UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

FELIPE FAGUNDES SCALCO

VISUALIZAÇÃO DE DADOS EM PROCESSOS DE MACHINE LEARNING

CAXIAS DO SUL

2021

FELIPE FAGUNDES SCALCO

VISUALIZAÇÃO DE DADOS EM PROCESSOS DE MACHINE LEARNING

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dr. Carine Geltrudes Webber

CAXIAS DO SUL

2021

FELIPE FAGUNDES SCALCO

VISUALIZAÇÃO DE DADOS EM PROCESSOS DE MACHINE LEARNING

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 24/07/2021

BANCA EXAMINADORA

Prof. Dr. Carine Geltrudes Webber
Universidade de Caxias do Sul - UCS

Prof. Dr. Elisa Boff
Universidade de Caxias do Sul - UCS

Prof. Dr. Maria de Fatima Webber do Prado Lima
Universidade de Caxias do Sul - UCS

RESUMO

Com o recente surgimento do *Big Data* a produção de dados sendo gerados diariamente nunca foi tão grande. E para que se possa extrair conhecimento desses dados o *Machine Learning* interpreta um papel fundamental, devido a sua capacidade de aprender com dados históricos. As técnicas de *Machine Learning* compreendem métodos orientados a dados que combinam conceitos fundamentais da ciência da computação com outros das áreas de estatística, probabilidade e otimização e tem como objetivo implementar algoritmos capazes de aprender, com pouca ou nenhuma necessidade de assistência ou intervenção humana. Tendo em vista a complexidade dos conjuntos de dados soma-se ao processo de *Machine Learning* componentes visuais. Os componentes visuais tem como objetivo representar visualmente as informações. Auxiliando desta maneira a interpretação dos dados, aproveitando o sistema visual dos seres humanos para reconhecer padrões. Além de oferecer diferentes maneiras de mostrar os resultados de uma forma mais amigável, ajudando na tomada de decisão. Para demonstrar a relevância da visualização de dados durante o processo de *Machine Learning* este trabalho desenvolve e avalia um processo de *Machine Learning* apoiado em ferramentas de visualização de dados. Utilizou-se como método o estudo de caso, por considerar-se que cada conjunto de dados deve ser tratado de maneira única, dadas as suas características e distribuições. As ferramentas de visualizações selecionadas foram as seguintes: *Yellowbrick*, *Seaborn*, *Ploty*, *MatPlotLib*. Neste trabalho, selecionou-se dois conjuntos de dados distintos. O primeiro conjunto de dados, contempla dados públicos que contém o número de casos de ocorrências de contaminados, recuperados e mortos pelo novo coronavírus em todos os continentes. Por meio deste conjunto explorou-se recursos de visualização de dados numéricos e categóricos, com mapas e gráficos mistos. Já o segundo conjunto de dados, amplamente utilizado em estudos na área, foi o conjunto de dados Iris. Este conjunto de dados é normalmente usado para avaliar algoritmos de classificação pois apresenta estrutura bem definida. Além disso, ele constitui um elemento importante para os estudos na área, pois seus dados apresentam características numéricas e categóricas (classes). Isso possibilita comparações, análises e visualizações que, uma vez compreendidas em cenário ilustrativo, podem ser replicadas em outros contextos complexos. Como resultados observados, destaca-se a importância das ferramentas de visualização no processo de *Machine Learning*, contribuindo para o entendimento e validação das suas etapas. Em especial, em conjuntos de dados com o Coronavirus recursos de análise do erro (médio, quadrático, etc) além de medidas estatísticas são as mais usadas. Já em conjuntos de dados com o *Iris* recursos de análise do precisão dos classificadores são mais relevantes. Para concluir, considera-se que os resultados foram satisfatórios e que as ferramentas de visualização, além de necessárias na tomada da decisão, auxiliam o ser humano no processo de exploração dos dados.

Palavras-chaves: Inteligência Artificial. Aprendizado de Máquina. Visualização de Dados. Ciência de Dados.

ABSTRACT

With the recent emergence of Big Data the amount of data being generated on a daily basis has never been greater. In order to extract knowledge from this data Machine Learning plays a key role, due to its ability to learn from historical data. Machine Learning techniques comprise data-driven methods that combine fundamental concepts from computer science with others fields like statistics, probability, optimization and aim to implement algorithms capable of learning, with little or no need for human assistance or intervention. Given the complexity of data sets, visual components are added to the Machine Learning process. The visual components aim to visually represent the information. Helping in the interpretation of data, taking advantage of the visual system of human beings to recognize patterns. Besides offering different ways to show the results in a more friendly way, helping in the decision making process. To demonstrate the relevance of data visualization during the Machine Learning process, this work develops and evaluates a Machine Learning process supported by data visualization tools. The case study method was used, since it is considered that each data set must be treated in a unique way, given its characteristics and distributions. The visualization tools selected were Yellowbrick, Seaborn, Ploty, and MatPlotLib. In this work, two distinct datasets were selected. The first data set, includes public data that contains the number of cases of those contaminated, recovered and killed by the new coronavirus in all continents. Through this set, numerical and categorical data visualization resources were explored, with maps and mixed graphics. The second dataset, widely used in studies in the area, was the Iris dataset. This dataset is commonly used to evaluate classification algorithms because it has a well-defined structure. Moreover, it constitutes an important element for studies in the area because its data present numerical and categorical characteristics (classes). This enables comparisons, analyses, and visualizations that, once understood in an illustrative scenario, can be replicated in other complex contexts. As observed results, we highlight the importance of visualization tools in the Machine Learning process, contributing to the understanding and validation of its steps. In particular, in data sets with Coronavirus, error analysis resources (mean, square, etc.) and statistical measures are the most used. On the other hand, on data sets with Iris, resources for analyzing the accuracy of the classifiers are more relevant. To conclude, it is considered that the results were satisfactory and that visualization tools, besides being necessary for decision making, help the human being in the data exploration process.

Keywords: Artificial Intelligence. Machine Learning. Data Visualization. Data Science.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

IA	*Inteligência Artificial*
VP	*Verdadeiro Positivo*
VN	*Verdadeiro Negativo*
FP	*Falso Positivo*
FN	*Falso Negativo*
ROC	*Receiver Operating Characteristic*
AUC	*Area Under the Curve*
EUA	*Estados Unidos da América*
KNN	*K-Nearest Neighbors*

SUMÁRIO

1 INTRODUÇÃO

Com o recente surgimento do *Big Data* a produção de dados nunca foi tão grande e para que se possa extrair informações relevantes desses dados, o *Machine Learning* possui um papel fundamental por conta de sua capacidade de aprender com os dados, prometendo transformar a forma como vivemos, trabalhamos e pensamos, além da possibilidade de otimizar processos, capacitando a descoberta de *insights* e melhorando a tomada de decisões baseada em dados (L'HEUREUX, 2017).

Enquanto a capacidade de armazenar e coletar os dados cresce rapidamente, a habilidade de analisar esse volume de dados não cresce na mesma proporção (KEIM *et al.*, 2006), tendo em vista que os conjuntos de dados que estão surgindo são extremamente volumosos, possuem muitos ruídos e são heterogêneos por natureza. Essas características acabam dificultando a interpretação desses dados, tornando difícil a extração de alguma informação de valor que possa ser utilizada. Dentro deste contexto a visualização de dados assume um papel relevante à medida que pode auxiliar na interpretação dos dados e no processo de *Machine Learning*.

O conceito de se utilizar visualizações para uma melhor compreensão, seja qual for o assunto, existe há séculos, desde a criação de mapas para auxiliar na navegação até tabelas que continham posições de estrelas e outros corpos celestes (CHEN WOLFGANG HäRDLE, 2008). O objetivo da visualização é representar visualmente as informações, auxiliando assim na interpretação dos dados, aproveitando desta maneira o sistema visual dos seres humanos para reconhecer padrões, possibilitando a descoberta de *insights* para então poder tirar conclusões e tomar melhores decisões (JEFFREY; MICHAEL; VADIM, 2010) deixando os cálculos complexos a serem realizados pelos computadores, combinando dessa maneira as forças dos humanos e computadores.

A área de *Machine Learning* se ocupa do desenvolvimento e aplicação de métodos computacionais para tornar as máquinas capazes de aprimorarem-se a partir da experiência (MITCHELL, 1997). Em processos de aprendizado supervisionado e não supervisionado, que são dependentes de dados, os recursos visuais podem contribuir no processo de desenvolvimento do sistema de *Machine Learning*. Isso porque, na maioria das vezes, os dados em si não fazem sentido até que possam ser analisados de uma forma visual como, por exemplo, por meio de gráficos.

A visualização de dados é uma das ferramentas de *Machine Learning* que desempenha um papel fundamental em todas as etapas. Essas ferramentas facilitam o processo de descoberta de padrões, além de oferecerem diferentes maneiras de mostrar os resultados de uma forma mais amigável ajudando na tomada de decisão.

A visualização de dados é um recurso normalmente empregado ao final do processo de

Machine Learning. Contudo existem ferramentas visuais que podem contribuir nas diferentes etapas do processo de *Machine Learning*, como por exemplo, no pré-processamento de dados onde uma análise exploratória pode ser usada para detectar pontos fora da curva, identificar desequilíbrio nas classes ou uma correlação entre os dados. Pode ser utilizada durante a modelagem dos dados com o intuito de identificar *clusters*. Na avaliação da performance do modelo é possível aplicar visualizações para as diversas métricas disponíveis e por fim no auxílio de apresentar os resultados finais provenientes do sistema de *Machine Learning*.

O campo da visualização de dados é motivado por esses problemas complexos que requerem um esforço em conjunto de algoritmos e de um analista. Sendo o analista um especialista no domínio específico, conseguindo desta maneira direcionar o poder dos algoritmos de *Machine Learning* para onde será mais efetivo e ajustando os resultados para transformá-los em decisões baseadas em dados (KEIM *et al.*, 2006). Um grande desafio em que a visualização de dados também pode ser utilizada é na interpretação de resultados de modelos com características caixa-preta (TURKAY; LARAMEE; HOLZINGER, 2017). Este assunto está ganhando cada vez mais importância, já que somente uma acurácia alta, ou uma baixa taxa de erros em um conjunto de dados de teste não será suficiente para convencer, por exemplo, médicos e pacientes a confiar em uma decisão tomada por um modelo que não possa ser explicada.

1.1 OBJETIVOS GERAIS E ESPECÍFICOS

O objetivo geral deste trabalho é identificar ferramentas ou bibliotecas que possam ser utilizados para produzir visualizações de dados em processos de *Machine Learning*, explorando-as por meio de um estudo de caso.

Como objetivos específicos, tem-se os seguintes:

1. Identificar como a visualização de dados pode ser aplicada nas etapas de um processo de *Machine Learning*.
2. Examinar o uso de recursos de visualização de dados sobre conjunto de dados.
3. Estabelecer relações entre modelos de visualizações e suas aplicações no processo de *Machine Learning*.
4. Implementar visualizações para dois conjuntos de dados, compondo um estudo de caso.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está organizado da seguinte forma:

- O Capítulo 1 consiste em uma introdução do presente trabalho, apresentando a sua motivação e seus objetivos.

- O Capítulo 2 apresenta uma revisão da área de *Machine Learning*, destacando alguns métodos e descrevendo os processos envolvidos.
- O Capítulo 3 apresenta o desenvolvimento deste trabalho de conclusão, ilustrando os modelos de visualizações produzidos em cenários variados.
- Por fim, o capítulo final encerra este trabalho com considerações finais e trabalhos futuros.

2 MACHINE LEARNING E VISUALIZAÇÃO DE DADOS

Na área da Inteligência Artificial estuda-se a capacidade das máquinas extraírem conhecimento dos dados. Esta área denomina-se de *Machine Learning* e compreende uma série de métodos e técnicas para o aprendizado de máquina (MITCHELL, 1997). A aplicação de métodos de *Machine Learning* tornou-se onipresente nos últimos anos na vida cotidiana, estando presente em recomendações automáticas de quais filmes assistir, a que comida pedir ou quais produtos comprar. Muitos *sites* e dispositivos modernos possuem algum processo que envolva *Machine Learning* (ELHAMRAOUI, 2020). Neste capítulo aborda-se os conceitos fundamentais deste tema. A principal ênfase é dada no processo de *Machine Learning*, visando compreender como ele opera e de que forma a visualização de dados pode contribuir para o sucesso desse processo.

2.1 O QUE É *MACHINE LEARNING*?

A área de *Machine Learning* ou aprendizagem de máquina é um subcampo dentro da Inteligência Artificial (IA) que estuda algoritmos de computador utilizados para aprender. Seja com o intuito de completar uma tarefa, para fazer previsões precisas ou simplesmente para se comportar de uma maneira inteligente (MITCHELL, 1997; MÜLLER, 2017). Este aprendizado geralmente é baseado em dados históricos. Pode ser entendido como um conceito amplo, a partir do qual um sistema aprende com as experiências passadas e melhora seu desempenho (SCHAPIRE, 2008).

Segundo Mitchell (1997, p.2) conceitua-se uma tarefa de aprendizado quando um programa de computador aprende com a experiência E, com relação a alguma tarefa T, e alguma medida de desempenho P, se seu desempenho em T, conforme medido por P, melhora com a experiência E.

Por exemplo, um programa de computador que aprende a jogar xadrez (tarefa T) pode melhorar seu desempenho observando partidas de xadrez anteriores ou jogando contra si mesmo (experiência E). Seu desempenho P pode ser medido contando a porcentagem de jogos ganhos contra um humano. Para se ter um problema de *Machine Learning* bem definido, é necessário identificar essas três características: a classe de tarefas, a medida de desempenho a ser melhorada e a fonte de experiência (MITCHELL, 1997).

De forma mais geral, as técnicas de *Machine Learning* compreendem métodos orientados a dados que combinam conceitos fundamentais da ciência da computação com outros das áreas de estatística, probabilidade e otimização (MOHRI; AFSHIN; AMEET, 2012). Essas técnicas tem como grande objetivo implementar algoritmos capazes de aprender, com pouca ou nenhuma necessidade de assistência ou intervenção humana (SCHAPIRE, 2008).

2.2 MOTIVAÇÕES PARA O USO DE *MACHINE LEARNING*

Existem diversas motivações para o uso de mecanismos, métodos e técnicas de *Machine Learning*. Cada área do conhecimento tem utilizado algoritmos com finalidades relevantes. A grande vantagem de utilizar técnicas de *Machine Learning* sobre a programação explícita é o fato de os resultados provenientes serem mais precisos (SCHAPIRE, 2008). A razão por trás disso é que os algoritmos de *Machine Learning* são dirigidos por dados, e como são capazes de analisar uma enorme quantidade de dados saem ganhando quando, colocado em contraponto um humano especialista em um determinado assunto, que pode ser guiado por análises imprecisas e ainda por cima tendo uma quantidade limitada de exemplos para se basear quando comparado a um computador.

