode ser díficil fazer essa suposição com base na distibuição dos dados. Sendo assim, podemos partir direto para o somatório de ocorrências ou buscar outras abordagens gráficas.

Além das informações acima podemos utilizar histogramas para verificar os seguintes itens:

- Qual é o centro da distribuição?
- A distribuição normalmente contém observações entre quais valores?
- Qual é o ponto de máximo e o ponto de mínimo?
- Entre outros.

## 2.2 Box Plot

A seguir podemos visualizar um box plot ou gráfico de caixa, conforme apresentamos no capítulo 1

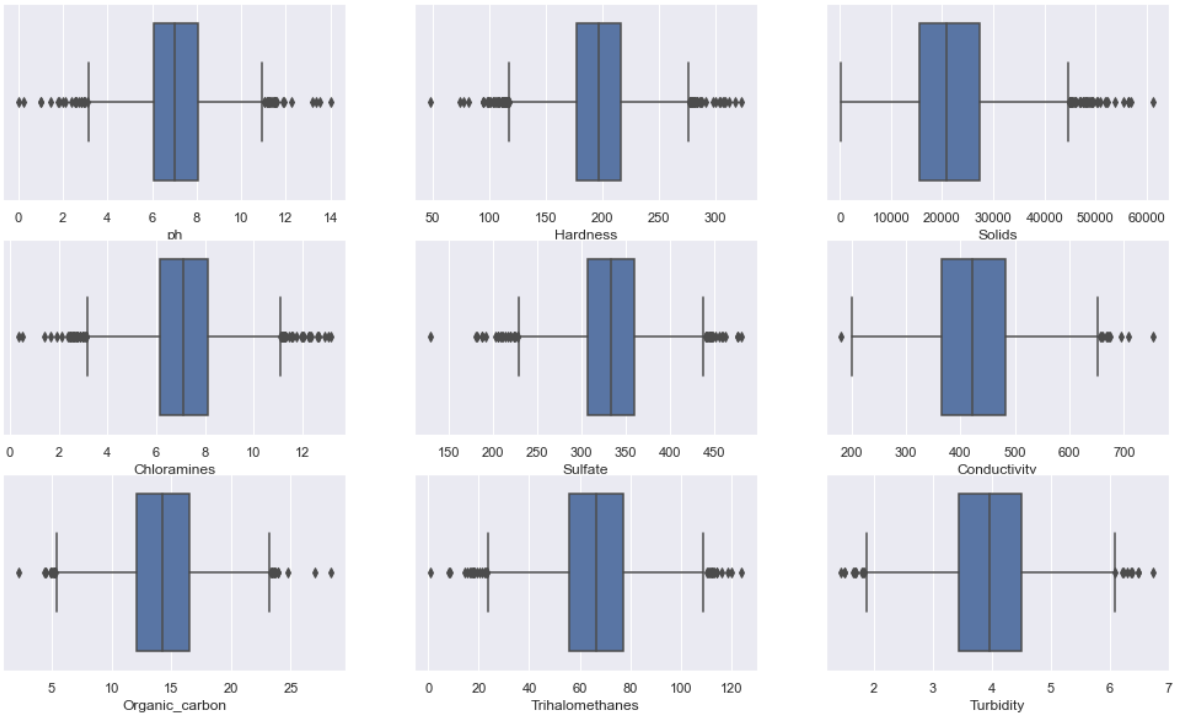
In [9]:

```
fig, ((ax1, ax2, ax3), (ax4, ax5, ax6), (ax7, ax8, ax9)) = plt.subplots(3, 3, figsize=(17,

sns.boxplot(x=df["ph"], ax=ax1)
sns.boxplot(x=df["Hardness"], ax=ax2)
sns.boxplot(x=df["Solids"], ax=ax3)
sns.boxplot(x=df["Chloramines"], ax=ax4)
sns.boxplot(x=df["Sulfate"], ax=ax5)
sns.boxplot(x=df["Conductivity"], ax=ax6)
sns.boxplot(x=df["Organic_carbon"], ax=ax7)
sns.boxplot(x=df["Trihalomethanes"], ax=ax8)
sns.boxplot(x=df["Turbidity"], ax=ax9)
```

Out[9]:

```
<AxesSubplot:xlabel='Turbidity'>
```




Com base no gráfico acima, podemos inferir a presença de outliers ou valores ausentes em nosso conjunto de

dados. Notem que em todos os exemplos existem valores que excedem o intervalo entre o valor mínimo e o valor máximo. Além desta informação, podemos visualizar diversas outras, como por exemplo a mediana ou tendência central, ou seja, o valor central do conjunto de dados ordenado, ou mesmo, o que separa os 50% menores dos 50% maiores valores.

Notem que o box plot é muito útil e traduz diversas informações relevantes dos dados. Contudo, apresenta um nível de detalhamento um pouco menor quando comparado com outras técnicas de plotagem. Considerando a presença de outliers em nosso conjunto de dados, seria interessante visualizar um gráfico capaz de trazer mais detalhes com relação a esses valores. A seguir, apresentamos uma técnica semelhante ao box plot, porém com um maior nível de detalhamento.

## 2.3 Violin Plot

A seleção do violin plot, como uma das técnicas de visualização desse trabalho, justifica-se pela capacidade em apresentar a distribuição dos outliers, o que nos permite inferir algumas informações, como por exemplo, a região onde tem maior concentração desses valores.

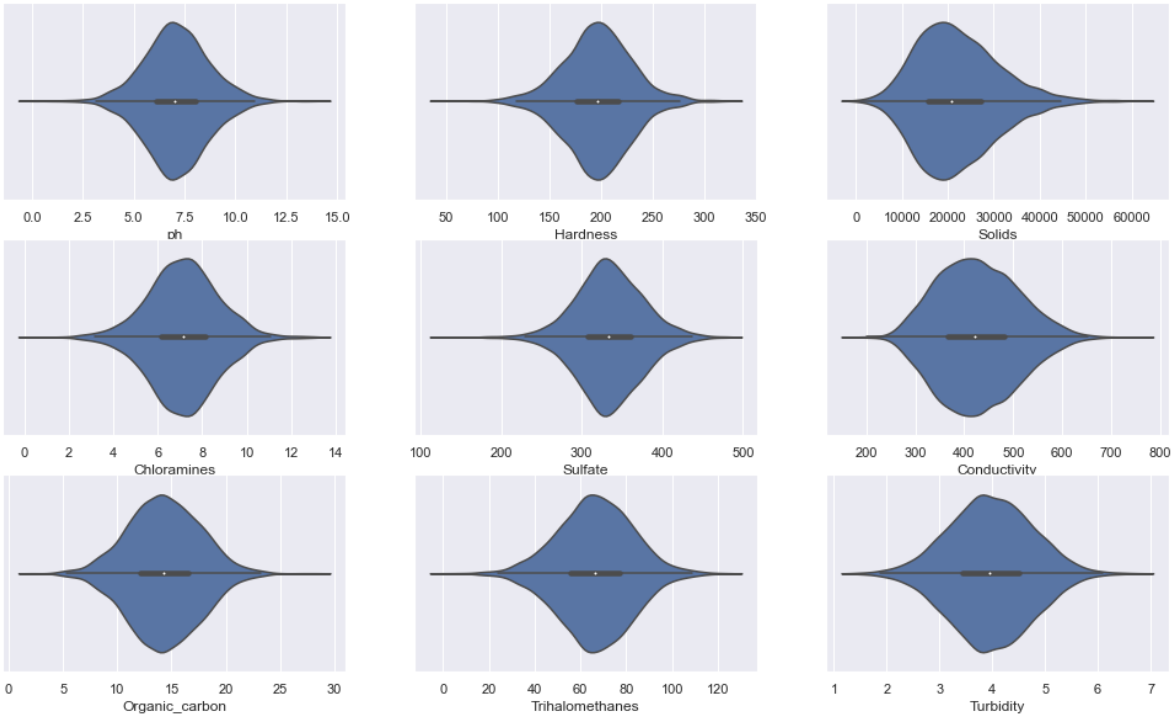
In [10]:

```
fig, ((ax1, ax2, ax3), (ax4, ax5, ax6), (ax7, ax8, ax9)) = plt.subplots(3, 3, figsize=(17,

sns.violinplot(x=df["ph"], ax=ax1)
sns.violinplot(x=df["Hardness"], ax=ax2)
sns.violinplot(x=df["Solids"], ax=ax3)
sns.violinplot(x=df["Chloramines"], ax=ax4)
sns.violinplot(x=df["Sulfate"], ax=ax5)
sns.violinplot(x=df["Conductivity"], ax=ax6)
sns.violinplot(x=df["Organic_carbon"], ax=ax7)
sns.violinplot(x=df["Trihalomethanes"], ax=ax8)
sns.violinplot(x=df["Turbidity"], ax=ax9)
```

Out[10]:

```
<AxesSubplot:xlabel='Turbidity'>
```



Como podemos observar, o violin plot apresenta as mesmas informações que o box plot. A principal diferença nesse gráfico é que podemos visualizar a distribuição dos dados, através do qual é possível observar diferentes picos nos dados, sua posição e amplitude relativa. Dependendo da aplicação podemos utilizá-lo para comparar dois conjuntos de dados e tomar melhores decisões.

Em comparação com o box plot, essa representação do violin plot é consideravelmente superior. Mesmo o box plot sendo ótimo e muito utilizado ele pode ser enganoso em alguns momentos. Isso ocorre pois eles não são afetados pela distribuição dos dados, ou seja, quando os dados mudam e eles conseguem manter seus resumos estatísticos, sua representação gráfica permanece a mesma.

Outra vantagem do violin plot é que ele é fácil de ler, isso ocorre pois é exatamente como ler um gráfico de densidade, a parte mais grossa significa que os valores nessa seção do violino têm frequência mais alta, e a parte mais fina indica frequência mais baixa.

## 2.4 Heatmap

Correlação é qualquer relação dentro de uma ampla classe de relações estatísticas que envolva dependência entre variáveis. Por exemplo, a correlação do peso da carga com o valor do frete (geralmente linear positiva). Embora seja comumente denotada como a medida de relação entre duas variáveis aleatórias, a correlação não implica causalidade (causa e efeito). Em alguns casos, a correlação não identifica dependência entre as variáveis. Em geral, há pares de variáveis que apresentam forte dependência estatística, mas que possuem correlação nula. Para estes casos, são utilizadas outras medidas de dependência.

Em nosso exemplo utilizaremos o método "corr" do pandas para obter a correlação entre as variáveis do nosso dataset. Em seguida, utilizaremos um Heatmap para melhor apresentar essas informações.

Notem que não eliminamos os dados faltantes do nosso dataset, isso interferiria de forma negativa no processo de obtenção da correlação? A resposta é não, segundo a documentação da biblioteca o método desconsidera esses valores de forma automática.