A título de exemplificação, pode-se considerar um problema de classificação de mensagens (*spam* ou não *spam*) por um ser humano. Diferente da máquina, ele utiliza premissas, hipóteses, e conhecimento prévio, os quais uma máquina não dispõe. As aplicações que não utilizam *Machine Learning* são contidas basicamente por condições "if"e "else"codificados manualmente. Pode-se considerar, como exemplo, um filtro de *spam*. Nele, o objetivo é identificar e mover *e-mails* em que o seu conteúdo é considerado *spam*, criando uma série de regras, podendo ainda ser utilizada uma lista de palavras removíveis, que resultará em classificar o *e-mail* como *spam* ou não *spam*. Desta maneira, são elaboradas, manualmente, um conjunto de regras que definirão se um *e-mail* será considerado *spam* ou não. Porém, isso traz limitações pois essas regras criadas são específicas para o domínio atual, ou seja, sempre que for necessário adicionar novas regras será preciso realizar a codificação manual, diminuindo sua manutenibilidade e aumentando a complexidade. Some-se a isso o fato de que, para se criar um conjunto de regras eficiente para um problema de classificação de *emails*, seria necessária a classificação de inúmeros exemplares de *e-mails* considerados *spam* por um humano (MÜLLER, 2017). Em contraste, um filtro de *spam* baseado em técnicas de *Machine Learning* pode aprender automaticamente quais palavras e frases são boas preditoras de *spam*, detectando padrões frequentes de palavras, não se baseando apenas em regras específicas (GÉRON, 2019).

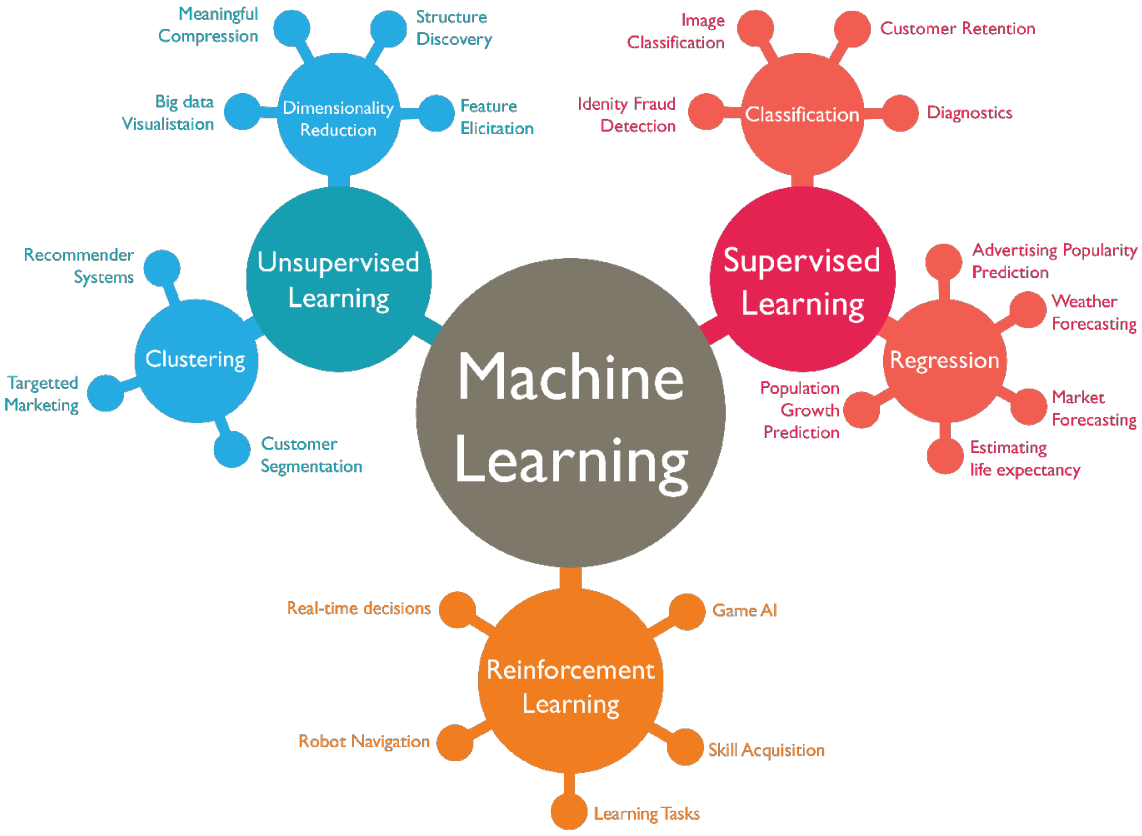
Além disso, o *Machine Learning* pode ser utilizado em problemas muito complexos ou em que não haja algoritmos conhecidos para a resolução destes problemas até o momento. Um exemplo que se enquadra bem é o reconhecimento de um rosto em uma imagem. Hoje em dia todos celulares possuem esta capacidade, no entanto esse era um problema não resolvido até 2001 (CHEN WOLFGANG HäRDLE, 2008). Nesse problema a maior dificuldade é que a maneira em que as imagens são interpretadas pelo computador é muito diferente quando comparado com a maneira em que os humanos interpretam um rosto. Essa diferença de interpretação torna basicamente impossível compilar uma série de regras utilizando programação explícita que consiga identificar o que constitui um rosto em uma imagem digital. Ao utilizar o *Machine Learning* para este problema, apresentando a um programa uma grande coleção de imagens que constituem rostos é o suficiente para um algoritmo determinar quais características são

necessárias para identificar um rosto (MÜLLER, 2017).

2.3 TÉCNICAS DE *MACHINE LEARNING*

As técnicas de *Machine Learning* podem ser classificadas em três tipos, de acordo com o tipo de supervisão que terão durante seu treinamento. Os tipos de aprendizagem podem ser: aprendizagem supervisionada, aprendizagem não-supervisionada e aprendizagem por reforço. Essas três técnicas são ilustradas pela Figura 1 que separa o *Machine Learning* em ramificações. Os tipos de aprendizado supervisionado e não supervisionado são detalhados nas seções seguintes.

Figura 1 – Principais técnicas de *Machine Learning*



Fonte: (WULFF, 2019)

2.3.1 Aprendizado Supervisionado

O aprendizado supervisionado é uma das técnicas de *Machine Learning* mais comumente usada e bem-sucedida. Esta técnica é usada quando o problema que se quer resolver envolve prever um determinado resultado com base em determinadas entradas, tendo como exemplos pares de entrada/saída conhecidos. Um sistema de *Machine Learning*, comumente chamado de modelo, é construído a partir desses pares de entrada/saída, que compõem o conjunto de

treinamento. O objetivo é realizar previsões precisas para dados novos e nunca antes vistos (MÜLLER, 2017). O aprendizado supervisionado frequentemente requer esforço humano para construir o conjunto de treinamento, mas depois se automatiza e geralmente acelera uma tarefa que de outra forma seria trabalhosa ou até inviável.

Os problemas de aprendizagem supervisionada podem ser agrupados em problemas de regressão e classificação. Ambos os problemas têm como objetivo a construção de um modelo que possa predizer um valor. A diferença entre as duas tarefas é que para a regressão o valor previsto é numérico e para a classificação o valor previsto é categórico.

Para distinguir as tarefas entre classificação e regressão basta notar se para o resultado esperado há algum tipo de continuidade. Caso haja continuidade entre os resultados possíveis, então é um problema de regressão. Podemos utilizar o exemplo de prever o preço de um carro. Se o algoritmo de regressão prever R$ 39.999 ou R$ 40.001 quando deveria ter previsto R$ 40.000, não terá uma diferença tangível, apesar de os valores serem diferentes. Por outro lado, em um problema de classificação, como a tarefa de reconhecer se um *e-mail* é *spam* ou não, não há uma questão de grau, um *e-mail* será *spam* ou não, sempre sendo atribuído a alguma classe (MÜLLER, 2017).

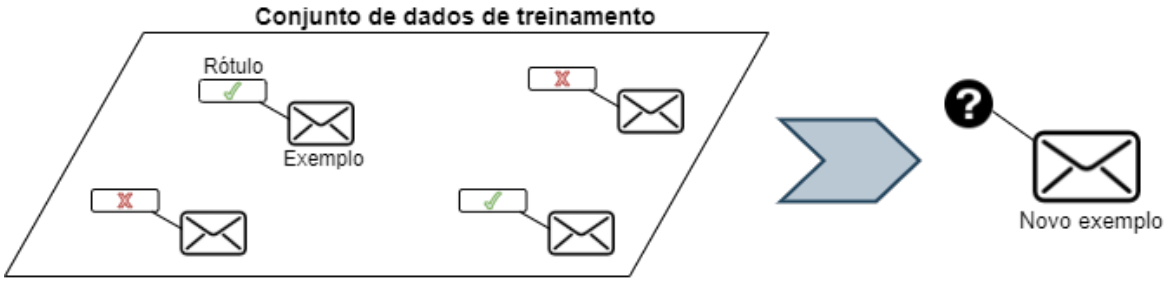
A classificação se refere a um problema de modelagem preditiva em que um rótulo de classe é previsto para um determinado conjunto de dados de entrada. No processo de classificação os dados de treinamento são vários conjuntos de dados e seus respectivos rótulos. Com base no conjunto de dados, o modelo tenta criar um padrão de como os dados são mapeados para os rótulos. Após ser treinado, o classificador atua como uma função que recebe novos conjuntos de dados e produz classificações previstas para eles.

A classificação pode ser separada em classificação binária, que é o caso em que o algoritmo irá distinguir exatamente entre duas classes, e a classificação *multiclasse*, onde a classificação ocorre entre mais de duas classes. Normalmente, as tarefas de classificação binária envolvem uma classe que é o estado normal e outra classe que é o estado anormal. Um exemplo de problema de classificação binária é a classificação de *e-mails* como *spam* ou não *spam* citado anteriormente. A Figura 2 ilustra o processo. Já na classificação *multiclasse* os exemplos são classificados como pertencentes a uma determinada classe entre uma variedade de classes conhecidas.

A regressão é outro problema de modelagem preditiva. O que difere a regressão da classificação é que o resultado esperado é um valor real ou contínuo, como por exemplo o salário ou peso de uma pessoa. Nesta tarefa o objetivo é aproximar uma função de mapeamento (f) de variáveis de entrada (x) para uma variável de saída contínua (y).

Um exemplo de regressão é o de prever o preço de um carro. O modelo é treinado com vários exemplares de carros, que incluem seus preditores, características como a quilometragem, data de fabricação, marca, etc. e seus respectivos rótulos que neste caso seriam os preços. Um

Figura 2 – Um exemplo de classificação de *spam*



Fonte: O Autor (2020)

novo exemplo de carro então seria utilizado como entrada e o modelo faz uma predição de preço para o carro. A Figura 3 ilustra de uma maneira simples a ideia.

Figura 3 – Exemplo de regressão

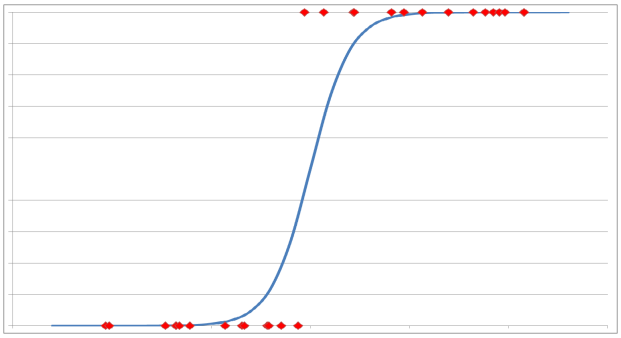
Fonte: (GÉRON, 2019)

Alguns algoritmos serão brevemente discutidos nas próximas sub-seções, tendo em vista que alguns algoritmos de aprendizagem supervisionada podem ser usados tanto para classificação como para regressão.

2.3.2 Regressão Logística

Este algoritmo de classificação utiliza uma função matemática para converter uma ou mais variáveis independentes para determinar um resultado. Esse resultado é medido com uma variável dicotômica, na qual terá apenas dois resultados possíveis (binário) (SIDANA, 2017), ilustrado na Figura 4. O objetivo da regressão logística é encontrar uma relação de melhor ajuste entre a variável de resultado esperada com um conjunto de variáveis independentes.

Figura 4 – Função logística



Fonte: (GERRY, 2017)

2.3.3 K-Nearest-Neighbors

Este algoritmo utiliza a proximidade como um parâmetro de semelhança. O algoritmo faz uso de diversos pontos já rotulados e conhecidos e os usa como base para rotular novos pontos (SIDANA, 2017). Para rotular um novo ponto, o algoritmo procura os pontos rotulados mais próximos desse novo ponto, sendo esses os seus vizinhos mais próximos, por isso o nome do algoritmo. Depois de verificar o número 'k' de vizinhos mais próximos do novo ponto, o rótulo que a maioria dos vizinhos tiver é então atribuído ao novo ponto. A Figura 5 ilustra o processo, onde as estrelas são os pontos novos e as bolinhas os pontos já conhecidos pelo algoritmo. Esse algoritmo pode ser utilizado para resolver problemas de classificação e de regressão.

Figura 5 – Exemplificação do algoritmo *K-Nearest-Neighbors*

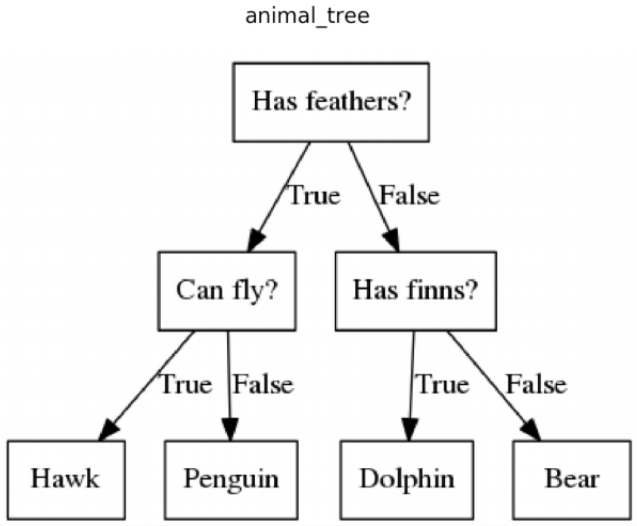
Fonte: (SIDANA, 2017)

2.3.4 Árvore de decisão

As árvores de decisão podem lidar com dados categóricos e numéricos, podendo então ser utilizado nos problemas de classificação e de regressão. Durante o processo desse algoritmo, o conjunto de dados é dividido em subconjuntos cada vez menores, enquanto, ao mesmo tempo,

uma árvore de decisão associada é desenvolvida de forma incremental. Tendo como resultado final uma árvore com nós de decisão e nós folha. Um nó de decisão tem dois ou mais ramos e um nó folha representa uma classificação ou decisão (SIDANA, 2017). A Figura 6 ilustra uma árvore de decisão criada por um modelo.

Figura 6 – Exemplo árvore de decisão

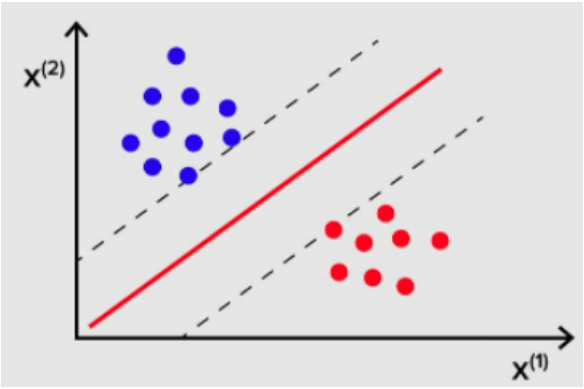


Fonte: (CHOWDARY, 2020)

2.3.5 Máquina de vetores de suporte

A máquina de vetores de suporte é um classificador que representa os dados de treinamento como pontos no espaço. Os pontos são separados em categorias por uma lacuna o mais ampla possível. Novos pontos são então adicionados ao espaço, o algoritmo irá prever em qual categoria estes novos pontos se enquadram e a qual espaço pertencerão. A Figura 7 ilustra um exemplo.

Figura 7 – Exemplo Máquina de vetores de suporte



Fonte: (GAVRILOVA, 2020)

2.4 APRENDIZADO NÃO-SUPERVISIONADO

Neste modelo de aprendizagem, contrário ao do supervisionado, os dados de treinamento não possuem um rótulo indicando o resultado esperado. Além disso, os dados podem possuir uma estrutura desconhecida, então nesta abordagem os modelos chamados de descritivos não recebem uma tarefa específica (MÜLLER, 2017). Por meio de técnicas de aprendizado não-supervisionado é possível criar visualizações, compactar os dados e explorar a estrutura dos dados para extrair informações significativas sem a orientação de uma variável de resultado conhecida ou uma função de recompensa.

Os problemas de aprendizagem não-supervisionada podem ser divididos em problemas de agrupamento e de redução de dimensionalidade. A tarefa de agrupamento tem como objetivo separar um conjunto de objetos de forma que esses objetos estejam no mesmo grupo, por serem mais semelhantes entre si do que em outros grupos. Já na tarefa de redução de dimensionalidade o objetivo não é procurar categorias distintas nos dados, em vez disso, a ideia é que no conjunto de dados algumas características podem ser desnecessárias, redundantes ou podem ser mescladas, tendo como objetivo então simplificar o conjunto de dados com uma perda mínima de informações (CADY, 2017).

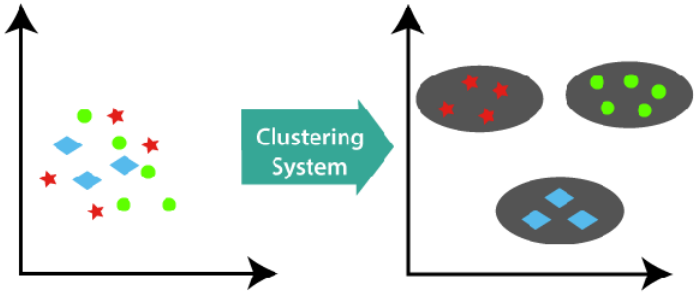
A diferença entre o agrupamento e a redução de dimensionalidade é que o agrupamento geralmente é aplicado para revelar a estrutura dos dados. Já a redução de dimensionalidade costuma ser motivada principalmente por questões computacionais. Por exemplo, ao processar arquivos de som, imagem ou vídeo, provavelmente o conjunto de dados será muito grande e para processar todos esses dados a tarefa computacional será de igual proporção, assim a redução da dimensionalidade atua como forma de simplificar o conjunto de dados com uma perda mínima de informações.

A tarefa de agrupamento é uma técnica de análise exploratória de dados, que permite organizar um monte de informações em subgrupos significativos sem ter qualquer conhecimento prévio de suas associações de grupo (RASCHKA, 2017). Cada grupo que surge durante a análise define um grupo de objetos que compartilham um certo grau de similaridade e são mais diferentes do que os objetos em outros grupos. É uma técnica para estruturar as informações e obter relacionamentos significativos a partir de dados.

A Figura 8 ilustra como o agrupamento pode ser aplicado para organizar dados não rotulados em três grupos distintos com base na semelhança de suas características.

A redução da dimensionalidade se refere a técnicas que reduzem o número de variáveis de entrada de um conjunto de dados. A quantidade de variáveis de entrada de um conjunto de dados é conhecido como dimensionalidade. Se um conjunto de dados possui uma quantidade muito grande de variáveis de entrada, uma tarefa de modelagem preditiva pode se tornar muito complexa para se modelar. Essas técnicas de aprendizagem não-supervisionada são também frequentemente usadas para a visualização de dados. No entanto, essas técnicas podem ser usadas

Figura 8 – Exemplo Agrupamento



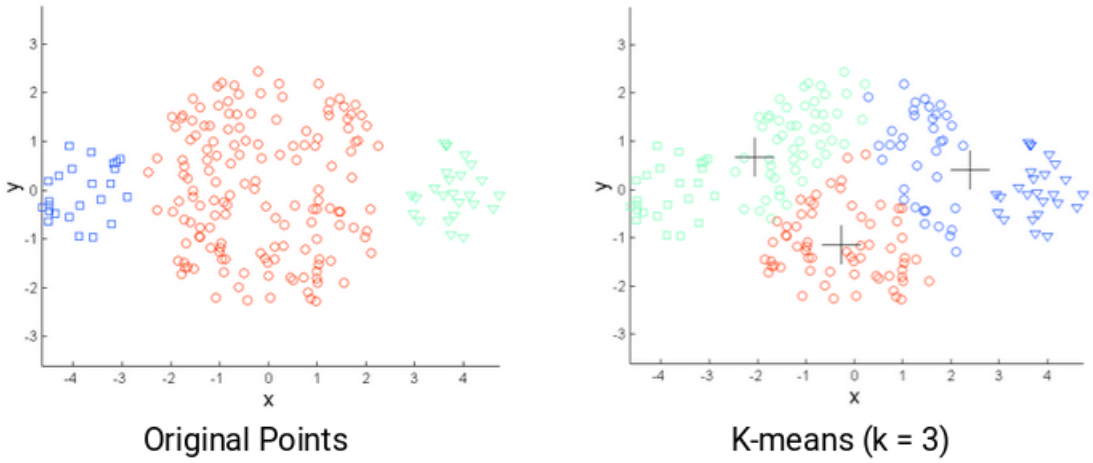
Fonte: (MISHRA, 2019)

para simplificar um conjunto de dados de classificação ou regressão a fim de se ajustar melhor a um modelo preditivo.

2.4.1 *K-Means*

O *K-Means* é um algoritmo de agrupamento. O algoritmo recebe como parâmetro um número k, esse número indica o número de centróides que devem ser criados no conjunto de dados. O centróide é a localização imaginária ou real que representa o centro de um dos grupos. O algoritmo começa com grupos de centróides que são selecionados aleatoriamente, usados como pontos iniciais para cada grupo. Em seguida, com base no conjunto de dados, realiza cálculos iterativos para otimizar as posições dos centróides. O algoritmo irá parar de criar e otimizar os grupos quando não haver mais mudança nos valores ou o número definido de iterações for alcançado (MCGREGOR, 2020). A Figura 9 ilustra um conjunto de dados após ser processado pelo algoritmo.

Figura 9 – Exemplo *K-Means*

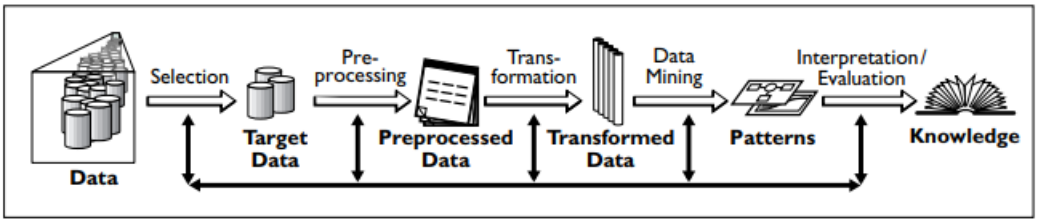


Fonte: (SHARMA, 2019)

2.5 ETAPAS DO PROCESSO DE *MACHINE LEARNING*

Antes de um modelo ser capaz de realizar predições confiáveis, o sistema de *Machine Learning* passa por um processo iterativo e pode ser baseado no modelo de (FAYYAD, 1996) para a descoberta de conhecimento, ilustrado na Figura 10. O processo compreende seis principais etapas, iniciando com a definição do problema, seguindo pela a coleta e a compreensão dos dados disponíveis, após é necessário realizar um pré-processamento dos dados para que possam ser utilizados. Com base nos dados e no problema em questão, deve-se escolher uma técnica de *Machine Learning* apropriada. Após isso, treinar e avaliar o modelo escolhido, para por fim, apresentar os resultados provenientes do modelo.

Figura 10 – Processo descrito pro Fayyad



Fonte: (FAYYAD, 1996)

Cada uma destas etapas é descrita nas seções que seguem, tendo como base os processos indicados por (FAYYAD, 1996) e por (PANT, 2011).

2.5.1 Definindo o problema

Para começar o processo de *Machine Learning* a primeira etapa é a formulação de um problema que possa ser resolvido através dos dados. A maioria dos problemas do mundo real que podem ser resolvidos por um modelo de *Machine Learning* tem como objetivo a previsão de uma variável ou variáveis de interesse. Nesta etapa o objetivo é definir uma classe de problemas que engloba formas de aprendizagem, para então poder explorar algoritmos que possam resolver tais problemas (MITCHELL, 1997). Esta etapa é importante porque vai determinar quais algoritmos podem ser utilizados e qual será a forma de avaliar o desempenho do modelo criado (GÉRON, 2019).

2.5.2 Coleta e compreensão dos dados

A coleta de dados é muito importante, pois a qualidade e a quantidade dos dados que serão utilizados impactam diretamente em quão eficiente será o modelo escolhido. O conjunto de dados pode ser obtido através de várias fontes, como por exemplo, um simples arquivo *CSV*, através de um bancos de dados, sensores de dispositivos de *Internet of Things* e muitas outras fontes.

Depois da coleta dos dados, é preciso então ter uma melhor compreensão do conjunto de dados para poder definir qual modelo aplicar e quais os resultados esperados. Antes de construir um modelo de *Machine Learning*, é uma boa prática inspecionar os dados, para ver se a tarefa pode ser facilmente resolvida sem um processo de *Machine Learning* ou se as informações desejadas podem não estar contidas nos dados. Além disso, inspecionar os dados é uma boa maneira de encontrar anormalidades e peculiaridades. No mundo real, inconsistências nos dados e medições inesperadas são muito comuns.

Então para se ter uma melhor compreensão do conjunto de dados, a visualização de dados pode ser utilizada, possibilitando mostrar por meio de gráficos a estrutura básica dos dados, dando possibilidade de identificar pontos fora da curva e de encontrar relações entre os atributos presentes no conjunto de dados (CODECADEMY, 2020). Com técnicas de visualização simples, é possível explorar a relação entre os atributos e a variável de saída esperada, auxiliando na construção e compreensão do modelo de *Machine Learning* e das previsões realizadas por ele (BRINK, 2016).

É muito importante entender quais os tipos de dados disponíveis, o que está faltando nos dados e como lidar com isso, como adicionar, alterar ou remover recursos para obter mais dos dados para propriamente preparar os dados.

2.5.3 Pré-processamento dos dados

Geralmente, os dados obtidos não estão preparados para serem utilizados diretamente, pois podem possuir elementos ausentes, desorganizados, com valores grandes ou pequenos demais e podem conter bastante ruído. Portanto, para aumentar as chances de eficácia do processo de *Machine Learning*, é crucial que haja um pré-processamento dos dados. Tarefas como a limpeza, preparação e manipulação dos dados, normalização e padronização entre os dados, removendo dados que não são relevantes para o modelo ou até aumentando os dados adicionando novas colunas relevantes, são tarefas fundamentais. Desta maneira o conjunto de dados fica mais limpo e claro o possível para o modelo, tendo em vista que, será com base no conjunto de dados que o modelo fará previsões para novos dados (KOTSIANTIS; KANELLOPOULOS; PINTELAS, 2006; PANT, 2011).

Não é incomum em aplicações do mundo real que falte um ou mais valores nas amostras de conjuntos de dados. Há uma grande possibilidade de ter ocorrido um erro no processo de coleta de dados, ou certas medidas não serem aplicáveis ou determinados campos podem ter sido simplesmente deixados em branco em uma pesquisa, por exemplo. E por isso as amostras podem conter valores ausentes como espaços em branco, strings de espaço reservado como *NaN*, que não representa um número, ou *NULL* que é um indicador comumente usado de valores desconhecidos em bancos de dados relacionais (RASCHKA, 2017). Infelizmente, a maioria das ferramentas computacionais não são capazes de lidar com tais valores ausentes ou podem produzir resultados imprevisíveis se esses valores forem ignorados. Portanto, é crucial que esses

valores ausentes sejam tratados antes de se prosseguir para as próximas etapas.

Além disso, alguns algoritmos de *Machine Learning* exigem que os dados sejam normalizados, o que significa que cada ponto do conjunto de dados precisa estar na mesma escala numérica. O intervalo de valores desses pontos pode influenciar na importância quando uma comparação for realizada. Se um ponto tiver valores entre 0 e 10 e outro tiver valores entre 0 e 1, o peso do primeiro é 10 em comparação com o segundo (BRINK, 2016). Então para garantir que todos os pontos sejam considerados igualmente, é preciso normalizar os dados.

Outro recurso necessário é a avaliação dos atributos ou características relevantes dos dados para o tipo de tarefa visada (classificação, regressão ou agrupamento). Alguns conjuntos de dados possuem um número excessivo de atributos, sendo desnecessários aos processos de *Machine Learning*. A redução da dimensionalidade se refere a técnicas que reduzem o número de atributos de entrada de um conjunto de dados. A quantidade de atributos de entrada de um conjunto de dados é conhecido como dimensionalidade. Se um conjunto de dados possui uma quantidade muito grande de atributos, uma tarefa de modelagem preditiva pode se tornar muito complexa para se modelar. Essa técnica também é frequentemente usada para a visualização de dados.

2.5.4 Escolhendo um modelo apropriado

A próxima etapa no processo de *Machine Learning* é usar os dados pré-processados para começar a construção de modelos estatísticos com base nos dados, tendo o problema definido e com alguns possíveis *insights* obtidos nas etapas anteriores é possível começar a escolher um modelo apropriado. Os modelos podem variar de acordo com o conjunto de dados disponível, já que, por exemplo, os algoritmos de classificação e regressão tem uma eficácia melhor em diferentes conjuntos de dados.

O objetivo do *Machine Learning* é descobrir padrões e relações entre os dados e aplicar essas descobertas em dados novos. Esse processo de descoberta é alcançado por meio do uso de técnicas de modelagem que foram desenvolvidas nos últimos 30 anos em estatística, ciência da computação e matemática aplicada (BRINK, 2016). Essas várias abordagens podem variar de simples a extremamente complexas, mas todas compartilham um objetivo comum, que é estimar uma relação funcional entre os dados de entrada e uma variável alvo. Essas abordagens usam dados históricos para construir e otimizar um modelo que, por sua vez, é usado para fazer previsões com base em novos dados.