In [11]:

```
df.corr(method='spearman')
```

Out[11]:

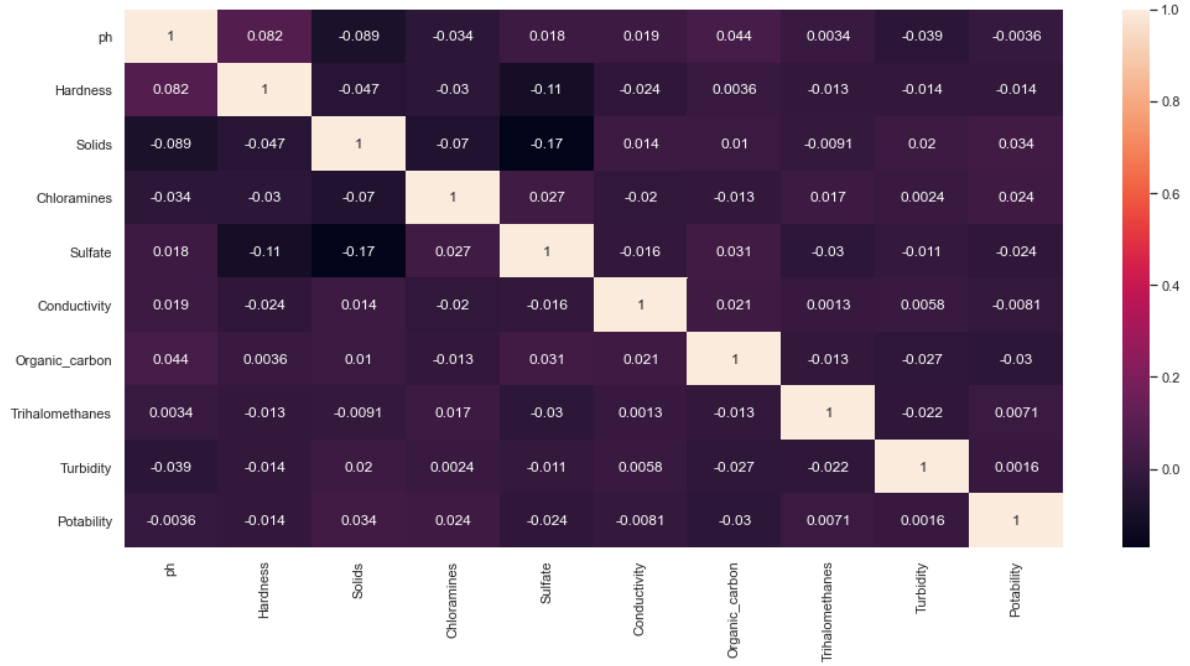
| | ph | Hardness | Solids | Chloramines | Sulfate | Conductivity | Organic |
|---|---|---|---|---|---|---|---|
| **ph** | 1.000000 | 0.116266 | -0.075239 | -0.042486 | 0.023552 | 0.016920 | ( |
| **Hardness** | 0.116266 | 1.000000 | -0.052584 | -0.024556 | -0.094803 | -0.032884 | ( |
| **Solids** | -0.075239 | -0.052584 | 1.000000 | -0.055037 | -0.154223 | 0.021089 | ( |
| **Chloramines** | -0.042486 | -0.024556 | -0.055037 | 1.000000 | 0.037229 | -0.016928 | -( |
| **Sulfate** | 0.023552 | -0.094803 | -0.154223 | 0.037229 | 1.000000 | -0.022277 | ( |
| **Conductivity** | 0.016920 | -0.032884 | 0.021089 | -0.016928 | -0.022277 | 1.000000 | ( |
| **Organic_carbon** | 0.043534 | 0.003340 | 0.017881 | -0.011910 | 0.019590 | 0.021311 | |
| **Trihalomethanes** | 0.004672 | -0.011667 | -0.019618 | 0.017784 | -0.031157 | -0.004094 | -( |
| **Turbidity** | -0.049050 | -0.012855 | 0.028470 | -0.007909 | -0.018826 | 0.010342 | -( |
| **Potability** | -0.002181 | -0.010606 | 0.026234 | 0.024980 | -0.016676 | -0.010386 | -( |

Notem que no código fonte acima, utilizamos o coeficiente de correlação de Spearman. Embora a correlação de Pearson (técnica padrão do método corr do pandas) seja igual a correlação de Spearman entre os valores de postos de duas variáveis, a correlação de Pearson avalia relações lineares, enquanto a correlação de Spearman avalia relações monótonas, sejam elas lineares ou não como é o caso do nosso conjunto de dados.

Se não houver valores de dados repetidos, uma correlação de Spearman perfeita de +1 ou -1 ocorre quando cada uma das variáveis é uma função monótona perfeita da outra.

In [12]:

```
f = plt.figure(figsize=(17, 8))
ax = sns.heatmap(df.corr(), annot=True)
```



Intuitivamente, a correlação de Spearman entre duas variáveis será alta quando as observações tiverem uma classificação semelhante (ou idêntica no caso da correlação igual a 1) entre as duas variáveis, isto é, a posição relativa das observações no interior da variável, e baixa quando observações tiverem uma classificação dessemelhante (ou completamente oposta no caso da correlação igual a -1) entre as duas variáveis.

Com base nisso, podemos concluir que a correção entre as variáveis do nosso dataset é baixa, e podemos fazer as seguintes observações quanto aos coeficientes apresentados:

- ***Coeficiente positivo:*** Y tende a aumentar quando X aumenta;
- ***Coeficiente negativo:*** Y tende a diminuir quando X aumenta;
- ***Coeficiente igual a zero:*** não há tendência de que Y aumente ou diminua quando X aumenta.

Vale lembrar que, embora tenhamos alguns valores positivos ou negativos, essas representações são bem discretas e próximas de zero, o que significa que as tendências apresentadas são mínimas.

# 3. Aprendizado de Máquina

Neste capítulo trabalharemos com desenvolvimento, avaliação e interpretação de modelos de aprendizado de máquina e para isso iremos fazer uso dos seguintes recursos da linguagem python.

- ***Scikit Learn***

  O scikit-learn é uma biblioteca de aprendizado de máquina para a linguagem de programação Python. Ela inclui vários algoritmos de classificação, regressão e agrupamento, incluindo máquinas de vetores de suporte, florestas aleatórias, gradient boosting, k-means e DBSCAN, e é projetada para interagir com as bibliotecas Python numéricas e científicas NumPy e SciPy.

- ***Numpy***

  O NumPy é uma poderosa biblioteca Python que é usada principalmente para realizar cálculos em Arrays Multidimensionais. O NumPy fornece um grande conjunto de funções e operações de biblioteca que ajudam os programadores a executar facilmente cálculos numéricos. Esses tipos de cálculos numéricos são amplamente utilizados em diversas tarefas.

- ***Math***

  Este módulo fornece acesso às funções matemáticas definidas na linguagem python.

Para não estender ainda mais este trabalho, deixaremos de lado algumas técnicas e tratativas que embora não sejam abordadas, gostaria de ter implementado, tenho certeza que agregaria maior valor ao trabalho.

- ***Desbalanceamento entre classes***

  Realizar a implementação e avaliação dos resultados de diferentes técnicas utilizadas para tratar o desbalanceamento entre classes.

- ***Tratamento de valores fora da curva (outliers)***

Neste trabalho tratamos valores ausentes, deixando de lado outliers, pois, para tratá-los, é necessário um conhecimento especializado sobre o conjunto de dados, para então verificar se foram coletados corretamente e fazem sentido.

In [13]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
import numpy as np
from math import pi
```

## 3.1 Pré-processamento

O pré-processamento é um passo importante para a análise de dados. Em sua maioria, os processos de aquisição dos dados são fracamente controlados, o que pode resultar em valores fora dos padrões para aquela observação. A exemplo, podemos citar algumas combinações de dados impossíveis. Considere dois atributos de um dataset, que são, "sexo" e "grávida", um exemplo de valor impossível seria obtido se o atributo "sexo" apresentasse o valor "masculino" e o atributo "grávida" apresentasse o valor "sim". Além deste exemplo, os dados poderiam estar ausentes e provocar erros durante o processamento. A seguir, pré-processaremos nosso conjunto de dados de três maneiras diferentes para em seguida utilizá-las em nossos modelos, a fim de comparar os resultados obtidos.

Vale lembrar que devido às características do nosso conjunto de dados a etapa de pré-processamento não se torna uma tarefa excessivamente complexa como em alguns casos, sendo assim, abordaremos algumas tratativas básicas.

Antes de iniciarmos vamos verificar qual o tamanho do nosso conjunto de dados em seu estado inicial.

In [14]:

```
file.shape
```

Out[14]:

```
(3276, 10)
```

Como podemos visualizar ele é composto por 3.276 linhas e 10 colunas.

A célula abaixo apresenta o somatório, por coluna, de todos os valores faltantes presentes em nosso dataset.

In [15]:

```
file.isna().sum()
```

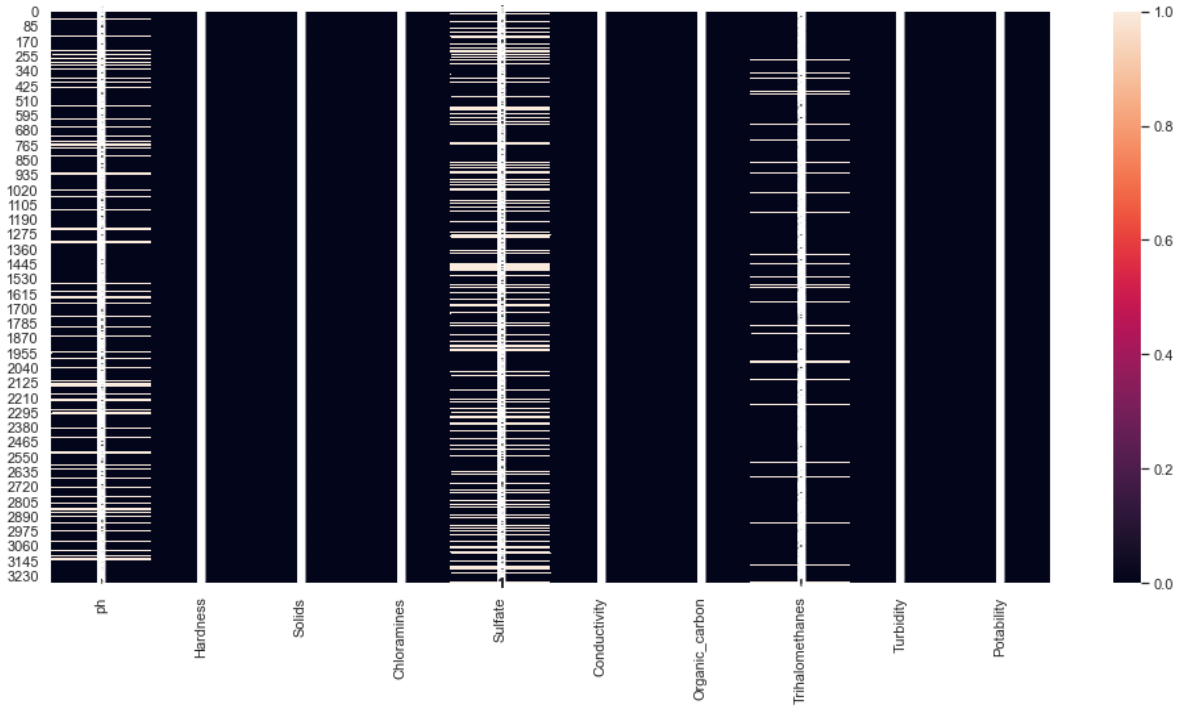
Out[15]:

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

Outra forma de visualizar os dados ausentes é através de um heatmap, onde podemos identificar as colunas com dados ausentes, como apresentado no gráfico abaixo.

In [16]:

```
f = plt.figure(figsize=(17, 8))
ax = sns.heatmap(file.isna(), annot=True)
```




## Eliminando valores ausentes

Dentre as técnicas utilizadas para o tratamento de dados ausentes, a primeira que selecionamos foi a de eliminação, nessa técnica, toda amostra que possui pelo menos um valor ausente é eliminada do conjunto de dados. O código fonte apresentado abaixo é responsável por aplicar esse filtro nos dados.

In [17]:

```
eliminating_missing_values_ds = file.copy()
eliminating_missing_values_ds.dropna(inplace=True)
```

Para verificar se os dados ausentes foram efetivamente eliminados, podemos realizar um novo somatório de valores ausente para cada coluna, conforme apresentado abaixo. Note que desta vez nenhuma coluna possui valor ausente.

In [18]:

```
eliminating_missing_values_ds.isna().sum()
```

Out[18]:

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

Um ponto negativo que podemos observar com relação a essa técnica é que, ao eliminar amostras, estamos reduzindo o tamanho do nosso conjunto de dados base, e assim impactando negativamente no aprendizado e capacidade de generalização dos nossos modelos. Além disso, ao eliminar os dados de forma aleatória como neste método, podemos tornar nosso conjunto de dados ainda mais enviesado.

A seguir podemos notar que o número de amostras do nosso conjunto de dados teve uma redução considerável após a aplicação do método.

In [19]:

```
eliminating_missing_values_ds.shape
```

Out[19]:

```
(2011, 10)
```

Por se tratar de um método de eliminação de dados, devemos eliminá-los considerando todo o conjunto, para posteriormente realizar a separação dos dados entre conjunto treino e teste, conforme especificação do método holdout. No código fonte a seguir, utilizamos nosso conjunto de dados resultante do processo anterior e separamos as variáveis preditoras das variáveis alvo.

In [20]:

```
y = eliminating_missing_values_ds['Potability']
x = eliminating_missing_values_ds.drop('Potability', axis=1)
```

Em seguida, utilizamos o método "train_test_split" para separar o conjunto de dados (considerando as variáveis preditoras (x) e variável alvo (y)) entre treino e teste. Notem que separamos 80% do conjunto de dados para treinamento e 20% para teste, conforme especificado no parâmetro "test_size". Nossa intenção é

utilizar essa mesma proporção de separação em todos os experimentos.

Observação: Atenção ao nome dos conjuntos de dados, eles serão utilizados para identificar o conjunto de dados e a técnica de tratamento de dados faltantes que está sendo aplicada no experimento.

In [21]:

```
x_training_first_method,x_test_first_method,y_training_first_method, y_test_first_method =
```

Por se tratar de técnicas que realizam a substituição de dados ausentes, sem perder a amostra, as técnicas apresentadas a seguir necessitam que antes de substituir os valores, seja separado o conjunto de dados de teste. Entretanto, o conjunto de dados de teste deve ser composto pelas amostras disponíveis no dataset que não possuem valor ausente, para que após a substituição, as amostras com valores ausentes façam parte exclusivamente do conjunto de dados de treinamento. Para isso implementamos o código a seguir.

In [22]:

```
# 1ª Parte
records_null =  file[(file['ph'].isnull() == True) | (file['Sulfate'].isnull() == True) | (
records_not_null = file[(file['ph'].isnull() == False) & (file['Sulfate'].isnull() == False

# 2ª Parte
y = records_not_null['Potability']
x = records_not_null.drop('Potability', axis=1)

# 3ª Parte
x_training_aux, x_test_aux, y_training_aux, y_test_aux = train_test_split(x, y, test_size=0

# 20% de 3.276 é igual a 655 valor aredondado (o que deve representar o tamanho do conjunto
print('X Teste: ', len(x_test_aux), '- Y Teste: ', len(y_test_aux))

# 4ª Parte
x_test = x_test_aux
y_test = y_test_aux
```

```
X Teste:  655 - Y Teste:  655
```

Notem que na 1ª parte, separamos as amostras que possuem valores ausentes das amostras que não possuem valores ausentes.

Na 2ª parte, realizamos a separação de variáveis preditoras (x) e variável alvo (y).

Na 3ª Parte, para obter os dados de forma aleatória, considerando o conjunto de dados sem valores ausentes, utilizaremos novamente o método "train_test_split". Para essa situação, atribuímos ao parâmetro "test_size" o valor 0.3257, nosso intuito com esse valor é manter o conjunto de dados de teste com a mesma proporção que utilizamos na técnica anterior, ou seja, 20%.

Na 4ª parte apenas atribuímos os dados para teste em suas respectivas variáveis, para que sejam utilizadas nos experimentos deste trabalho.

Após definir nosso conjunto de dados de teste, devemos reunir novamente os dados que sobraram, para então formar o conjunto de dados de treinamento, onde serão realizadas as substituições de valores ausentes. Para isso, reunimos as amostras que sobraram do procedimento anterior com as amostras que continham valores ausentes (separadas anteriormente).

In [23]:

```
 df_training = x_training_aux
df_training = df_training.assign(Potability=y_training_aux)

my_list = []

# Incluindo os dados de 'df_training' na lista
for lista in df_training.values.tolist():
    my_list.append(lista)

# Incluindo os dados do dataframe 'records_null'
for lista in records_null.values.tolist():
    my_list.append(lista)

# Criando um dataframe com os registros que sobraram entre os não nulos e os registros que
df_training_base = pd.DataFrame(my_list, columns=['ph', 'Hardness', 'Solids', 'Chloramines'

df_training_base
```

Out[23]:

| | ph | Hardness | Solids | Chloramines | Sulfate | Conductivity | Organic_carl |
|---|---|---|---|---|---|---|---|
| **0** | 10.268415 | 225.074218 | 14694.420625 | 6.722843 | 300.763772 | 353.630913 | 13.553 |
| **1** | 7.005230 | 219.921997 | 26597.586447 | 7.351021 | 333.583723 | 379.598806 | 13.284 |
| **2** | 7.609521 | 223.288394 | 40879.789573 | 7.030699 | 310.632450 | 370.270880 | 14.375 |
| **3** | 7.817901 | 221.089708 | 13742.145965 | 6.373737 | 287.698481 | 460.599214 | 12.704 |
| **4** | 5.039407 | 194.404170 | 19336.608073 | 7.194765 | 339.232126 | 515.807182 | 10.728 |
| **...** | ... | ... | ... | ... | ... | ... | |
| **2616** | 8.372910 | 169.087052 | 14622.745494 | 7.547984 | NaN | 464.525552 | 11.083 |
| **2617** | 7.808856 | 193.553212 | 17329.802160 | 8.061362 | NaN | 392.449580 | 19.903 |
| **2618** | 9.419510 | 175.762646 | 33155.578218 | 7.350233 | NaN | 432.044783 | 11.039 |
| **2619** | 5.126763 | 230.603758 | 11983.869376 | 6.303357 | NaN | 402.883113 | 11.168 |
| **2620** | 7.874671 | 195.102299 | 17404.177061 | 7.509306 | NaN | 327.459760 | 16.140 |

2621 rows × 10 columns

Com nosso dataset de treinamento pronto, vamos separar as variáveis preditoras da variável alvo, isso é feito pois as substituições de valores ausentes só é aplicada às variáveis preditoras.

In [24]:

```
aux_df = df_training_base.copy()

y_training = aux_df['Potability']
x = aux_df.drop('Potability', axis=1)

y_training = y_training.astype({"Potability": int})
```

A seguir utilizaremos o "SimpleImputer", da biblioteca scikit-learn para aplicar ao conjuto de dados nossas técnicas de substituição.

Antes de iniciar, podemos verificar que o conjunto de dados resultante, possui os dados ausentes, conforme mencionado anteriormente.

In [25]:

```
x.isna().sum()
```

Out[25]:

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

Além disso, podemos verificar também a quantidade de amostras do dataset.

In [26]:

```
x.shape
```

Out[26]:

```
(2621, 9)
```

Conforme podemos visualizar temos 2.621 amostras.

## Média

A primeira técnica de substituição aplicada é a média, que substitui os valores ausentes de uma coluna pela média desses valores. A seguir, apresentamos o código fonte utilizado.

Notem que no final de cada técnica, geramos um dataframe com o conjunto de treinamento já atualizado. Nesse caso, devemos lembrar que o conjunto de testes já foi estruturado anteriormente, sendo assim, só precisamos gerar os conjuntos de treinamento de acordo com a técnica utilizada (nesse caso a média).

In [27]:

```
from sklearn.impute import SimpleImputer

aux_x = x.copy()

imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
imp_mean.fit(aux_x)
my_array = imp_mean.transform(aux_x)

aux_x = pd.DataFrame(my_array, columns=['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate

x_training_second_method = aux_x
```

Para verificarmos se os dados ausentes foram realmente substituídos, podemos novamente quantificar as ocorrências desses valores para cada coluna, conforme apresentado abaixo. Notem que a quantidade para cada coluna é zero, ou seja, não há valores ausentes.

In [28]:

```
x_training_second_method.isna().sum()
```

Out[28]:

```
ph                0
Hardness          0
Solids            0
Chloramines       0
Sulfate           0
Conductivity      0
Organic_carbon    0
Trihalomethanes   0
Turbidity         0
dtype: int64
```

Logo abaixo, podemos ver que a quantidade de registros permanece a mesma.

In [29]:

```
x_training_second_method.shape
```

Out[29]:

```
(2621, 9)
```

## Mediana

A terceira técnica de substituição aplicada é a mediana, que substitui os valores ausentes de uma coluna pela mediana desses valores, ou seja, o valor central de um conjunto de dados ordenado. A seguir, apresentamos o código fonte utilizado.

Notem que no final dessa técnica, apresentamos o dataframe com o conjunto de treinamento já atualizado. Além disso, a quantidade de registros permanece igual ao apresentado anteriormente.

In [30]:

```
aux_x = x.copy()

imp_mean = SimpleImputer(missing_values=np.nan, strategy='median')
imp_mean.fit(aux_x)
my_array = imp_mean.transform(aux_x)

x_training_third_method = pd.DataFrame(my_array, columns=['ph', 'Hardness', 'Solids', 'Chlo

x_training_third_method
```

Out[30]:

| | ph | Hardness | Solids | Chloramines | Sulfate | Conductivity | Organic_carl |
|---|---|---|---|---|---|---|---|
| **0** | 10.268415 | 225.074218 | 14694.420625 | 6.722843 | 300.763772 | 353.630913 | 13.553 |
| **1** | 7.005230 | 219.921997 | 26597.586447 | 7.351021 | 333.583723 | 379.598806 | 13.284 |
| **2** | 7.609521 | 223.288394 | 40879.789573 | 7.030699 | 310.632450 | 370.270880 | 14.375 |
| **3** | 7.817901 | 221.089708 | 13742.145965 | 6.373737 | 287.698481 | 460.599214 | 12.704 |
| **4** | 5.039407 | 194.404170 | 19336.608073 | 7.194765 | 339.232126 | 515.807182 | 10.728 |
| **...** | ... | ... | ... | ... | ... | ... | |
| **2616** | 8.372910 | 169.087052 | 14622.745494 | 7.547984 | 332.958794 | 464.525552 | 11.083 |
| **2617** | 7.808856 | 193.553212 | 17329.802160 | 8.061362 | 332.958794 | 392.449580 | 19.903 |
| **2618** | 9.419510 | 175.762646 | 33155.578218 | 7.350233 | 332.958794 | 432.044783 | 11.039 |
| **2619** | 5.126763 | 230.603758 | 11983.869376 | 6.303357 | 332.958794 | 402.883113 | 11.168 |
| **2620** | 7.874671 | 195.102299 | 17404.177061 | 7.509306 | 332.958794 | 327.459760 | 16.140 |

2621 rows × 9 columns

Para confirmar se os dados ausentes foram efetivamente substituídos, vamos novamente contabilizar esses valores para cada uma das colunas do nosso dataframe. Notem que a contagem dos valores estão zeradas, ou seja, foram substituídos.

In [31]:

```
x_training_third_method.isna().sum()
```

Out[31]:

```
ph                0
Hardness          0
Solids            0
Chloramines       0
Sulfate           0
Conductivity      0
Organic_carbon    0
Trihalomethanes   0
Turbidity         0
dtype: int64
```

# 3.2 Classificadores

Nesta seção, apresentamos toda a codificação por trás dos classificadores implementados. Para os experimentos, selecionamos a média como técnica para o tratamento de valores ausentes, e com base nela, implementamos os classificadores, Random Forest, K-Nearest Neighbors (KNN), Support Vector Machine (SVM) e Gradient Tree Boosting. Contudo, para fins de curiosidade, alguns dos resultados obtidos com as demais técnicas são apresentados no apêndice 1 ao final deste trabalho.

Além do que foi desenvolvido, gostaria de ressaltar que as técnicas a seguir são um bom começo para melhorar os resultados obtidos com os classificadores deste trabalho. Podemos considerá-los como itens a serem desenvolvidos em trabalhos futuros.

- ***K-Fold Cross Validation***

  Embora neste trabalho tenhamos abordado somente o método holdout, seria relevante a implementação e validação do método k-fold.

- ***Ajuste Fino dos Modelos***

  Além disso, seria interessante realizar um ajuste fino nos modelos, de tal forma a obter um melhor desempenho na classificação das amostras de água.

In [32]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import recall_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

In [33]:

```
vet_acuracy = []
vet_sensitivity = []
vet_specificity = []
vet_precision = []
vet_recall = []
vet_f1_score = []
vet_cohen_kappa = []
```

## Random Forest

In [34]:

```
random_forest_second_model = RandomForestClassifier(n_estimators=200)
random_forest_second_model.fit(x_training_second_method, y_training)
```

Out[34]:

```
RandomForestClassifier(n_estimators=200)
```

In [35]:

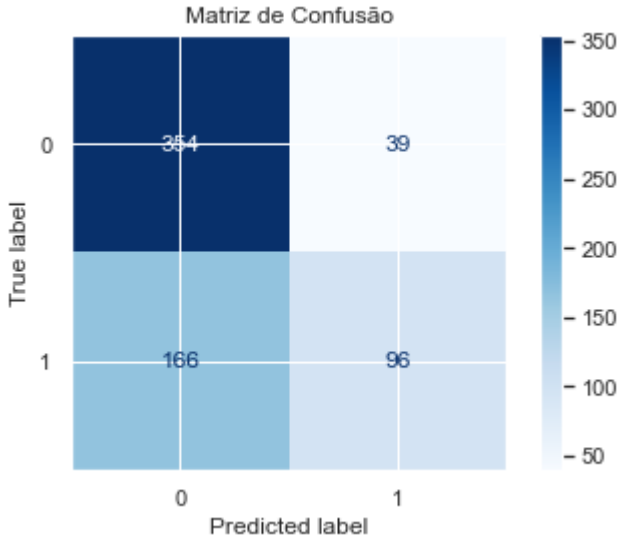
```
random_forest_second_model_predictions = random_forest_second_model.predict(x_test)
```

- ***Avaliação do modelo***

- Matriz de confusão

In [36]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(random_forest_second_model, x_test, y_test, cmap=plt.cm.Blues,
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [37]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [38]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
vet_acuracy.append(acuracy)
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [39]:

```
sensitivity = true_positives / (true_positives + false_negatives)
vet_sensitivity.append(sensitivity)
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [40]:

```
specificity = true_negatives / (true_negatives + false_positives)
vet_specificity.append(specificity)
```

- **Precisão**

In [41]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
vet_precision.append(precision)
```

- ***Recall***

In [42]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
vet_recall.append(recall)
```

- ***F1-Score***

In [43]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0

vet_f1_score.append(f1_score)
```

- ***Cohen's kappa***

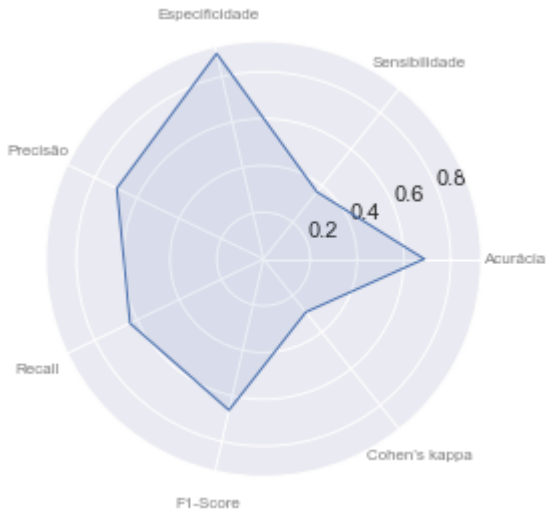
In [44]:

```
cohen_kappa = cohen_kappa_score(y_test, random_forest_second_model_predictions)
vet_cohen_kappa.append(cohen_kappa)
```

- ***Radar Plot***

In [45]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




## KNN (K-Nearest Neighbors)

In [47]:

```
knn_second_model = KNeighborsClassifier(n_neighbors=200)
knn_second_model.fit(x_training_second_method, y_training)
```

Out[47]:

```
KNeighborsClassifier(n_neighbors=200)
```

In [48]:

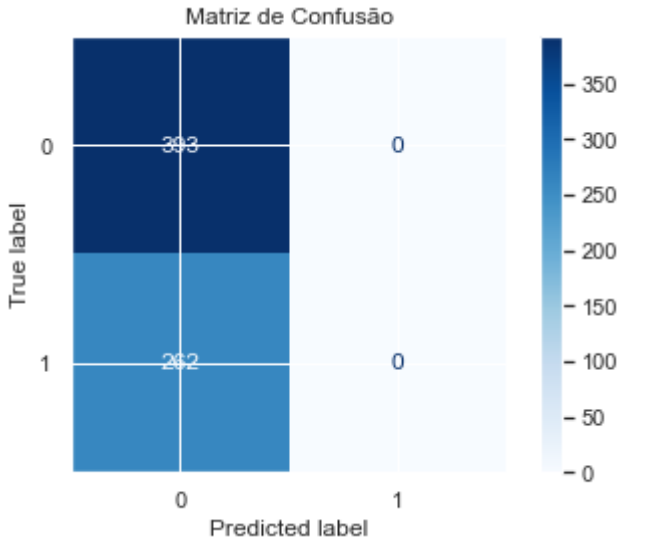
```
knn_second_model_predictions = knn_second_model.predict(x_test)
```

- ***Avaliação do modelo***

- Matriz de confusão

In [49]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(knn_second_model, x_test, y_test, cmap=plt.cm.Blues, normalize
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [50]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [51]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
vet_acuracy.append(acuracy)
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [52]:

```
sensitivity = true_positives / (true_positives + false_negatives)
vet_sensitivity.append(sensitivity)
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [53]:

```
specificity = true_negatives / (true_negatives + false_positives)
vet_specificity.append(specificity)
```

- **Precisão**

In [54]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
vet_precision.append(precision)
```

- ***Recall***

In [55]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
vet_recall.append(recall)
```

- ***F1-Score***

In [56]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0

vet_f1_score.append(f1_score)
```

- ***Cohen's kappa***

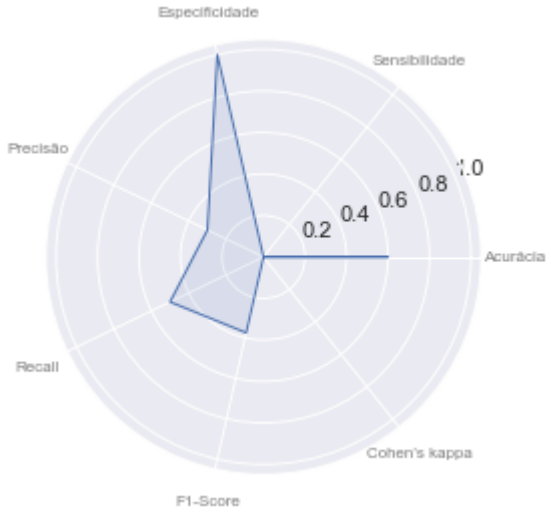
In [57]:

```
cohen_kappa = cohen_kappa_score(y_test, knn_second_model_predictions)
vet_cohen_kappa.append(cohen_kappa)
```

- ***Radar Plot***

In [58]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




## SVM - Support Vector Machine

In [60]:

```
svm_second_model = svm.SVC(kernel='linear', C=0.01)
svm_second_model.fit(x_training_second_method, y_training)
```

Out[60]:

```
SVC(C=0.01, kernel='linear')
```

In [61]:

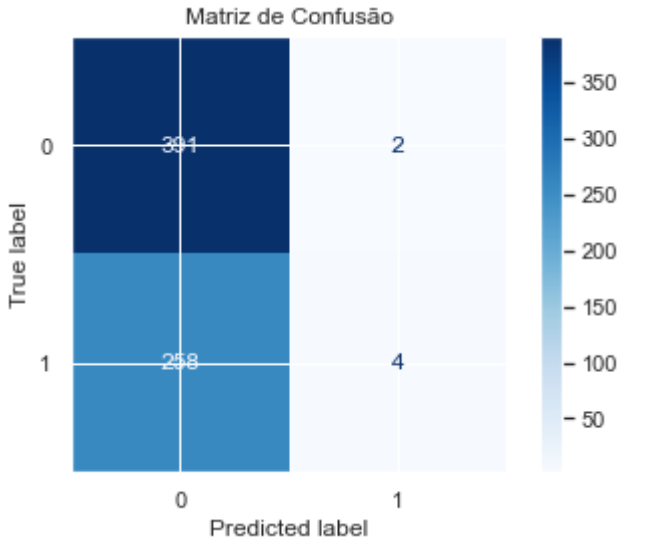
```
svm_second_model_predictions = svm_second_model.predict(x_test)
```

- ***Avaliação do modelo***

- Matriz de confusão

In [62]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(svm_second_model, x_test, y_test, cmap=plt.cm.Blues, normalize
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [63]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [64]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
vet_acuracy.append(acuracy)
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [65]:

```
sensitivity = true_positives / (true_positives + false_negatives)
vet_sensitivity.append(sensitivity)
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [66]:

```
specificity = true_negatives / (true_negatives + false_positives)
vet_specificity.append(specificity)
```

- **Precisão**

In [67]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
vet_precision.append(precision)
```

- ***Recall***

In [68]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
vet_recall.append(recall)
```

- ***F1-Score***

In [69]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0
vet_f1_score.append(f1_score)
```

- ***Cohen's kappa***

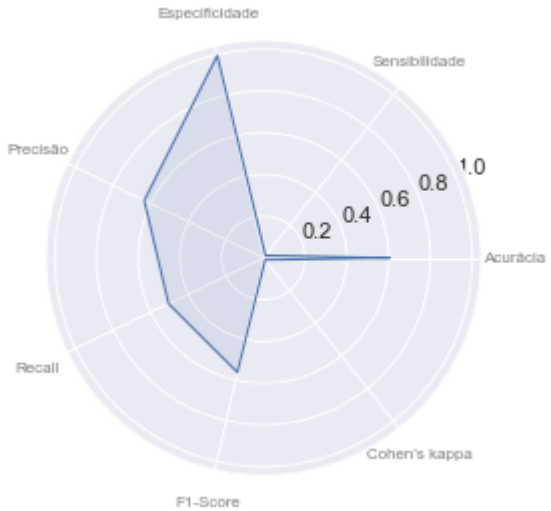
In [70]:

```
cohen_kappa = cohen_kappa_score(y_test, svm_second_model_predictions)
vet_cohen_kappa.append(cohen_kappa)
```

- ***Radar Plot***

In [71]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




## Gradient Tree Boosting

In [73]:

```
g_boost_second_model = GradientBoostingClassifier(n_estimators=200, learning_rate=1.0, max_
g_boost_second_model.fit(x_training_second_method, y_training)
```

Out[73]:

```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, n_estimators=200,
                           random_state=0)
```

In [74]:

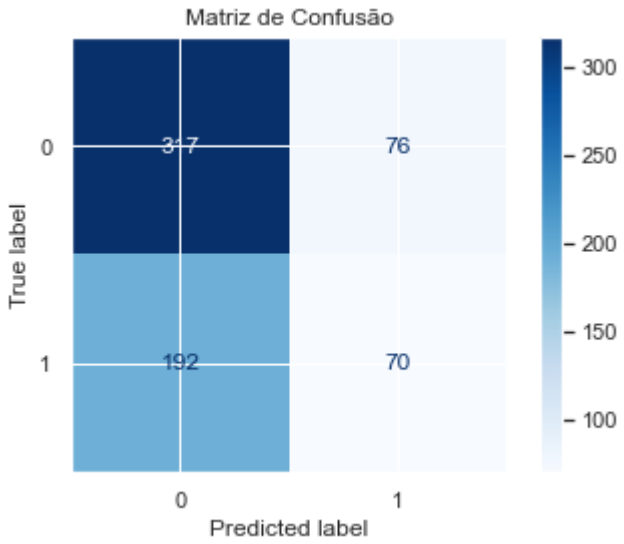
```
g_boost_second_model_predictions = g_boost_second_model.predict(x_test)
```

- ***Avaliação do modelo***

- Matriz de confusão

In [75]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(g_boost_second_model, x_test, y_test, cmap=plt.cm.Blues, norma
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [76]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [77]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
vet_acuracy.append(acuracy)
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [78]:

```
sensitivity = true_positives / (true_positives + false_negatives)
vet_sensitivity.append(sensitivity)
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [79]:

```
specificity = true_negatives / (true_negatives + false_positives)
vet_specificity.append(specificity)
```

- **Precisão**

In [80]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
vet_precision.append(precision)
```

- ***Recall***

In [81]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
vet_recall.append(recall)
```

- ***F1-Score***

In [82]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0

vet_f1_score.append(f1_score)
```

- ***Cohen's kappa***

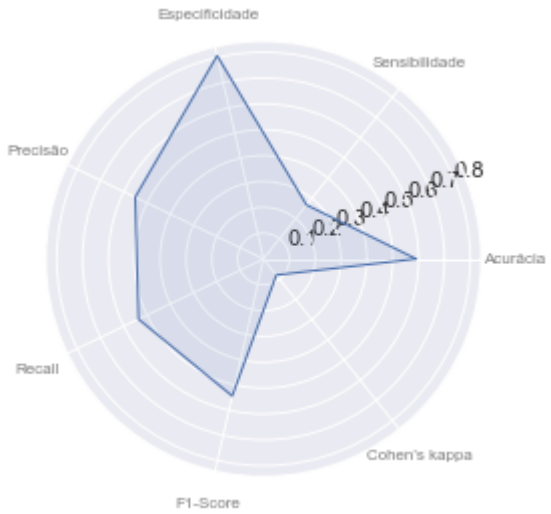
In [83]:

```
cohen_kappa = cohen_kappa_score(y_test, g_boost_second_model_predictions)
vet_cohen_kappa.append(cohen_kappa)
```

- ***Radar Plot***

In [84]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




# 4 Conclusão

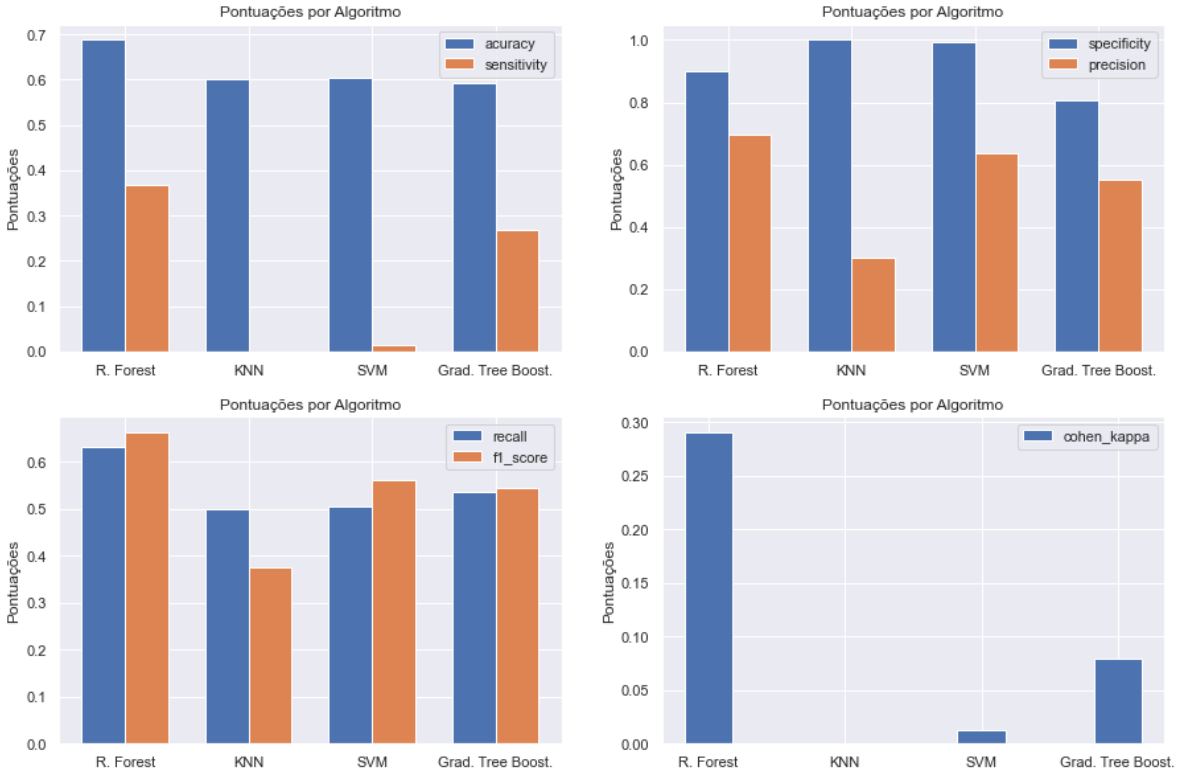
Com a realização dos vários experimentos, é possível avaliar o desempenho dos classificadores sobre a base de dados selecionada. Para isso, começaremos avaliando e comparando o resultado de cada classificador considerando as métricas citadas no início deste trabalho.

In [86]:

```
labels = ['R. Forest', 'KNN', 'SVM', 'Grad. Tree Boost.']
x = np.arange(len(labels))
width = 0.35
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
rects1 = ax1.bar(x - width/2, vet_acuracy, width, label='acuracy')
rects2 = ax1.bar(x + width/2, vet_sensitivity, width, label='sensitivity')
rects3 = ax2.bar(x - width/2, vet_specificity, width, label='specificity')
rects4 = ax2.bar(x + width/2, vet_precision, width, label='precision')
rects5 = ax3.bar(x - width/2, vet_recall, width, label='recall')
rects6 = ax3.bar(x + width/2, vet_f1_score, width, label='f1_score')
rects7 = ax4.bar(x, vet_cohen_kappa, width, label='cohen_kappa')
ax1.set_ylabel('Pontuações')
ax1.set_title('Pontuações por Algoritmo')
ax1.set_xticks(x)
ax1.set_xticklabels(labels)
ax1.legend()
ax2.set_ylabel('Pontuações')
ax2.set_title('Pontuações por Algoritmo')
ax2.set_xticks(x)
ax2.set_xticklabels(labels)
ax2.legend()
ax3.set_ylabel('Pontuações')
ax3.set_title('Pontuações por Algoritmo')
ax3.set_xticks(x)
ax3.set_xticklabels(labels)
ax3.legend()
ax4.set_ylabel('Pontuações')
ax4.set_title('Pontuações por Algoritmo')
ax4.set_xticks(x)
ax4.set_xticklabels(labels)
ax4.legend()
ax1.bar_label(rects1, padding=2)
ax1.bar_label(rects2, padding=2)
ax2.bar_label(rects3, padding=2)
ax2.bar_label(rects4, padding=2)
ax3.bar_label(rects5, padding=2)
ax3.bar_label(rects6, padding=2)
ax4.bar_label(rects7, padding=2)
fig.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-86-c96a56952c92> in <module>
     30 ax4.set_xticklabels(labels)
     31 ax4.legend()
---> 32 ax1.bar_label(rects1, padding=2)
     33 ax1.bar_label(rects2, padding=2)
     34 ax2.bar_label(rects3, padding=2)

AttributeError: 'AxesSubplot' object has no attribute 'bar_label'
```




Notem que as sete métricas foram divididas entre os quatro gráficos acima e para cada uma delas vamos realizar uma avaliação geral.

**Acurácia:** Por não ser adequada para conjunto de dados desbalanceados, não daremos muito crédito a este resultado. Entretanto, com base nele podemos afirmar que, o Random Forest, dentre os classificadores, foi o que realizou mais classificações corretas, podendo ser de uma ou de ambas as classes (potável ou não potável).

**Sensibilidade:** Por ser a proporção de casos positivos que foram identificados corretamente podemos inferir o seguinte:

- Nenhum classificador foi capaz de classificar corretamente metade ou mais que a metade das amostras positivas, ou seja, água potável;
- Novamente o Random Forest foi o que se destacou um pouco mais dos demais nesse quesito.
- Notem que o KNN não foi capaz de classificar corretamente nenhuma amostra de água potável e o SVM apresentou um resultado ínfimo.

**Especificidade:** Por ser a proporção de casos negativos que foram identificados corretamente podemos inferir o seguinte:

- Notem que com relação a essa métrica, o KNN e o SVM obtêm os melhores resultados, entretanto, com base na métrica anterior isso é preocupante, pois ambos estão tendenciosos, ou seja, independente da amostra, na maioria dos casos consideram como não potável. Um dos fatores que podem causar isso é o desbalanceamento entre classes, contudo, pode estar relacionado com os dados em si, onde os mesmos podem não apresentar uma divisão clara entre uma classe e outra, o que dificulta o trabalho dessa categoria de classificador.

- Todos os algoritmos foram capazes de classificar corretamente mais da metade das amostras de água não potável.

**Precisão:** Notem que o algoritmo Random Forest volta a obter o melhor resultado entre os classificadores, isso significa que ele obteve uma maior taxa de acerto em suas classificações.

**Recall:** O recall representa a proporção de acertos dos verdadeiros positivos e verdadeiros negativos com base na quantidade total de cada uma dessas categorias. Notem que o Random Forest apresenta o melhor resultado.

**F1-Score:** F1-Score é uma média Harmônica entre precisão e recall, adequada para análise de datasets com classes desproporcionais, como é o caso do nosso dataset. Quanto maior o f1 score melhor, notem que o random forest volta a apresentar o melhor desempenho.

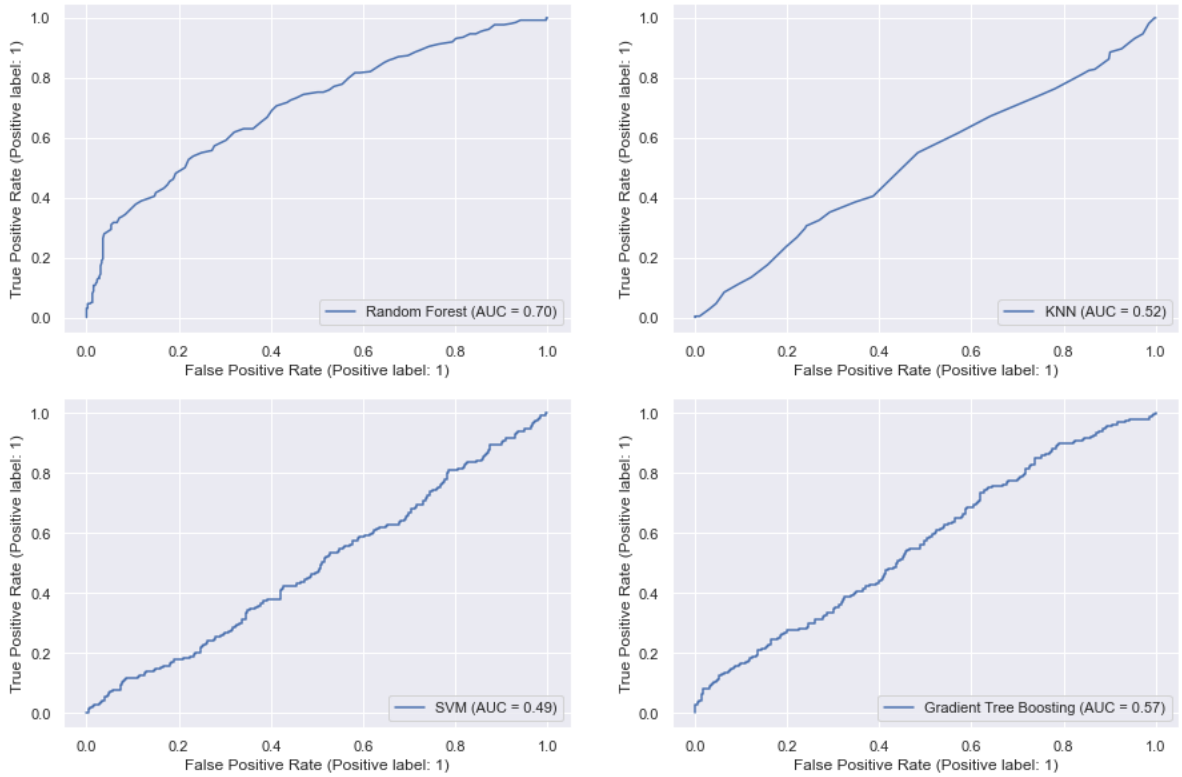
**Cohen's kappa:** Kappa é uma métrica que avalia a concordância observada, em relação à concordância esperada, e realiza uma análise do classificador de tal forma a verificar o seu desempenho em comparação com um classificador que simplesmente faz adivinhações aleatórias. Notem que em comparação com os demais classificadores, o Random Forest obteve o melhor resultado, contudo, não é um um resultado relativamente alto.

## Curva ROC

**Média**

In [87]:

```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
metrics.plot_roc_curve(random_forest_second_model, x_test, y_test, name="Random Forest", ax
metrics.plot_roc_curve(knn_second_model, x_test, y_test, name="KNN", ax=ax2)
metrics.plot_roc_curve(svm_second_model, x_test, y_test, name="SVM", ax=ax3)
metrics.plot_roc_curve(g_boost_second_model, x_test, y_test, name="Gradient Tree Boosting",
plt.show()
```




Com base na curva ROC acima e nas análises anteriores, é possível concluir que o melhor desempenho para a classificação de amostras de água, foi obtido com o algoritmo Random Forest, seguido do Gradient Tree Boosting, os quais podem ser aprimorados para utilização na tarefa de classificação da água. Em contrapartida, devido às características dos dados e o péssimo desempenho, consideramos os algoritmos KNN e SVM inapropriados para essa tarefa. Contudo, devemos levar em consideração que vários fatores podem ter induzido tal resultado. As classes desbalanceadas são um deles, além disso, com base nos resultados, não somos capazes de inferir que a quantidade de dados disponível é suficientemente grande a ponto de proporcionar um treinamento adequado para os modelos. Por fim, a capacidade de cada modelo pode ser estendida a desempenhos bem superiores aos deste trabalho, contudo, é preciso realizar um ajuste fino dos modelos, de tal forma a adequá-los ao problema de classificação da água.

# Referências


[1] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. **LightGBM: uma árvore de decisão de aumento de gradiente altamente eficiente.** NIPS 2017.

[2] Tianqi Chen, Carlos Guestrin. **XGBoost: A Scalable Tree Boosting System.** Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[3] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Annals of Statistics, 28(2):337–407, 2000.

[4] Flai. **Scatter plot: Um Guia Completo para Gráficos de Dispersão.** Link (https://www.flai.com.br/juscudilio/scatter-plot-um-guia-completo-para-graficos-de-dispersao/). Acessado em 19/07/2021.

[5] Wikipédia. **Gráfico de dispersão.** Link (https://pt.wikipedia.org/wiki/Gr%C3%A1fico_de_dispers%C3%A3o). Acessado em 19/07/2021.

[6] Significados. **Histograma.** Link (https://www.significados.com.br/histograma/). Acessado em 19/07/2021.

[7] Oper. **BOXPLOT: COMO INTERPRETAR?.** Link (https://operdata.com.br/blog/como-interpretar-um-boxplot/). Acessado em 19/07/2021.

[8] EDTI. **BoxPlot: Saiba tudo sobre o Diagrama de caixa e como interpretar esse gráfico.** Link (https://www.escolaedti.com.br/o-que-e-um-box-plot). Acessado em 19/07/2021.

[9] Towards Data Science. **Violin plots explained.** Link (https://towardsdatascience.com/violin-plots-explained-fb1d115e023d). Acessado em 19/07/2021.

[10] Wikipédia. **Violin plot.** Link (https://en.wikipedia.org/wiki/Violin_plot). Acessado em 19/07/2021.

[11] NEILPATEL. **Heatmap: Entenda o Que é, Quando Usar e Que Ferramentas Usar.** Link (https://neilpatel.com/br/blog/heatmap-o-que-e/). Acessado em 19/07/2021.

[12] ZOOX. **HEATMAP: POR ONDE ANDAM SEUS CLIENTES?.** Link (https://zooxsmart.com/pt/heatmap-o-que-e-como-funciona-tipos-e-beneficios/). Acessado em 19/07/2021.

[13] Wikipédia. **Gráfico de radar.** Link (https://pt.wikipedia.org/wiki/Gr%C3%A1fico_de_radar). Acessado em 19/07/2021.

[14] Computação Inteligente. **K vizinhos mais próximos - KNN.** Link (http://computacaointeligente.com.br/algoritmos/k-vizinhos-mais-proximos/). Acessado em 19/07/2021.

[15] Wikipédia. **Random forest.** Link (https://en.wikipedia.org/wiki/Random_forest). Acessado em 19/07/2021.

[16] Didática Tech. **O que é e como funciona o algoritmo RandomForest.** Link (https://didatica.tech/o-que-e-e-como-funciona-o-algoritmo-randomforest/). Acessado em 19/07/2021.

[17] DATARISK. **O que é Matriz de Confusão?.** Link (https://ajuda.datarisk.io/knowledge/o-que-%C3%A9-matriz-de-confus%C3%A3o). Acessado em 19/07/2021.

[18] Mario Filho. **As Métricas Mais Populares para Avaliar Modelos de Machine Learning.** Link (https://www.mariofilho.com/as-metricas-mais-populares-para-avaliar-modelos-de-machine-learning/). Acessado em 19/07/2021.

[19] MEDIUM. **Métricas Comuns em Machine Learning: como analisar a qualidade de chat bots inteligentes.** Link (https://medium.com/as-m%C3%A1quinas-que-pensam/m%C3%A9tricas-comuns-em-machine-learning-como-analisar-a-qualidade-de-chat-bots-inteligentes-m%C3%A9tricas-1ba580d7cc96).

Acessado em 19/07/2021.

[20] Diego Mariano. **Métricas de avaliação em machine learning.** Link (https://diegomariano.com/metricas-de-avaliacao-em-machine-learning/#Precisao). Acessado em 19/07/2021.

[21] The Data Scientist. **Performance Measures: Cohen's Kappa statistic.** Link (https://thedatascientist.com/performance-measures-cohens-kappa-statistic/). Acessado em 19/07/2021.

[22] Wikipédia. **Característica de Operação do Receptor.** Link (https://pt.wikipedia.org/wiki/Caracter%C3%ADstica_de_Opera%C3%A7%C3%A3o_do_Receptor). Acessado em 19/07/2021.

[23] Wikipédia. **Correlação.** Link (https://pt.wikipedia.org/wiki/Correla%C3%A7%C3%A3o). Acessado em 19/07/2021.

# Apêndice 1

## Implementações adicionais

#### Random Forest

***Eliminação de valores ausentes***

In [88]:

```
random_forest_first_model = RandomForestClassifier(n_estimators=200)
random_forest_first_model.fit(x_training_first_method, y_training_first_method)
```

Out[88]:

```
RandomForestClassifier(n_estimators=200)
```

In [89]:

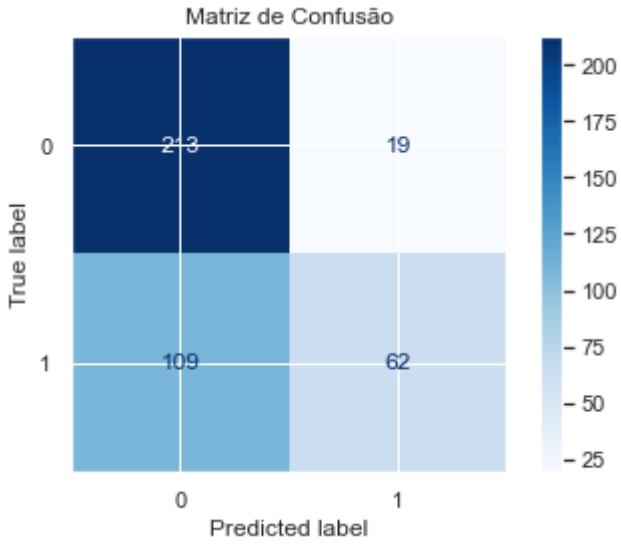
```
random_forest_first_model_predictions = random_forest_first_model.predict(x_test_first_meth
```

***Avaliação do modelo***

- Matriz de confusão

In [90]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(random_forest_first_model, x_test_first_method, y_test_first_m
                             normalize=None)
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [91]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [92]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
acuracy
```

Out[92]:

```
0.6823821339950372
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [93]:

```
sensitivity = true_positives / (true_positives + false_negatives)
sensitivity
```

Out[93]:

```
0.36257309941520466
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [94]:

```
specificity = true_negatives / (true_negatives + false_positives)
specificity
```

Out[94]:

```
0.9181034482758621
```

- ***Precisão***

In [95]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_negatives = 0

precision = (precision_positives + precision_negatives) / 2
precision
```

Out[95]:

```
0.7134613909976228
```

- ***Recall***

In [96]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = (true_positives / (true_positives + false_negatives))
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = (true_negatives / (true_negatives + false_positives))
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
recall
```

Out[96]:

```
0.6403382738455334
```

- ***F1-Score***

In [97]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0
f1_score
```

Out[97]:

```
0.6749250238879028
```

- ***Cohen's kappa***

In [98]:

```
cohen_kappa = cohen_kappa_score(y_test_first_method, random_forest_first_model_predictions)
cohen_kappa
```

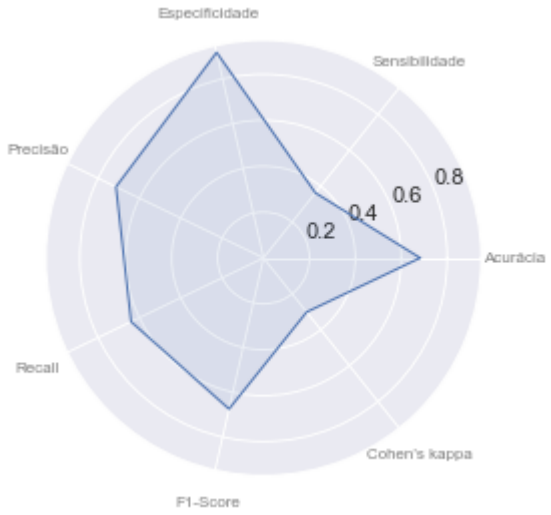
Out[98]:

```
0.3015408779483847
```

- ***Radar Plot***

In [99]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




***Mediana***

In [100]:

```
random_forest_third_model = RandomForestClassifier(n_estimators=200)
random_forest_third_model.fit(x_training_third_method, y_training)
```

Out[100]:

```
RandomForestClassifier(n_estimators=200)
```

In [101]:

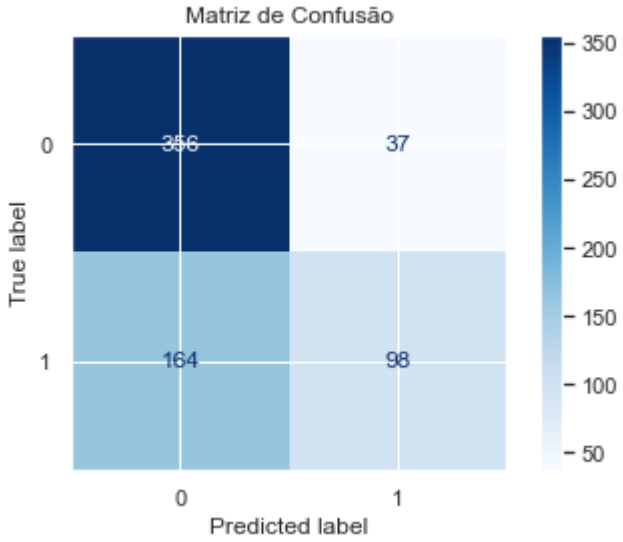
```
random_forest_third_model_predictions = random_forest_third_model.predict(x_test)
```

***Avaliação do modelo***

- Matriz de confusão

In [102]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(random_forest_third_model, x_test, y_test, cmap=plt.cm.Blues,
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [103]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [104]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
acuracy
```

Out[104]:

```
0.6931297709923664
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [105]:

```
sensitivity = true_positives / (true_positives + false_negatives)
sensitivity
```

Out[105]:

```
0.37404580152671757
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [106]:

```
specificity = true_negatives / (true_negatives + false_positives)
specificity
```

Out[106]:

```
0.905852417302799
```

- **Precisão**

In [107]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
precision
```

Out[107]:

```
0.7052706552706554
```

- ***Recall***

In [108]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
recall
```

Out[108]:

```
0.6399491094147582
```

- ***F1-Score***

In [109]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0
f1_score
```

Out[109]:

```
0.6710239316805852
```

- ***Cohen's kappa***

In [110]:

```
cohen_kappa = cohen_kappa_score(y_test, random_forest_third_model_predictions)
cohen_kappa
```

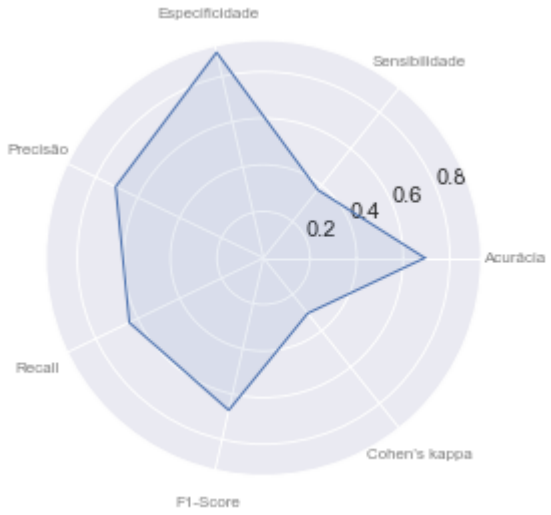
Out[110]:

```
0.3044982698961938
```

- ***Radar Plot***

In [111]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




**KNN (K-Nearest Neighbors)**

***Eliminação de valores ausentes***

In [112]:

```
knn_first_model = KNeighborsClassifier(n_neighbors=200)
knn_first_model.fit(x_training_first_method, y_training_first_method)
```

Out[112]:

```
KNeighborsClassifier(n_neighbors=200)
```

In [113]:

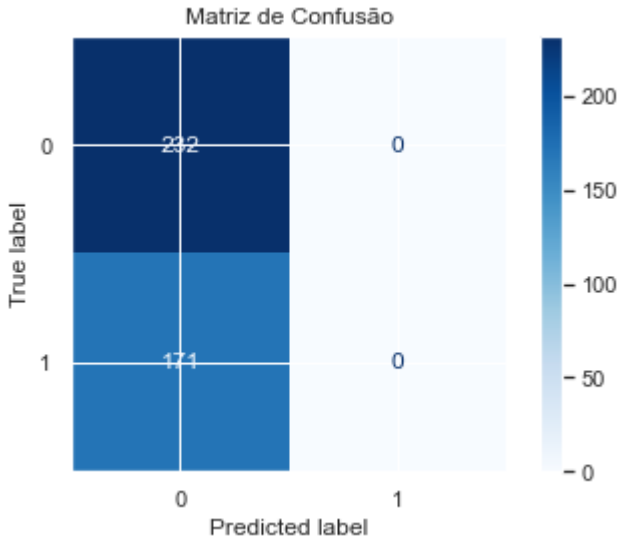
```
knn_first_model_predictions = knn_first_model.predict(x_test_first_method)
```

***Avaliação do modelo***

- Matriz de confusão

In [114]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(knn_first_model, x_test_first_method, y_test_first_method, cma
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [115]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [116]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
acuracy
```

Out[116]:

```
0.575682382133995
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [117]:

```
sensitivity = true_positives / (true_positives + false_negatives)
sensitivity
```

Out[117]:

```
0.0
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [118]:

```
specificity = true_negatives / (true_negatives + false_positives)
specificity
```

Out[118]:

```
1.0
```

- **Precisão**

In [119]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
precision
```

Out[119]:

```
0.2878411910669975
```

- ***Recall***

In [120]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
recall
```

Out[120]:

```
0.5
```

- ***F1-Score***

In [121]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0
f1_score
```

Out[121]:

```
0.3653543307086614
```

- ***Cohen's kappa***

In [122]:

```
cohen_kappa = cohen_kappa_score(y_test_first_method, knn_first_model_predictions)
cohen_kappa
```

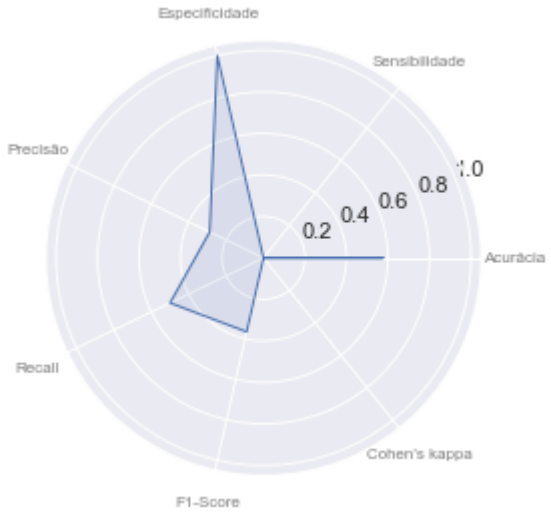
Out[122]:

```
0.0
```

- ***Radar Plot***

In [123]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




***Mediana***

In [124]:

```
knn_third_model = KNeighborsClassifier(n_neighbors=200)
knn_third_model.fit(x_training_third_method, y_training)
```

Out[124]:

```
KNeighborsClassifier(n_neighbors=200)
```

In [125]:

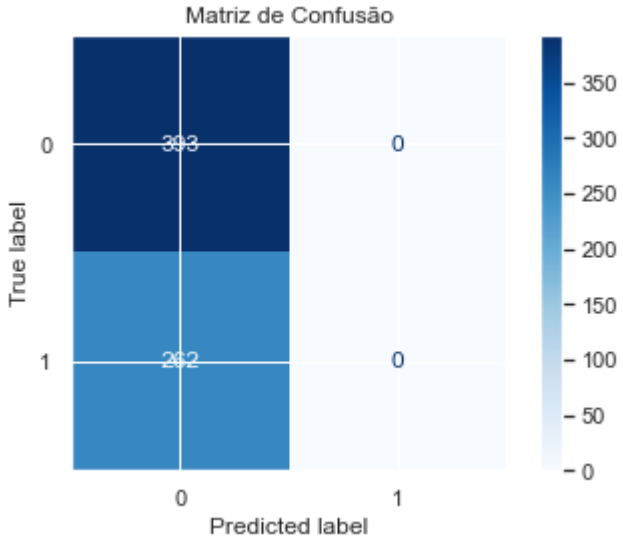
```
knn_third_model_predictions = knn_third_model.predict(x_test)
```

***Avaliação do modelo***

- Matriz de confusão

In [126]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(knn_third_model, x_test, y_test, cmap=plt.cm.Blues, normalize=
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [127]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [128]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
acuracy
```

Out[128]:

```
0.6
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [129]:

```
sensitivity = true_positives / (true_positives + false_negatives)
sensitivity
```

Out[129]:

```
0.0
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [130]:

```
specificity = true_negatives / (true_negatives + false_positives)
specificity
```

Out[130]:

```
1.0
```

- **Precisão**

In [131]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
precision
```

Out[131]:

```
0.3
```

- ***Recall***

In [132]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
recall
```

Out[132]:

```
0.5
```

- ***F1-Score***

In [133]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0
f1_score
```

Out[133]:

```
0.37499999999999994
```

- ***Cohen's kappa***

In [134]:

```
cohen_kappa = cohen_kappa_score(y_test, knn_third_model_predictions)
cohen_kappa
```

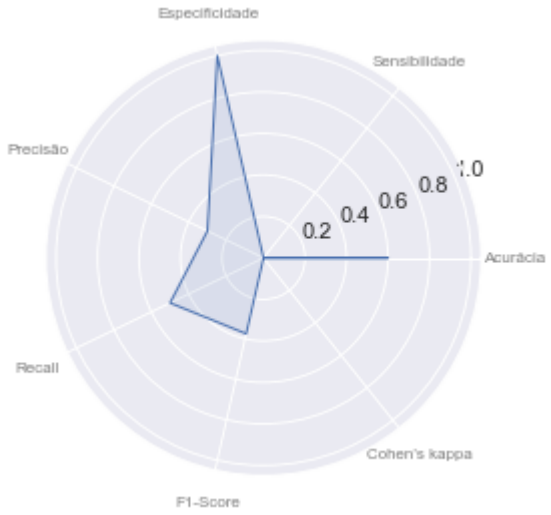
Out[134]:

```
0.0
```

- ***Radar Plot***

In [135]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




**SVM - Support Vector Machine**

***Eliminação de valores ausentes***

In [136]:

```
svm_first_model = svm.SVC(kernel='linear', C=0.01)
svm_first_model.fit(x_training_first_method, y_training_first_method)
```

Out[136]:

```
SVC(C=0.01, kernel='linear')
```

In [137]:

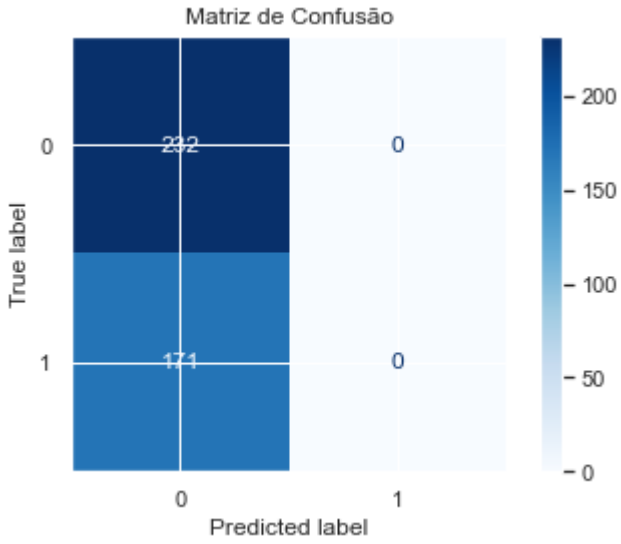
```
svm_first_model_predictions = svm_first_model.predict(x_test_first_method)
```

***Avaliação do modelo***

- Matriz de confusão

In [138]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(svm_first_model, x_test_first_method, y_test_first_method, cma
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [139]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [140]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
acuracy
```

Out[140]:

```
0.575682382133995
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [141]:

```
sensitivity = true_positives / (true_positives + false_negatives)
sensitivity
```

Out[141]:

```
0.0
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [142]:

```
specificity = true_negatives / (true_negatives + false_positives)
specificity
```

Out[142]:

```
1.0
```

- **Precisão**

In [143]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
precision
```

Out[143]:

```
0.2878411910669975
```

- ***Recall***

In [144]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
recall
```

Out[144]:

```
0.5
```

- ***F1-Score***

In [145]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0
f1_score
```

Out[145]:

```
0.3653543307086614
```

- ***Cohen's kappa***

In [146]:

```
cohen_kappa = cohen_kappa_score(y_test_first_method, svm_first_model_predictions)
cohen_kappa
```

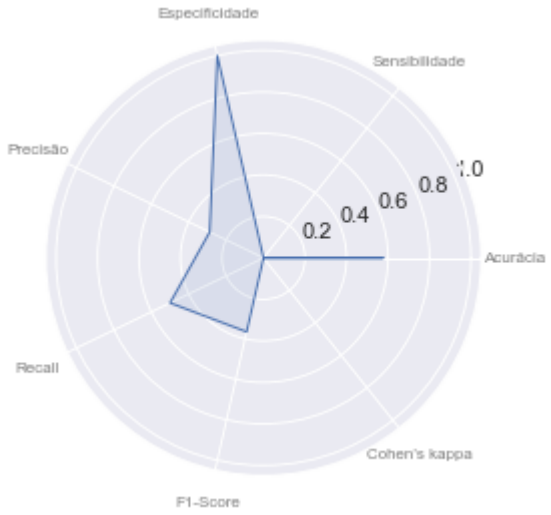
Out[146]:

```
0.0
```

- ***Radar Plot***

In [147]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```




***Mediana***

In [148]:

```
svm_third_model = svm.SVC(kernel='linear', C=0.01)
svm_third_model.fit(x_training_third_method, y_training)
```

Out[148]:

```
SVC(C=0.01, kernel='linear')
```

In [149]:

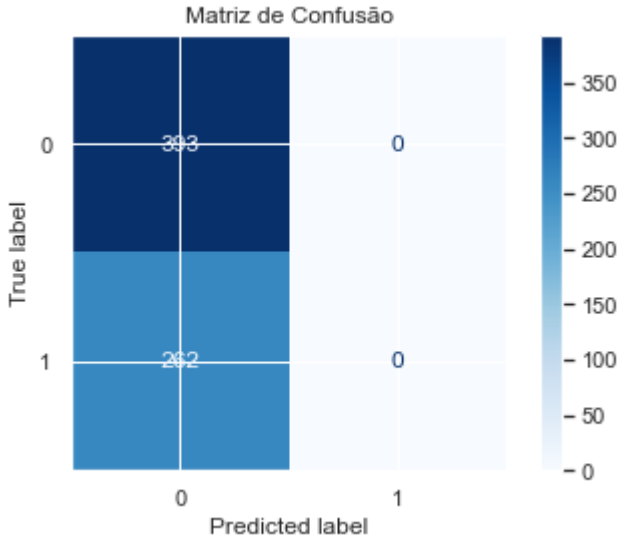
```
svm_third_model_predictions = svm_third_model.predict(x_test)
```

***Avaliação do modelo***

- Matriz de confusão

In [150]:

```
np.set_printoptions(precision=2)
disp = plot_confusion_matrix(svm_third_model, x_test, y_test, cmap=plt.cm.Blues, normalize=
disp.ax_.set_title('Matriz de Confusão')
plt.show()
```




In [151]:

```
matrix = disp.confusion_matrix
true_positives  = matrix[1, 1]
true_negatives  = matrix[0, 0]
false_positives = matrix[0, 1]
false_negatives = matrix[1, 0]
```

- ***Acurácia:*** (quão frequênte o classificador está correto)

In [152]:

```
acuracy = (true_positives + true_negatives) / (true_positives + true_negatives + false_posi
acuracy
```

Out[152]:

```
0.6
```

- ***Sensibilidade:*** (quando o valor real é positivo, quão frequênte as predições são corretas?)

In [153]:

```
sensitivity = true_positives / (true_positives + false_negatives)
sensitivity
```

Out[153]:

```
0.0
```

- ***Especificidade:*** (quando o valor real é negativo, quão frequênte as predições são corretas?)

In [154]:

```
specificity = true_negatives / (true_negatives + false_positives)
specificity
```

Out[154]:

```
1.0
```

- **Precisão**

In [155]:

```
if (true_positives + false_positives) > 0:
    precision_positives = (true_positives / (true_positives + false_positives))
else:
    precision_positives = 0

if (true_negatives + false_negatives) > 0:
    precision_negatives = (true_negatives / (true_negatives + false_negatives))
else:
    precision_positives = 0

precision = (precision_positives + precision_negatives) / 2
precision
```

Out[155]:

```
0.3
```

- ***Recall***

In [156]:

```
if (true_positives + false_negatives) > 0:
    recall_positives = true_positives / (true_positives + false_negatives)
else:
    recall_positives = 0

if (true_positives + false_negatives) > 0:
    recall_negatives = true_negatives / (true_negatives + false_positives)
else:
    recall_negatives = 0

recall = (recall_positives + recall_negatives) / 2
recall
```

Out[156]:

```
0.5
```

- ***F1-Score***

In [157]:

```
if (precision + sensitivity) > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0
f1_score
```

Out[157]:

```
0.37499999999999994
```

- ***Cohen's kappa***

In [158]:

```
cohen_kappa = cohen_kappa_score(y_test, svm_third_model_predictions)
cohen_kappa
```

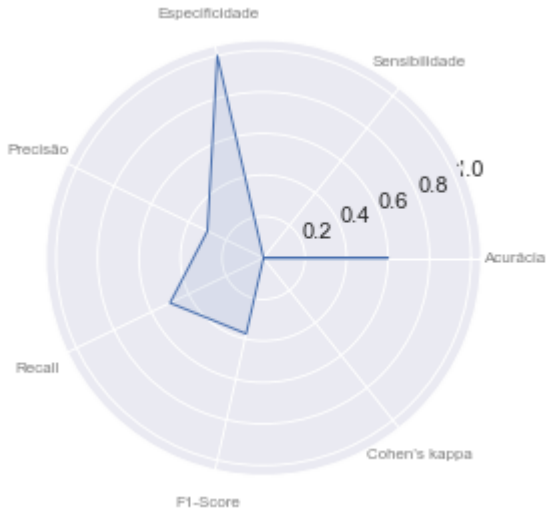
Out[158]:

```
0.0
```

- ***Radar Plot***

In [159]:

```
df = pd.DataFrame({
'group': ['métricas'],
'Acurácia': [acuracy],
'Sensibilidade': [sensitivity],
'Especificidade': [specificity],
'Precisão': [precision],
'Recall': [recall],
'F1-Score': [f1_score],
'Cohen’s kappa': [cohen_kappa],
})
categories = list(df)[1:]
N = len(categories)
values = df.loc[0].drop('group').values.flatten().tolist()
values += values[:1]
angles = [n / float(N) * 2 * pi for n in range(N)]
angles += angles[:1]
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], categories, color='grey', size=8)
ax.plot(angles, values, linewidth=1, linestyle='solid')
ax.fill(angles, values, 'b', alpha=0.1)
plt.show()
```

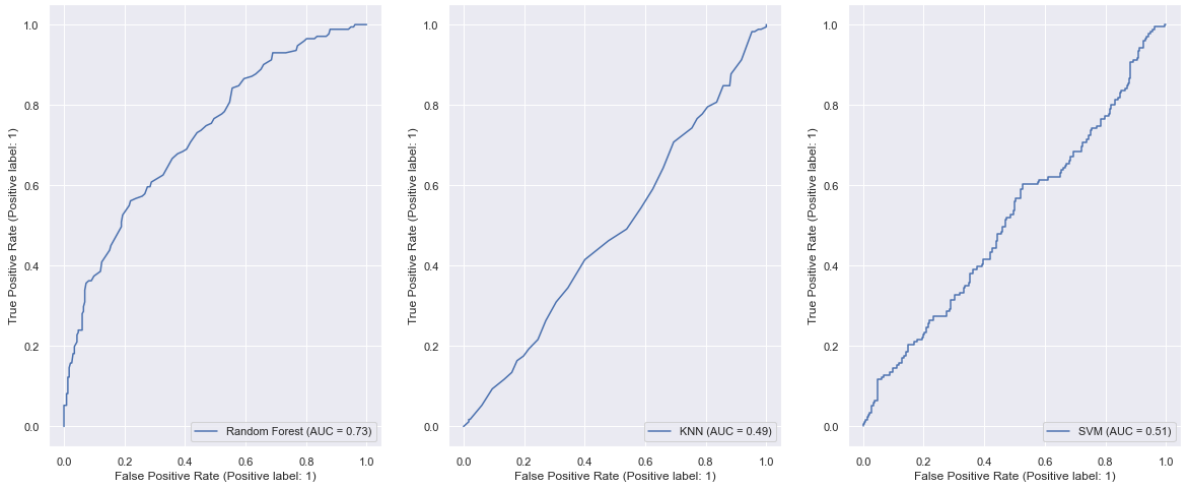



## Curva ROC

### Eliminação de valores ausentes

In [160]:

```
fig, ((ax1, ax2, ax3)) = plt.subplots(1, 3, figsize=(20, 8))
metrics.plot_roc_curve(random_forest_first_model, x_test_first_method, y_test_first_method,
metrics.plot_roc_curve(knn_first_model, x_test_first_method, y_test_first_method, name="KNN
metrics.plot_roc_curve(svm_first_model, x_test_first_method, y_test_first_method, name="SVM
plt.show()
```




**Mediana**

In [161]:

```
fig, ((ax1, ax2, ax3)) = plt.subplots(1, 3, figsize=(20, 8))
metrics.plot_roc_curve(random_forest_third_model, x_test, y_test, name="Random Forest", ax
metrics.plot_roc_curve(knn_third_model, x_test, y_test, name="KNN", ax=ax2)
metrics.plot_roc_curve(svm_third_model, x_test, y_test, name="SVM", ax=ax3)
plt.show()
```