Para ajudar na escolha de um modelo apropriado algumas questões em relação ao conjunto de dados podem ser levantadas. Como por exemplo, se os dados são categóricos ou numéricos. Dependendo da resposta alguns modelos não poderão trabalhar diretamente no conjunto de dados, podendo ser necessário uma transformação nos dados. Caso os dados sejam lineares, um modelo de regressão linear ou regressão logística pode ser utilizado. Caso não

seja, o modelo de redes neurais pode ser uma boa escolha. Se há um limite de tempo em que o modelo pode ser treinado, alguns modelos como redes neurais são conhecidos por possuir um treinamento mais lento. Já modelos de regressão linear ou modelos baseados em árvore de decisão são muito mais rápidos. Se a explicabilidade dos resultados é um fator importante, entre os modelos mais precisos comumente chamados de caixa-preta, pode ser difícil de se entender o porquê do modelo ter feito uma previsão específica e ainda mais difícil de se explicar. Um exemplo de modelo caixa-preta é o modelo de redes neurais. Por outro lado os modelos como regressão linear e *K-Nearest-Neighbors* nem sempre são os mais precisos, porém suas previsões são muito diretas (BURKOV, 2019).

2.5.5 Treinando e avaliando o modelo

Nesta etapa o conjunto de dados que foi pré-processado anteriormente é dividido em dois, onde uma das partes é utilizada para treinar o modelo e a outra para testar e avaliar o desempenho do modelo. O conjunto de dados utilizado no treinamento é o material em que o modelo aprenderá a processar novas informações, ajustando suas métricas para utilizá-las em futuras predições.

Assim que o treinamento do modelo tiver terminado é então aplicado o conjunto de dados de teste no modelo. Como esses dados são desconhecidos para o modelo, será possível medir a sua performance com base nos resultados obtidos. Com os resultados provenientes é possível então avaliar a precisão deste modelo. Antes de colocar um modelo em uso, é preciso saber quão bem ele irá prever novos dados. Se o desempenho preditivo do modelo é bom, é possível utilizar esse modelo em dados reais para analisar dados novos. Da mesma forma, se o desempenho preditivo não for bom o suficiente para a tarefa em questão, é preciso revisitar as etapas anteriores para tentar melhorar e otimizar o desempenho do modelo (BRINK, 2016).

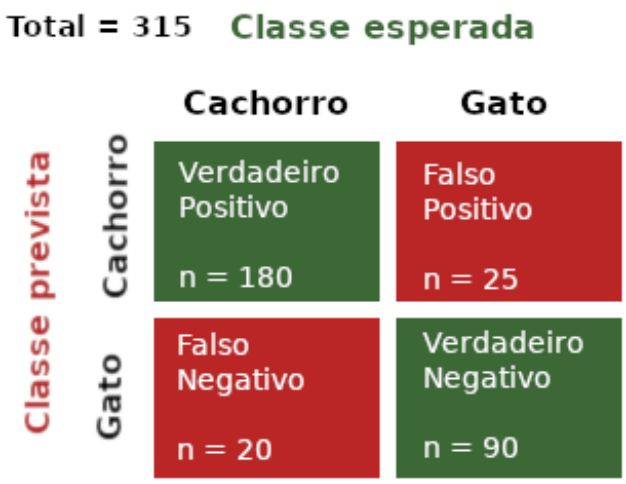
Para avaliar a performance do modelo a visualização de dados pode ser utilizada como ferramenta. Utilizando as métricas do modelo, combinado com visualizações, é possível obter melhores *insights* em torno do desempenho. Há diversos tipos de visualização que são possíveis dependendo do tipo de tarefa que está sendo executada pelo modelo.

Por exemplo, uma maneira comum de analisar as métricas de desempenho de um classificador binário é através de uma matriz de confusão. A matriz de confusão é uma matriz de 2 linhas e 2 colunas que mostra quantos pontos dos dados de teste foram colocados em qual categoria e em qual categoria eles deveriam ter sido colocados, como a Figura 11 demonstra.

Com base na matriz de confusão é possível calcular a acurácia, que é uma métrica utilizada para avaliar modelos de classificação. Formalmente, a acurácia tem a definição estipulada pela seguinte fórmula:

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN}$$

Figura 11 – Matriz de confusão binária



Fonte: O Autor (2020)

Onde VP = verdadeiros positivos, VN = verdadeiros negativos, FP = falsos positivos e FN = falsos negativos. Resumindo, a acurácia é o número de previsões corretas (VP e VN), dividido pelo número de todas as predições do conjunto de dados. Se aplicarmos a fórmula da acurácia para a matriz de confusão da Figura 11, teremos o seguinte resultado:

$$\text{Acurácia} = \frac{180 + 90}{180 + 90 + 25 + 20} = 0,85$$

A acurácia do modelo é alta de 85%, nesse modelo de classificação de cães e gatos não haverá consequências severas caso haja um falso negativo ou um falso positivo. Porém, se o objetivo do modelo fosse classificar uma pessoa que está doente e infectada com o novo coronavírus por exemplo, haveria um grande problema em classificar incorretamente uma pessoa como falso negativo. Portanto a acurácia não é a métrica de modelo fundamental a ser usada para selecionar o melhor modelo.

Além da acurácia, através da matriz de confusão é possível obter também a precisão e a revocação. A precisão mede quantas das predições que foram previstas como positivas são realmente positivas:

$$\text{Precisão} = \frac{VP}{VP + FP}$$

A precisão é usada como uma métrica de desempenho quando o objetivo é limitar o número de falsos positivos. Por exemplo, na detecção de *spam* de *e-mails*, se ocorrer um falso positivo, ou seja um *e-mail* que não é *spam* for identificado como *spam*, o usuário final pode perder *e-mails* importantes caso a precisão não for alta o suficiente.

A revocação por outro lado, mede quantas das predições positivas são capturadas pelas previsões positivas:

$$\text{Revocação} = \frac{VP}{VP + FN}$$

A revocação é usada como métrica de desempenho quando é preciso identificar todas as amostras positivas sem exceção, ou seja, quando é importante evitar falsos negativos. Um bom exemplo seria novamente a detecção do coronavírus, ou de um diagnóstico de câncer, é muito importante encontrar todas as pessoas que estão doentes, possivelmente incluindo pacientes saudáveis na previsão realizada.

Com base na precisão e na revocação é possível obter outra métrica, o *F1-Score*. Esta métrica é calculada através da média harmônica entre a precisão e a revocação. O *F1-Score* pode ser utilizado quando é preciso ter um equilíbrio entre a precisão e a revocação. Como citado anteriormente, mesmo com uma acurácia alta é possível ter um grande número de falsos negativos ou falsos positivos, que podem ter um impacto indesejado dependendo do cenário. A fórmula do *F1-Score* é a seguinte:

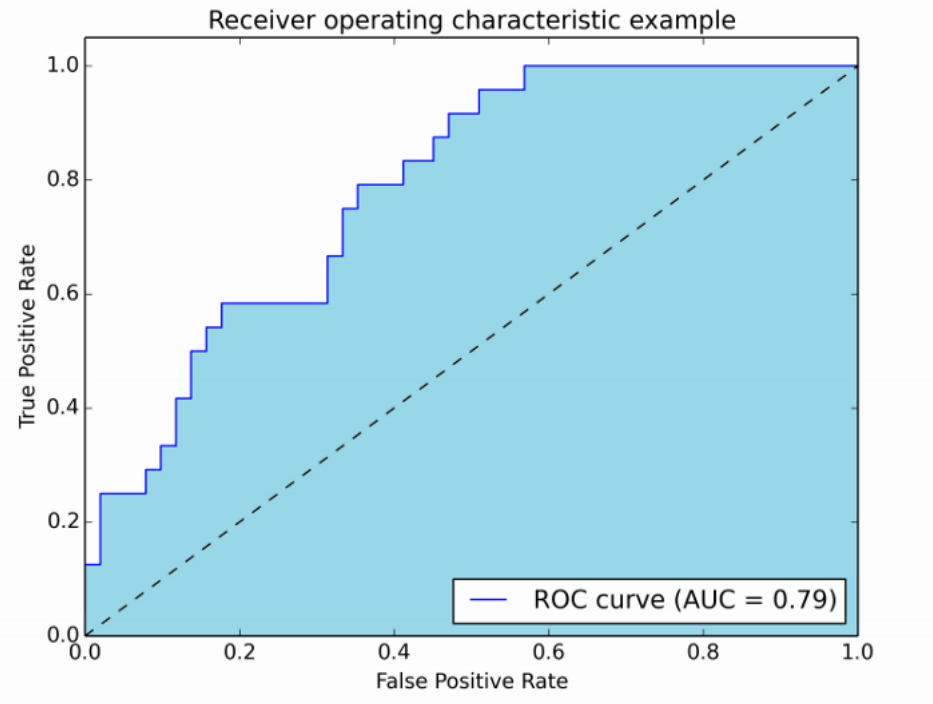
$$\textit{F1-Score} = 2 * \frac{\text{Precisão} * \text{Revocação}}{\text{Precisão + Revocação}}$$

Outra maneira de avaliar o desempenho de um modelo de classificação é através da curva *ROC*, que deriva do inglês *Receiver Operating Characteristic*. A curva *ROC* representa a taxa de verdadeiros positivos versus a taxa de falsos positivos em diferentes limiares de classificação. A taxa de verdadeiros positivos nada mais é do que um sinônimo para a revocação, já a taxa de falsos positivos pode ser calculada por meio da seguinte fórmula:

$$\text{Taxa de falsos positivos} = \frac{FP}{FP + VN}$$

Através da curva *ROC*, é possível medir o valor da área sob a curva, comumente chamada de *AUC*, do inglês *Area Under the Curve*. A *AUC* mede toda a área bidimensional abaixo de toda a curva *ROC*. A Figura 12 ilustra a curva *ROC* em azul e a área abaixo da curva representa a *AUC*. A *AUC* é uma maneira de resumir a curva *ROC* em um único valor, agregando todos os limiares da curva *ROC*, calculando a área sob a curva. Um modelo cujas previsões estão 100% erradas tem uma *AUC* de 0, enquanto um modelo onde as previsões são 100% corretas tem uma *AUC* de 1.

Figura 12 – Curva *ROC* e *AUC*



Fonte: (RODRIGUES, 2020)

2.5.6 Apresentando os resultados do modelo

Com os resultados provenientes do modelo treinado, é preciso compreender e poder explicar esses resultados. Os resultados precisam fazer sentido para a resolução do problema proposto para que seja possível utilizá-lo para conduzir decisões baseadas em dados. Nesta etapa a visualização de dados desempenha um papel muito importante, auxiliando na apresentação dos dados de uma forma que dê razão e passe transparência do modelo. Mostrar somente uma precisão alta ou baixos erros em amostras de teste não será suficiente para passar confiança de que o modelo está funcionando corretamente. Explicações adequadas sobre o que o modelo está fazendo e mostrar do por que os resultados são o que são podem então ser descritos de uma forma visual para pessoas não técnicas.

2.6 VISUALIZAÇÃO DE DADOS PARA *MACHINE LEARNING*

Há uma expressão popular, atribuída a Confúcio, que diz que uma imagem vale mais que mil palavras. Ela é comumente utilizada para apresentar a ideia e o poder da comunicação através das imagens em geral. Na computação, a área que trata da comunicação por meio de imagens variadas é denominada de visualização de dados ou informações. Em *Machine Learning*, a visualização de dados é a área de conhecimento que se preocupa em buscar representações

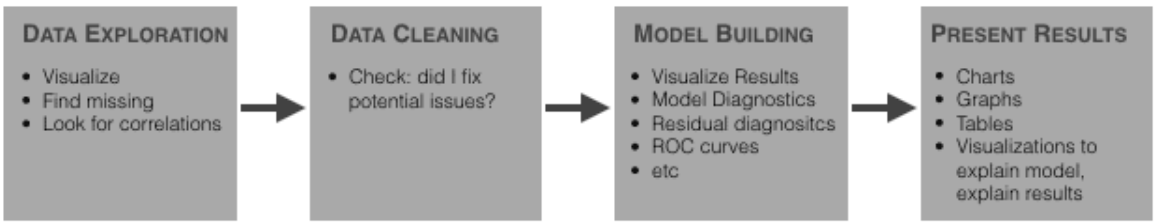
gráficas apropriadas de um conjunto de dados, visando auxiliar nos processos de análise para uma melhor compreensão humana (WARE, 2012).

O objetivo da visualização de dados é representar visualmente as informações, auxiliando assim na interpretação dos dados, aproveitando o sistema visual dos seres humanos para reconhecer padrões. Ela possibilita a descoberta de *insights* para então permitir conclusões e a tomada de melhores decisões (JEFFREY; MICHAEL; VADIM, 2010). Desta forma, o processo de visualização de dados combina habilidades humanas (visão e interpretação) e de máquina, deixando os cálculos complexos para serem realizados pelos computadores.

A visualização de dados pode ser usada nas diferentes etapas do processo de *Machine Learning*, e desempenha um papel fundamental. A visualização de dados é uma das principais ferramentas utilizadas para analisar e estudar as relações entre diferentes atributos, providenciando uma melhor compreensão dos dados que serão tratados. Pode ser usada para análises descritivas e no pré-processamento de um conjunto de dados, na seleção de atributos que tem grande influência no resultado esperado, na própria construção, testes e avaliação do modelo. Por fim, as visualizações são frequentemente usadas para ajudar os usuários finais a tomarem decisões com base nos dados, onde fatores como a validação dos resultados da análise desempenha um papel muito importante (SAID, 2018).

A Figura 13 ilustra algumas possibilidades do uso de visualizações durante o processo de *Machine Learning*. Na primeira etapa, ocorre a exploração dos dados, onde diferentes visualizações do conjunto de dados auxiliam a descoberta de valores ausentes, valores grandes ou pequenos demais e possibilita encontrar correlações entre os atributos do conjunto de dados. Na segunda etapa, com os problemas no conjunto de dados definidos, é preciso realizar uma limpeza nos dados para seguir para a construção do modelo. Para a construção de modelos, a visualização assume um papel fundamental, auxiliando na apresentação dos resultados. A visualização compreende gráficos, diagramas, tabelas, e todo o tipo de visualizações que permitam apresentar e explicar os dados de maneira minuciosa e atenta.

Figura 13 – Visualizações em etapas do processo de *Machine Learning*



Fonte: (GODARD, 2016)

O processo de *Machine Learning* inicia com a constatação de um problema. A partir da definição do problema já definida, é preciso ter uma descrição básica do conjunto de dados. Para isso, por meio das visualizações é possível colocar os dados em contexto com o problema.

Ao visualizar as colunas em do conjunto de dados, é possível obter uma compreensão melhor de cada coluna, diferenciando as variáveis entre categóricas ou numéricas, e distinguindo se os dados são variáveis independentes ou dependentes. Todas essas observações devem ser feitas com o intuito de se obter uma noção básica dos dados.

2.6.1 Exemplo de visualizações do conjunto de dados Iris

Para fins de exemplificação e demonstração dos testes realizados nesta seção, foi utilizado um famoso conjunto de dados Iris sobre flores (lírios), desenvolvido por (FISHER, 1936). O conjunto de dados possui 4 atributos, sendo eles: o comprimento da sépala e da pétala, e a largura da sépala e da pétala. Como atributo classificador, tem-se três tipos de flores: iris setosa, iris versicolor e iris virgínica.

A fim de ilustrar visualizações sobre estes dados, realizou-se testes caracterizando um experimento simples. A manipulação dos dados ocorreu por meio de um pacote denominado de *Pandas*[1], para a linguagem de programação *Python*[2]. Além de facilitar a importação e análise dos dados, o Pandas ainda fornece métodos que realizam a descrição do conjunto de dados. Ao utilizar-se o comando .describe() sobre o conjunto de dados, algumas estatísticas descritivas são apresentadas. A Figura 14 ilustra a saída da função describe(), que exibe a contagem, a média, o desvio-padrão, os valores mínimo e máximo e os quartis dos dados. Nota-se que, neste caso, os atributos do conjunto de dados são definidos pelo comprimento e a largura das sépalas e pétalas das flores.

Figura 14 – Estatísticas descritivas geradas do conjunto de dados Iris

Out[1]:

	Sepal_length	Sepal_width	Petal_length	Petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

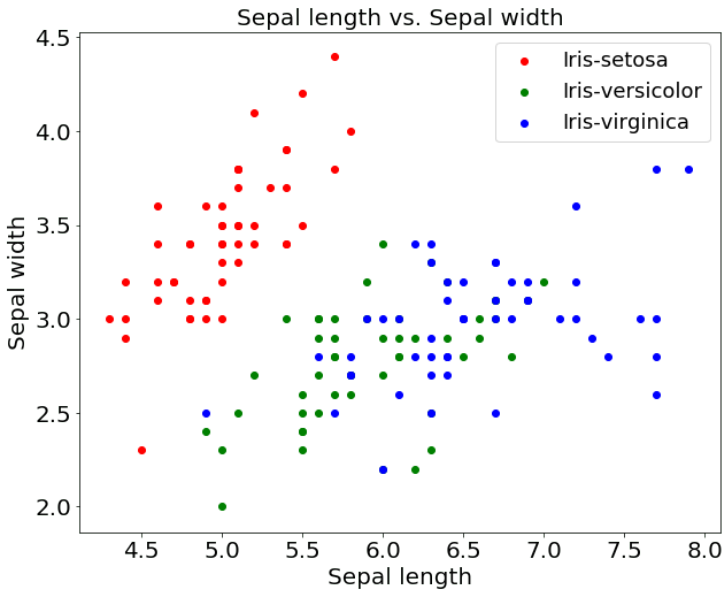
Fonte: (WILLEMS, 2017)

Com base nessas estatísticas descritivas é possível avaliar a qualidade do conjunto de dados, para então, decidir se correções são necessárias, se é preciso descartar ou lidar com os dados de outra forma. Esta etapa visa entender os elementos do conjunto dados e suas anomalias um pouco melhor e ver se os dados se adaptam às necessidades do problema em questão (WILLEMS, 2017).

[1] https://pandas.pydata.org
[2] https://www.python.org

Ainda com o intuito de analisar os dados, é possível descobrir o quanto uma variável é afetada por outra variável, ou qual o nível de correlação que existe entre duas variáveis. Por meio de um pacote denominado de *matplotlib*[3], que é utilizado para criar visualizações para *Python*, é possível criar um gráfico de dispersão do conjunto de dados citado anteriormente. A Figura 15 mostra que há uma alta correlação entre as flores de iris do tipo setosa, com o comprimento e a largura das sépalas. Por outro lado, há menos correlação entre as espécies iris versicolor e iris virgínica. Os pontos de dados das espécies versicolor e virgínica estão mais espalhados do que os da setosa, que são densos (SHARMA, 2018).

Figura 15 – Gráfico de dispersão, comprimento vs largura das sépalas do conjunto de dados Iris

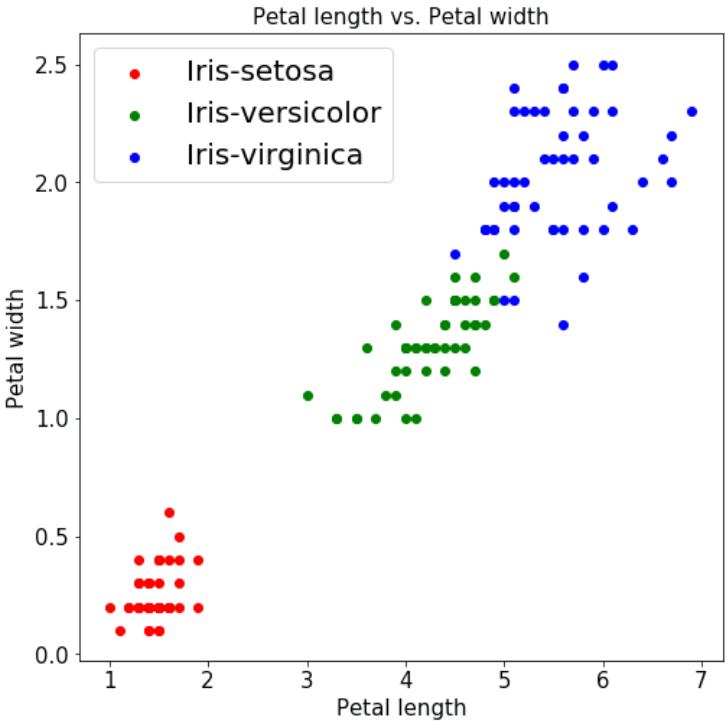


Fonte: (SHARMA, 2018)

Ao se analisar o comprimento e largura, agora da variável que diz respeito as pétalas dos lírios, o gráfico ilustrado pela Figura 16 também indica uma forte correlação para as flores da classe setosa que se encontram novamente densamente agrupadas.

[3] https://matplotlib.org

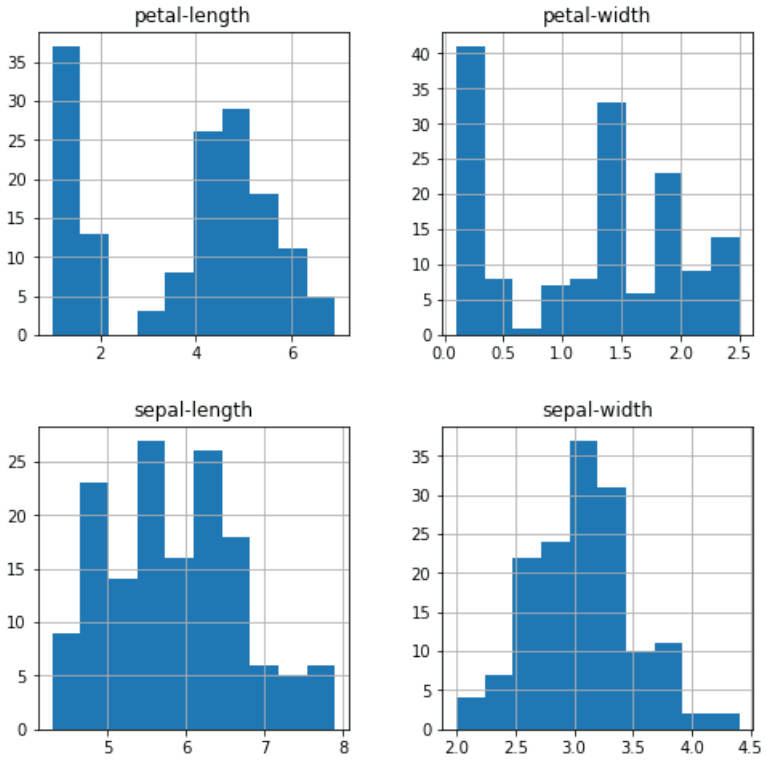
Figura 16 – Gráfico de dispersão, comprimento vs largura das pétalas do conjunto de dados Iris



Fonte: (SHARMA, 2018)

Utilizando histogramas, pode-se ainda analisar a distribuição dos atributos no conjunto de dados. A Figura 17 ilustra os histogramas gerados por meio do pacote *matplotlib*. Os gráficos gerados para o comprimento da pétala, largura da pétala e comprimento da sépala mostram uma distribuição unimodal. Enquanto o que foi gerado para a largura da sépala mostra um tipo de distribuição Gaussiana. Essas análises são úteis na hora de se escolher um algoritmo que funcione bem com esse tipo de distribuição.

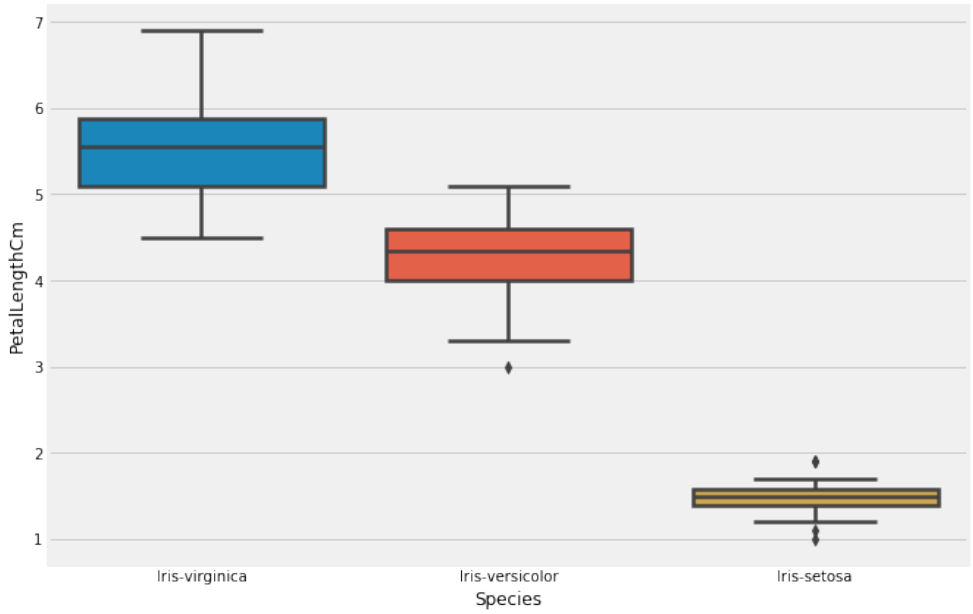
Figura 17 – Histograma dos atributos disponíveis no conjunto de dados Iris



Fonte: (SHARMA, 2018)

Além disso, por meio da análise de distribuição dos atributos pode-se encontrar valores que divergem fortemente da grande maioria dos outros dados. Esses dados são chamados de *outliers*, ou pontos fora da curva normal. As causas de valores discrepantes nos dados podem variar, desde erros de sistema até pessoas interferindo nos dados por meio de entrada ou processamento de dados, porém, é importante considerar o efeito que eles podem ter. Esses valores podem mudar o resultado dos testes estatísticos como o desvio padrão e a média e potencialmente diminuir a normalidade e impactar nos resultados finais de modelos estatísticos (GODARD, 2016). Com base nisso o *boxplot*, ou diagrama de caixa fornece um resumo estático dos atributos sendo plotados. A Figura 18 ilustra um *boxplot*, no qual o conjunto de dados é o das flores lírios. Os pontos pretos no gráfico representam os valores atípicos no conjunto de dados.

Figura 18 – *Boxplot* do conjunto de dados Iris



Fonte: (BINU, 2020)

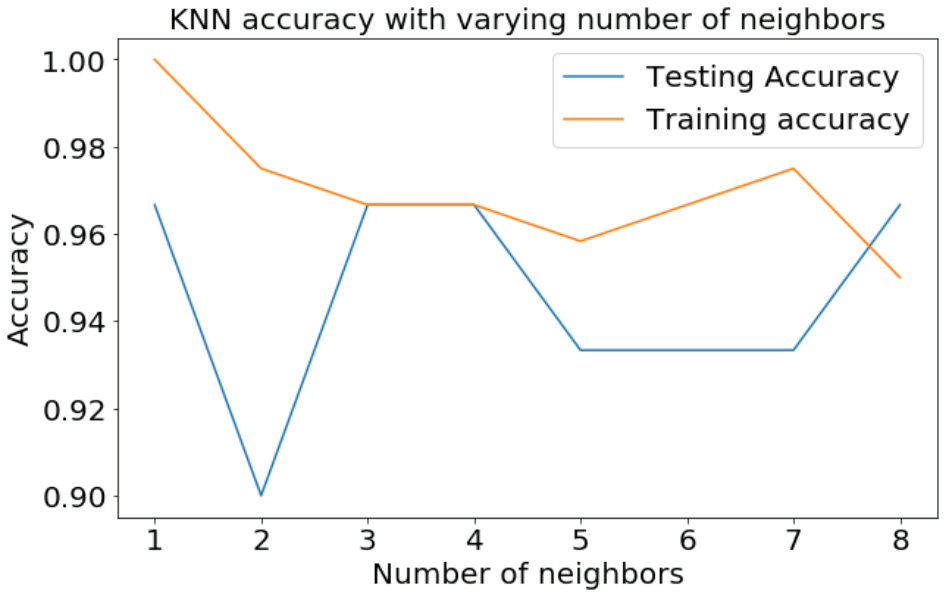
Após as etapas de análise e pré-processamento dos dados, a próxima etapa é fazer uso dos dados em conjunto com um modelo de *Machine Learning*. Para isso, é preciso dividir o conjunto de dados em dados de treinamento e dados de teste. O conjunto de dados das flores iris possui 150 amostras, será utilizado 80% dos dados para treinamento e os 20% restantes serão usados para teste. A título de exemplificação, um modelo com base no algoritmo *K-Nearest-Neighbors* será treinado e avaliado. Para isso, um pacote chamado *scikit-learn*[4], que possui vários algoritmos de classificação, regressão e de agrupamento auxiliará na implementação.

Como visto anteriormente, o algoritmo *K-Nearest-Neighbors* usa a proximidade como parâmetro de semelhança, e possui um parâmetro k que define a quantidade de dados vizinhos a serem considerados. Para decidir o melhor valor para o parâmetro k, o modelo foi treinado com 10 diferentes valores para k. Após o modelo ser treinado, é realizada uma previsão em cima dos dados de teste. Através dos resultados provenientes, é possível representar graficamente a precisão dos modelos em cima do conjunto de dados de teste, com o auxílio do pacote *matplotlib*. A Figura 19 mostra o gráfico de precisão versus o número da variável k, auxiliando na escolha do melhor valor de k para utilizar no modelo.

Por fim, para descrever o desempenho do modelo nos dados de teste, uma matriz de confusão será gerada. Novamente, o pacote *scikit-learn* auxiliará, já que possui uma função que calcula a matriz de confusão. Ao observar o gráfico gerado na Figura 20, nota-se que o modelo classificou quase todas as flores corretamente, com exceção de uma flor versicolor que foi classificada como uma flor virgínica.

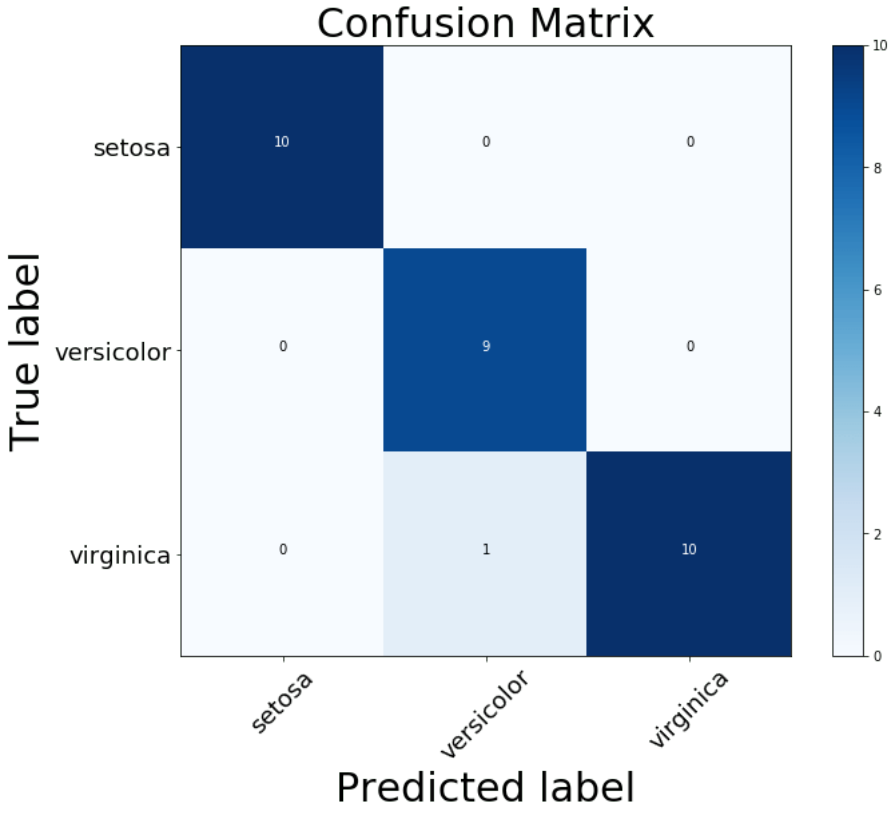
[4] https://scikit-learn.org

Figura 19 – Gráfico de precisão versus o número da variável k



Fonte: (SHARMA, 2018)

Figura 20 – Matriz de confusão do modelo

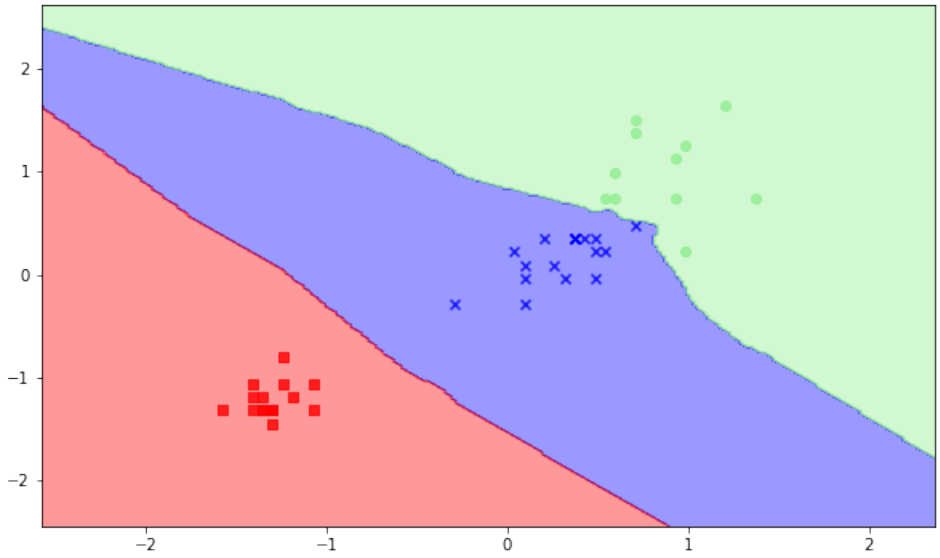


Fonte: (TAYO, 2020)

Para melhor uma compreensão dos resultados, é possível visualizar como o modelo classificou as amostras no conjunto de dados de teste. A Figura 21 gerada por meio do pacote

matplotlib ilustra o resultado final de classificação proveniente do modelo implementado.

Figura 21 – Resultado final da classificação do modelo implementado



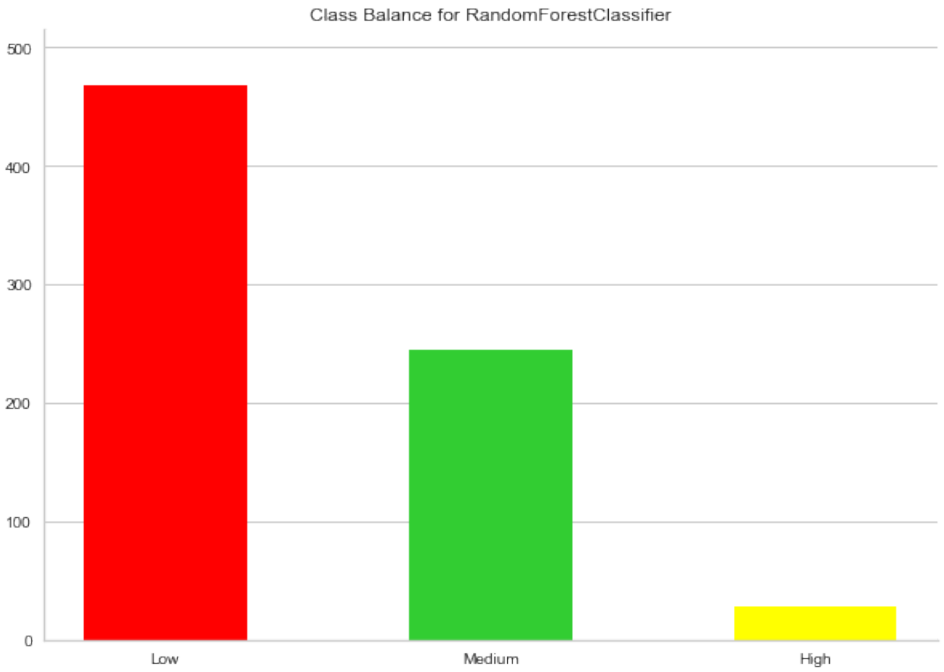
Fonte: (KERR, 2016)

2.6.2 Exemplos de visualização de processos de *Machine Learning*

Algumas visualizações são especialmente importantes em um processo de *Machine Learning*. Os histogramas podem ser úteis no processo de desenvolvimento de um modelo classificador, que tem como alvo uma variável categórica. É importante analisar o conjunto de dados procurando um desequilíbrio entre as classes, onde uma classe está super-representada, enquanto a outra está sub-representada. Dados desequilibrados afetam diretamente o desempenho final do modelo. A Figura 22 ilustra um conjunto de dados onde há um desequilíbrio de classe, ficando visível uma grande discrepância entre as classes "Low"e "High". Este gráfico foi gerado por meio do pacote denominado *Yellowbrick*[5] para a linguagem de programação *Python*.

[5] https://www.scikit-yb.org/en/latest/

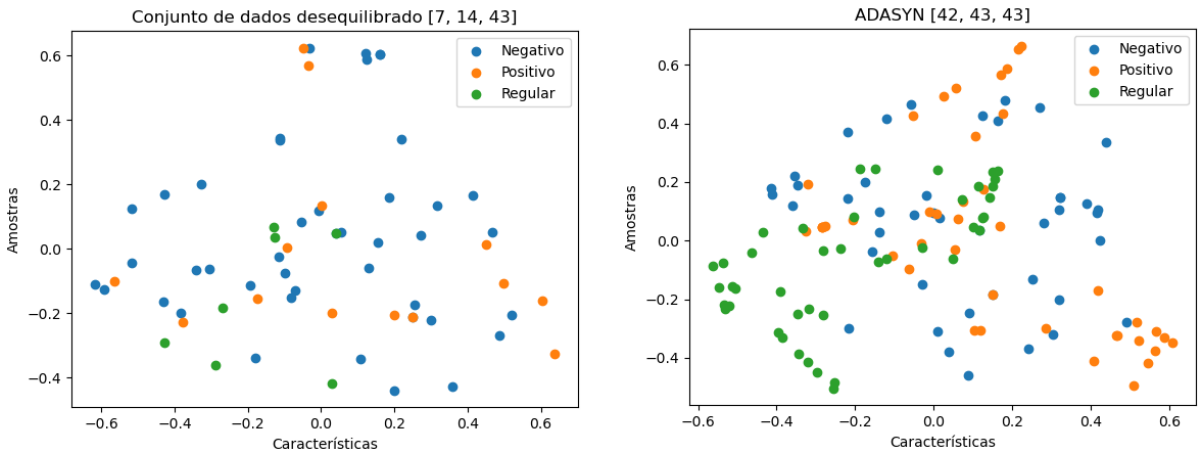
Figura 22 – Exemplo desequilíbrio de classe



Fonte: (GUK, 2020)

Ainda se tratando de dados desbalanceados, o algoritmo denominado *ADASYN* (Adaptive Synthetic) é comumente utilizado para resolver tal problema. Este algoritmo gera dados sintéticos buscando balancear o conjunto de dados. A Figura 23 ilustra dados desbalanceados na esquerda e na direita os dados balanceados pelo método *ADASYN*.

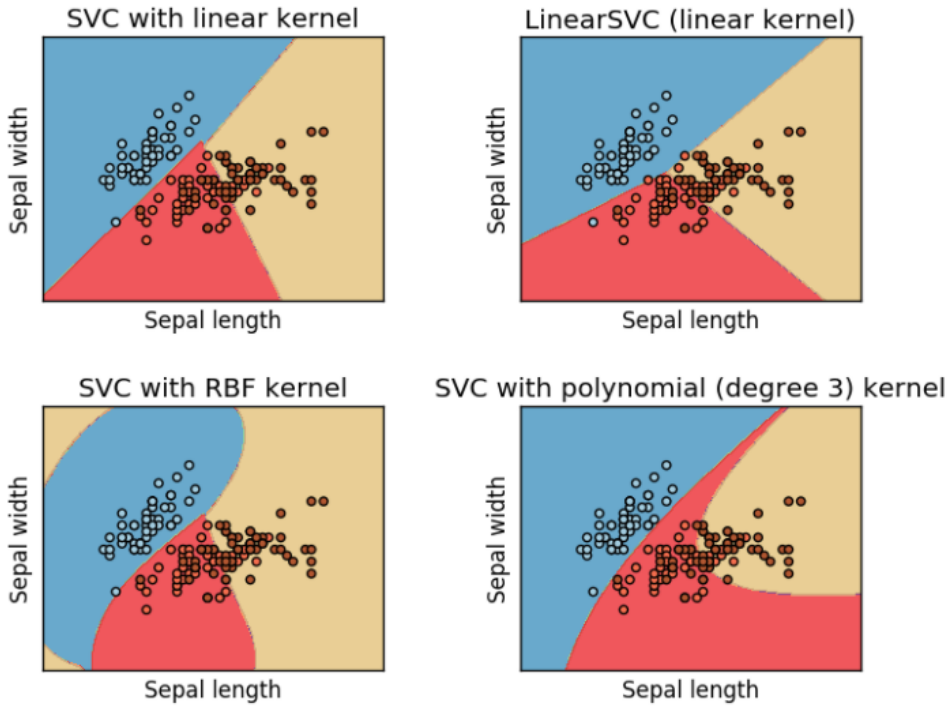
Figura 23 – Exemplo desequilíbrio de classe e respectivo balanceamento



Fonte: (BELLINI, 2020)

As visualizações ainda podem auxiliar na identificação dos melhores parâmetros para um algoritmo. A título de exemplificação será usado o algoritmo denominado *SVC* (Support Vectors Classifier). O *SVC* procura encontrar o melhor hiperplano para separar as diferentes classes do conjunto de dados, maximizando a distância entre os pontos de amostra e o hiperplano (FRAJ, 2018). O parâmetro que dirige o algoritmo *SVC* é chamado de *kernel*. A Figura 24 ilustra diferentes parâmetros utilizados para o valor do *kernel*, e para cada valor seleciona um tipo de hiperplano que é então usado para separar os dados.

Figura 24 – Visualização para auxiliar na identificação dos melhores parâmetros

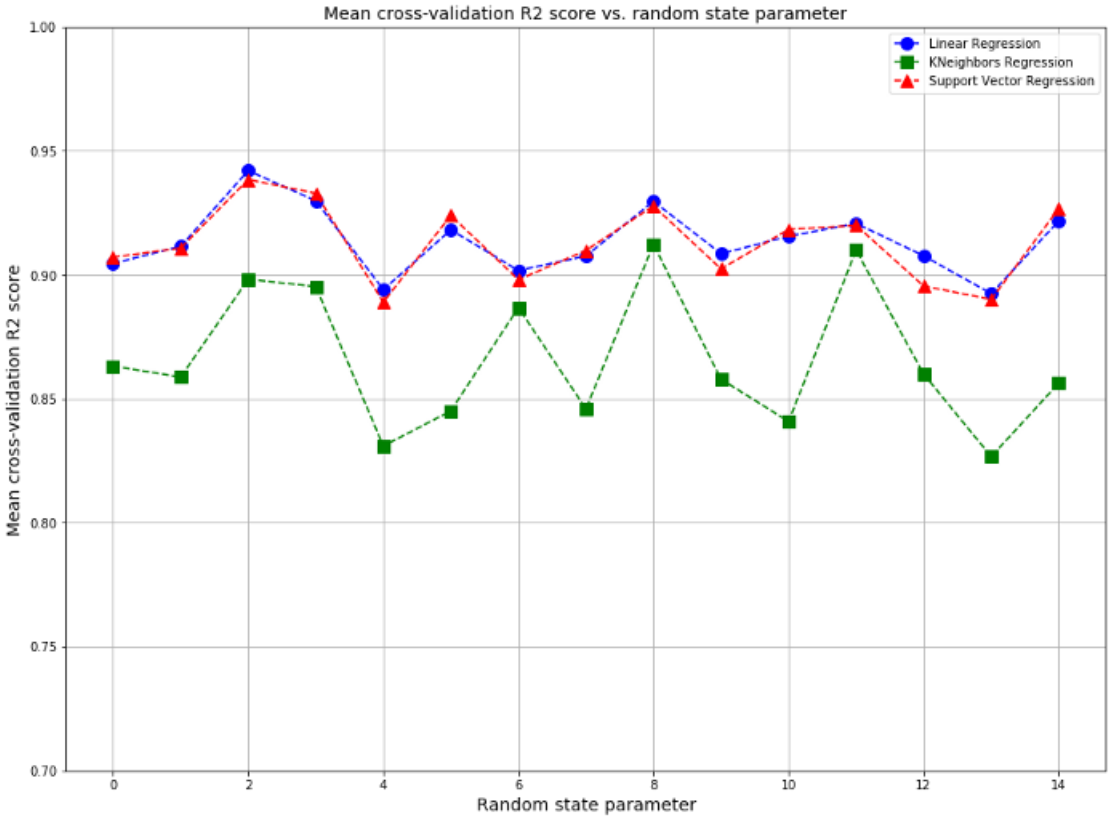


Fonte: (BELLINI, 2020)

Além do exemplo anterior, o uso de gráficos de linhas é muito comum. Para ilustrar, considere o método de amostragem denominado de validação cruzada. Ele permite a análise de múltiplos modelos de aprendizado simultaneamente para fins de comparação entre o desempenho de um classificador. Neste caso, a visualização de dados pode ser utilizada, como ilustra a Figura 25. Nela tem-se um cenário em que cada ponto representa o valor obtido pelas execuções variando-se um parâmetro representado no eixo x (random state), assumindo o desempenho de três técnicas de regressão distintas (linear, *kneigbors* e *support vector*). Este gráfico foi gerado por meio do pacote denominado *matplotlib*.

Além dos exemplos precedentes, existem inúmeras outras formas de visualização. A Figura 26 apresenta uma visão geral sobre as visualizações disponíveis na biblioteca *YellowBrick*. Como pode-se perceber, todas as formas podem ser complementares em um processo de análise. Cada uma delas destaca alguma característica do conjunto de dados, ou do classificador obtido a partir dos dados.

Figura 25 – Exemplo validação cruzada



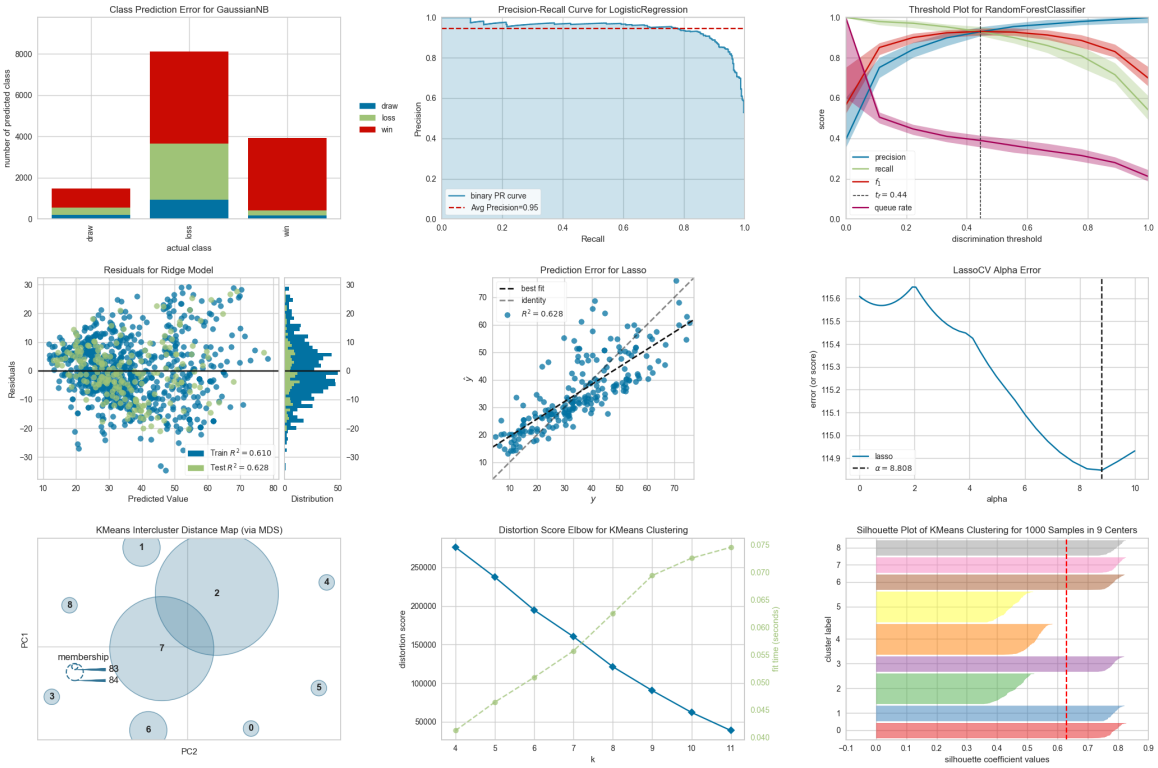
Fonte: (TAYO, 2020)

2.7 CONSIDERAÇÕES FINAIS

Durante o processo de desenvolvimento de um modelo de *Machine Learning* existe o desafio de interpretar os dados que estão disponíveis e os resultados que são gerados durante cada etapa do processo, seja na exploração dos dados, durante a seleção de atributos ou na própria avaliação do modelo. Essa questão se torna ainda mais proeminente quando surge a necessidade de apresentar descobertas provisórias a um grupo de partes interessadas, clientes ou outros indivíduos com interesse pessoal. Então, para lidar com grandes matrizes de números, notações científicas e fórmulas que possuem uma história dentro do conjunto de dados, a visualização se torna uma ferramenta essencial.

Percebe-se pelos estudos realizados que os componentes de visualização de dados no processo de *Machine Learning* desempenham um papel importante e podem ser usados nas diferentes etapas do processo. Podendo ajudar a avaliar o desempenho, estabilidade e o valor preditivo do modelo criado, auxiliando também no diagnóstico de problemas, fornecendo meios de visualização compreensíveis e interpretáveis durante todo o processo de *Machine Learning*.

Figura 26 – Visão geral de visualizações disponíveis na biblioteca Yellowbrick



Fonte: (DEVELOPERS, 2020)

3 VISUALIZAÇÕES DE DADOS: UM ESTUDO DE CASO

A visualização de dados é um recurso normalmente empregado ao final do processo de *Machine Learning*. Contudo, novas ferramentas vem sendo desenvolvidas para contribuir nas diversas etapas do processo de *Machine Learning*. Para demonstrar o seu uso, este trabalho apresenta um estudo de caso de visualização de dados, inserida em um processo de *Machine Learning*. Este capítulo descreve o desenvolvimento realizado. Aplicou-se o mesmo percurso metodológico para dois diferentes conjunto de dados com o objetivo de mostrar a importância das visualizações de dados nas diferentes etapas do processo de *Machine Learning*. Os detalhamentos são feitos nas seções seguintes.

3.1 PERCURSO METODOLÓGICO

O trabalho foi desenvolvido a partir das seguintes etapas:

1. Seleção de ferramentas de *software*
2. Seleção dos conjuntos de dados
3. Exploração dos dados e aplicação de visualizações
4. Pré-processamento dos dados e aplicação de visualizações
5. Aplicação de algoritmos de *Machine Learning* e aplicação de visualizações
6. Definição de critérios de avaliação para o modelo e aplicação de visualizações
7. Seleção do modelo de melhor desempenho e visualizações para justificar a escolha

As seções seguintes descrevem o desenvolvimento e resultados obtidos em cada etapa.

3.2 SELEÇÃO DE FERRAMENTAS DE *SOFTWARE*

Dentre as linguagens de programação disponíveis atualmente existem algumas que são mais apropriadas para tarefas de *Machine Learning* do que outras. Por exemplo, a linguagem de programação denominada R[1] possui código aberto e é muito utilizada nos meios de computação estatística. Outro exemplo é a linguagem de programação denominada Julia[2], que foi desenvolvida especialmente para atender as necessidades das áreas da ciência da computação e da computação estatística, e por fim, *Python* que atualmente é a linguagem de programação mais

[1] https://www.r-project.org/about.html
[2] https://julialang.org

utilizada para *Machine Learning*, sendo a linguagem escolhida por grandes empresas do mundo de tecnologia, como o Google [3], Instagram[4], Facebook[5], Netflix[6] e Amazon[7] (MÜLLER, 2017).

Python possui uma coleção extensa de bibliotecas e pacotes para auxiliar no desenvolvimento de um modelo de *Machine Learning*, inclusive para a visualização de dados, por esse motivo foi a linguagem de programação escolhida para desenvolvimento deste trabalho.

Como plataforma de desenvolvimento foi utilizado o Google Colaboratory [8]. Comumente chamado de Google Colab, essa plataforma é basicamente um serviço de nuvem gratuito, hospedado pela própria Google com o intuito de incentivar a pesquisa de *Machine Learning* e Inteligência Artificial. O Google Colab permite o desenvolvimento de código em Python no próprio navegador, facilitando a configuração e o compartilhamento de código.

Durante o processo de *Machine Learning* foram utilizados pacotes para auxiliar seu desenvolvimento. Dentre os pacotes que foram utilizados na implementação, está incluso o pacote denominado *Pandas*, utilizado para facilitar a importação e análise dos dados. Ele é útil para organizar o conjunto de dados de forma tabular e é possível realizar algumas operações e manipulações nos dados. Em particular, este pacote oferece estruturas de dados e algumas operações para manipular tabelas numéricas e séries temporais.

Além do *Pandas*, o pacote denominado *scikit-learn* também foi utilizado. Este pacote é uma das bibliotecas de *Machine Learning* mais populares para *Python*. A biblioteca *scikit-learn* é de código aberto e reutilizável em vários contextos, incentivando o uso acadêmico e comercial. Ela fornece uma variedade de algoritmos de aprendizagem supervisionada e não supervisionada em *Python*. Para implementar o *scikit-learn*, é necessário importar alguns pacotes, como o *NumPy*[9] que adiciona suporte para matrizes e vetores multidimensionais grandes, junto com uma extensa coleção de funções matemáticas de alto nível para operar esses vetores. Além do *NumPy*, para utilizar o *scikit-learn* também será necessário o *SciPy*[10] que também é uma biblioteca de código aberto usada para resolver problemas científicos e matemáticos. O *SciPy* é construído com base no pacote *NumPy* e permite a manipulação e a criação de visualizações dos dados com uma ampla gama de comandos de alto nível.

Como demonstrado neste trabalho, as técnicas de visualização provam ser uma ferramenta útil durante o processo de *Machine Learning*. Por isso, além dos pacotes para auxiliar na manipulação dos dados e na implementação dos algoritmos de *Machine Learning*, foram utilizados pacotes para auxiliar na criação dos diversos tipos de visualização de dados, com o intuito de agregar valor nas diferentes etapas do processo de *Machine Learning*. Entre os pacotes

[3] https://www.google.com
[4] instagram.com
[5] facebook.com
[6] netflix.com
[7] amazon.com.br
[8] https://colab.research.google.com/notebooks/intro.ipynb
[9] https://numpy.org
[10] https://www.scipy.org

está o *matplotlib*, que é a biblioteca de *Python* mais popular para visualização e exploração de dados. O *matplotlib* faz parte do ecossistema do *SciPy* e possibilita a criação de todos os tipos de gráficos e visualizações em 2D e será especialmente útil na exploração dos dados.

Os pacotes denominados *Plotly*[11] e *Seaborn*[12] também foram utilizados para auxiliar na criação de visualizações. Estes pacotes possuem inúmeros métodos disponíveis, como alguns dos exemplos que foram utilizados neste trabalho encontram-se os gráficos interativos e os mapas coropléticos.

Por último, outro pacote que será utilizado é o pacote denominado *Yellowbrick*, que oferece uma ampla variedade de visualizações que podem ser utilizadas em cada etapa do processo de criação de um modelo de *Machine Learning*. Variando de análise e seleção de atributos à seleção e otimização de modelo, tornando mais fácil decidir quais atributos manter no modelo, qual modelo tem melhor desempenho e como ajustar os parâmetros de um modelo para atingir o desempenho ideal para uso futuro. Além disso, as diferentes possíveis visualizações dos resultados finais do modelo facilitam a apresentação para o público, simplificando os resultados do modelo.

3.3 SELEÇÃO DOS CONJUNTOS DE DADOS

Foram escolhidos dois conjuntos de dados para se aplicar o processo de *Machine Learning* apoiado em visualizações. Utilizou-se como fonte de pesquisas repositórios de dados públicos. Essa seção apresenta brevemente os conjuntos de dados selecionados e suas características.

Devido a situação pandêmica que o mundo se encontra o primeiro conjunto de dados escolhido é gerenciado pelo Centro de Ciência e Engenharia de Sistemas da Universidade *Johns Hopkins*. Este conjunto de dados possui dados globais relacionados ao Coronavírus desde 22 de janeiro até os dias atuais, sendo atualizado diariamente (ENSHENG; HONGRU; LAUREN, 2020), os dados são cumulativos, portanto representam uma série temporal. Todos os dados coletados e exibidos são disponibilizados gratuitamente em um repositório no *GitHub*[13]. Dentre alguns dos atributos presentes neste conjunto de dados se encontram o nome do país, o número de recuperados, curados, mortes e de casos ativos. No site *Kaggle*[14], que representa uma comunidade online de cientistas de dados e profissionais de *Machine Learning*, foi desenvolvido um arquivo compilado que será usado além das séries temporais disponíveis no *GitHub*.

O segundo conjunto de dados escolhido é bem reconhecido na área de *Machine Learning* denominado Iris. Esse conjunto de dados foi apresentado na seção 3.5 deste trabalho, como exemplo ilustrativo das visualizações mais comuns aplicadas na exploração de dados. O conjunto de dados foi desenvolvido por FISHER (1936) e possui 4 atributos, sendo eles: o comprimento da

[11] https://plotly.com/python/
[12] https://seaborn.pydata.org
[13] https://github.com
[14] kaggle.com

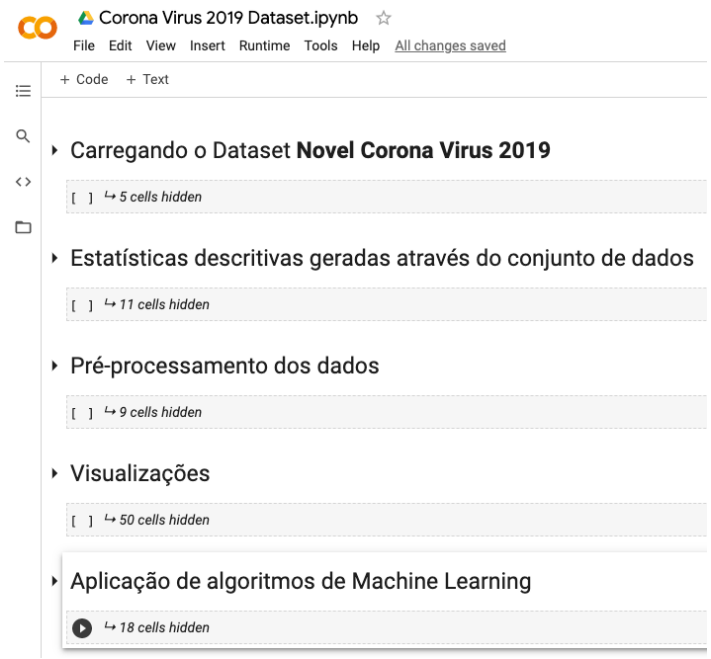
sépala e da pétala, e a largura da sépala e da pétala. Como atributo classificador, tem-se o tipo da flor representado por três classes: iris setosa, iris versicolor e iris virgínica.

3.4 PRIMEIRO ESTUDO DE CASO: CONJUNTO DE DADOS SOBRE CORONAVÍRUS

Este conjunto de dados contém dados sobre o novo Coronavírus chamado de COVID-19 que literalmente parou o mundo no ano de 2020. Gerenciado pelo Centro de Ciência e Engenharia de Sistemas da Universidade Johns Hopkins contém dados de todos os países do mundo, desde 22 de janeiro até os dias atuais. Dentre as 8 colunas, cada linha possui dados como o número de série, a data de observação, a província ou estado se presente, o país de observação, quando foi a última atualização, o número cumulativo de casos confirmados, recuperados e mortes para cada data.

Como foi mencionado no Capítulo 3.2 para o desenvolvimento dos estudos de caso foi utilizado o Google Colab. O ambiente do Google Colab é interativo, e é composto pelo *Jupyter* Notebook[15] no qual, devido a sua linguagem de marcação é possível organizar os trechos de código de acordo com as etapas que serão realizadas como é ilustrado na Figura 27, além de claro desenvolver na linguagem de programação *Python*, possibilitando então a importação de bibliotecas e pacotes para auxiliar em cálculos ou em geração de gráficos.

Figura 27 – *Google Colab* do conjunto de dados sobre o COVID-19



Fonte: O Autor (2021)

[15] https://ipython.org/notebook.html

3.4.1 Exploração dos dados e aplicação de visualizações

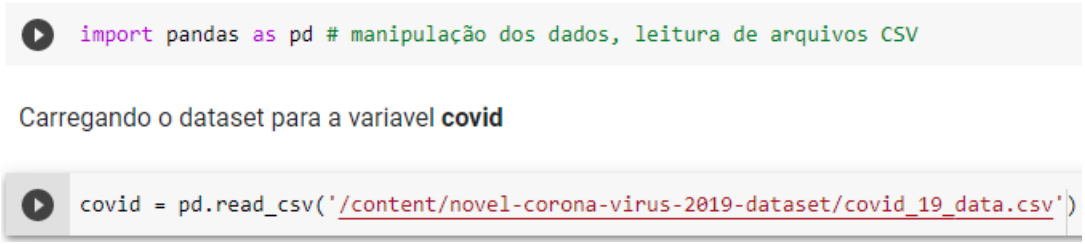
Primeiramente foi feito o *download* do conjunto de dados relacionado ao COVID-19 no site do *Kaggle*[16]. Este arquivo contém 3 séries temporais contendo os casos confirmados, casos recuperados e as morte desde o dia 22 de Janeiro de 2020 até os dias atuais. Além de conter um arquivo contendo todas essas informações juntas, das datas do dia 22 de Janeiro de 2020 até o dia 27 de Fevereiro de 2021, todos os arquivos estão em no formato *CSV*. Posteriormente, como o Google Colab é uma ferramenta hospedada na nuvem, foi realizado o *upload* destes arquivos para o ambiente do Google Colab, como ilustra a Figura 28.

Figura 28 – Upload do conjunto de dados relacionado ao COVID-19 para o *Google Colab*

Fonte: O Autor (2021)

O carregamento do conjunto de dados foi realizado através da biblioteca denominada *Pandas*, que é responsável por abstrair a leitura do arquivo *CSV* e também de auxiliar na manipulação deste arquivo ao decorrer do processo. Primeiramente foi importada a biblioteca *Pandas* e atribui-se um apelido para ela, denominado *pd*. Posteriormente, utilizando um comando chamado *read_csv*, o conjunto de dados foi carregado para uma variável denominada *covid*, como ilustra a Figura 29.

Figura 29 – Carregando o conjunto de dados relacionado ao COVID-19



Fonte: O Autor (2021)

Com o conjunto de dados carregado e atribuído a uma variável é possível utilizar o comando *head()* proveniente da biblioteca *Pandas*, que irá imprimir os cinco primeiros valores presentes no conjunto de dados como demonstra a Figura 30.

[16] https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset

Figura 30 – Os 5 primeiros valores do conjunto de dados relacionado ao COVID-19

	SNo	ObservationDate	Province/State	Country/Region	Last Update	Confirmed	Deaths	Recovered
0	1	2020-01-22	Anhui	Mainland China	2020-01-22 17:00:00	1.0	0.0	0.0
1	2	2020-01-22	Beijing	Mainland China	2020-01-22 17:00:00	14.0	0.0	0.0
2	3	2020-01-22	Chongqing	Mainland China	2020-01-22 17:00:00	6.0	0.0	0.0
3	4	2020-01-22	Fujian	Mainland China	2020-01-22 17:00:00	1.0	0.0	0.0
5	6	2020-01-22	Guangdong	Mainland China	2020-01-22 17:00:00	26.0	0.0	0.0

Fonte: O Autor (2021)

De início é possível visualizar que as colunas estão originalmente em inglês, então para realizar a tradução para o português utilizou-se o método *rename*, também proveniente da biblioteca *Pandas*, na variável *covid*, que contém o conjunto de dados, sendo utilizado como parâmetros os nomes atuais das colunas e os respectivos nomes desejados.

Figura 31 – Os 5 primeiros valores do conjunto de dados atualizados

```
covid.rename(columns={'ObservationDate': 'Data',
                      'Province/State':'Província/Estado',
                      'Country/Region':'País/Região',
                      'Confirmed':'Confirmados',
                      'Deaths':'Mortes',
                      'Recovered':'Recuperados',
                      'Last Update':'Última Atualização',
                     }, inplace=True)

covid.head()
```

	SNo	Data	Província/Estado	País/Região	Última Atualização	Confirmados	Mortes	Recuperados
0	1	01/22/2020	Anhui	Mainland China	1/22/2020 17:00	1.0	0.0	0.0
1	2	01/22/2020	Beijing	Mainland China	1/22/2020 17:00	14.0	0.0	0.0
2	3	01/22/2020	Chongqing	Mainland China	1/22/2020 17:00	6.0	0.0	0.0
3	4	01/22/2020	Fujian	Mainland China	1/22/2020 17:00	1.0	0.0	0.0
4	5	01/22/2020	Gansu	Mainland China	1/22/2020 17:00	0.0	0.0	0.0

Fonte: O Autor (2021)

A biblioteca *Pandas* oferece métodos capazes de gerarem uma descrição geral do conjunto de dados carregado. Como por exemplo, o método *shape*, este método mostra o número de linhas e de colunas presentes no conjunto de dados, também é possível analisar a quantidade de valores *nullos* ou vazios presentes nas linhas do conjunto de dados, para isso utiliza-se o método *isnull()* que verifica para cada coluna de todas as linhas se possui um valor vazio, seguido de *.sum()* que retorna a soma dos valores. Também é possível verificar quais os tipos de dados sendo utilizados em cada coluna ao utilizar-se o comando *dtypes*. A Figura 32 demonstra o resultado dos comandos citados anteriormente, aplicado a variável *covid*.

Figura 32 – Descrição geral do conjunto de dados relacionado ao COVID-19

```
print("Número de linhas/Número de colunas do dataset:", covid.shape)
print("\n")
print("Procurando por valores nullos:")
print(covid.isnull().sum())
print("\n")
print("Verificando os tipos de dados de cada coluna:")
print(covid.dtypes)
```

```
Número de linhas/Número de colunas do dataset: (236017, 8)


Procurando por valores nullos:
SNo                      0
Data                     0
Província/Estado     62045
País/Região              0
Última Atualização       0
Confirmados              0
Mortes                   0
Recuperados              0
dtype: int64


Verificando os tipos de dados de cada coluna:
SNo                    int64
Data                  object
Província/Estado      object
País/Região           object
Última Atualização    object
Confirmados          float64
Mortes               float64
Recuperados          float64
dtype: object
```

Fonte: O Autor (2021)

Através da Figura 32 é possível visualizar que este conjunto de dados possui 236017 linhas e 8 colunas. Além disso, é possível analisar que há uma grande quantidade de valores *nullos* na coluna Província/Estado e por fim é possível identificar quais os tipos de dados de cada coluna.

3.4.2 Pré-processamento dos dados e aplicação de visualizações

Com base na descrição geral do conjunto de dados é possível identificar dois problemas, o primeiro é a presença de valores *nullos*, o segundo problema se refere ás colunas *Data* e *Última Atualização* estarem sendo representadas como um tipo dinâmico de objeto, quando na verdade representam uma data.

Primeiramente serão removidas as colunas *SNo* e *Última Atualização* que não serão utilizadas ao decorrer do processo. Para realizar a remoção das colunas foi utilizado o método denominado *drop*, passando como parâmetro as colunas a serem removidas. A Figura 33 ilustra o método sendo aplicado na variável *covid*.

Figura 33 – Método *drop* sendo utilizado

```
covid.drop(["SNo"], 1, inplace=True)
covid.drop(["Última Atualização"], 1, inplace=True)
```

Fonte: O Autor (2021)

O primeiro problema se refere aos valores *nullos* presentes na coluna Província/Estado. Para solucionar este problema, foi atribuído *strings* vazias para as linhas que possuem valores *nullos*, utilizando o método da biblioteca *Pandas*, denominado por *fillna()*, com o objetivo de evitar algum erro de compilação ao manipular os dados desta coluna. A Figura 34 ilustra o resultado ao utilizar novamente os método *isnull().sum()*.

Figura 34 – Método *fillna()* sendo aplicado

```
covid[['Província/Estado']] = covid[['Província/Estado']].fillna('')
print(covid.isnull().sum())
```

```
Data              0
Província/Estado  0
País/Região       0
Confirmados       0
Mortes            0
Recuperados       0
dtype: int64
```

Fonte: O Autor (2021)

O segundo problema presente no conjunto de dados está presente na representatividade da coluna denominada *Data*, essa coluna representa uma data, porém está sendo representada como um objeto do tipo dinâmico. Para realizar a conversão da coluna para um tipo de dado que represente uma data foi utilizado o método *to_datetime*. Este método realiza a transformação para um objeto que representa uma data. A Figura 35 ilustra o método sendo utilizado e demonstra novamente os tipos de dados presentes no conjunto de dados atualizado.

Figura 35 – Método *to_datetime()* sendo aplicado

```
covid['Data'] = pd.to_datetime(covid['Data'], format='%m/%d/%Y')
print(covid.dtypes)
```

```
Data              datetime64[ns]
Província/Estado          object
País/Região               object
Confirmados              float64
Mortes                   float64
Recuperados              float64
dtype: object
```

Fonte: O Autor (2021)

Com o conjunto de dados devidamente pré-processado é possível desenvolver visualizações a fim de explorar o conjunto de dados. Este conjunto de dados possui dados temporais,

portanto procurou-se identificar as estruturas temporais presentes, como tendências, ciclos e sazonalidade que influenciam na escolha do modelo a ser utilizado posteriormente.

Antes de tudo foi analisado o período de datas presente no conjunto de dados. Para isso, na coluna *Data* foi utilizado os métodos *min()* e *max()* que respectivamente retornam o menor e o maior valor presente na coluna, realizado posteriormente uma diferença entre as datas para obter o número total de dias presente no conjunto de dados como a Figura 36 demonstra.

Figura 36

```
first_record = covid['Data'].min().strftime("%d %b %Y")
last_record = covid['Data'].max().strftime("%d %b %Y")
total_days_count = (covid['Data'].max() - covid['Data'].min())

print(f"Primeiro registro em:    {first_record}")
print(f"Último registro em:      {last_record}")
print(f"Total de dias até então: {total_days_count}")
```

```
Primeiro registro em:    22 Jan 2020
Último registro em:      27 Feb 2021
Total de dias até então: 402 days 00:00:00
```

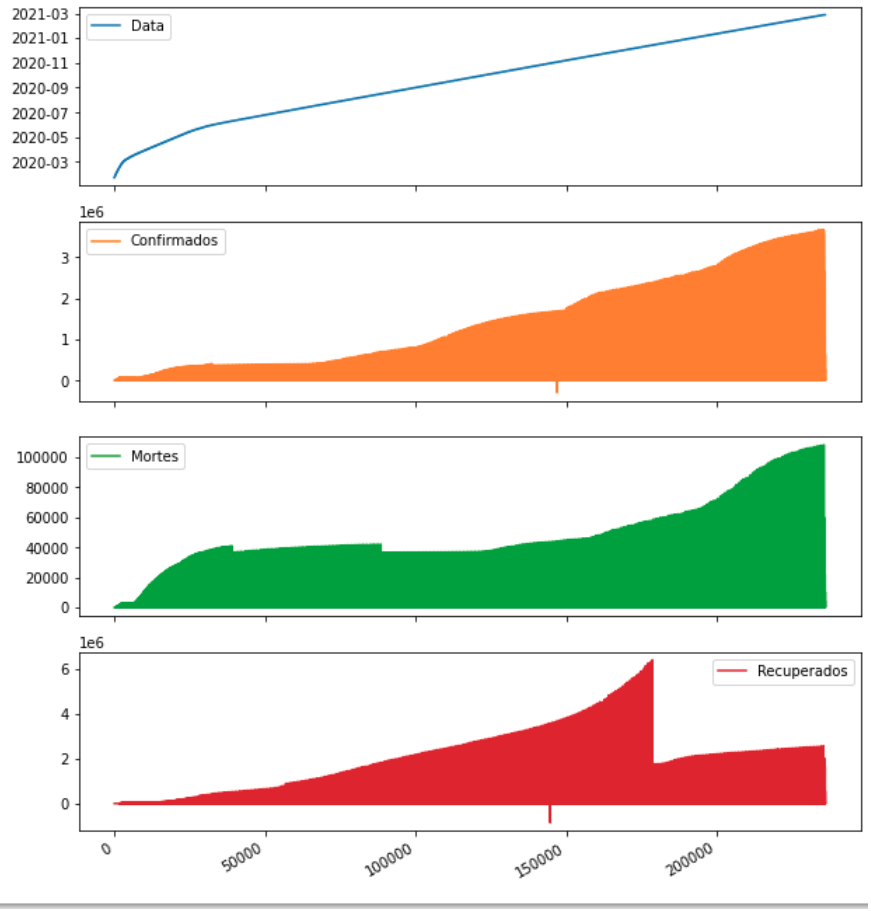
Fonte: O Autor (2021)

Posteriormente foi gerado uma visualização através do método *plot()*, proveniente da biblioteca *Pandas*, que para desenhar este gráfico utiliza a biblioteca *matplotlib*. O resultado do método é ilustrado pela Figura 37, demonstrando o crescimento no número de casos confirmados, recuperados e de mortes ao decorrer do tempo. Além disso, é possível analisar que há uma queda brusca no número de recuperados, onde a curva deveria seguir numa crescente, mostrando um possível problema no conjunto de dados.

Figura 37 – Método *plot()* aplicado a variável *covid*

```
covid.set_index('Data')
covid.plot(subplots=True, figsize=(10,12))
```

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f1f9b7233d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f1f9b571f90>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f1f9b534390>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f1f9b4ec750>],
      dtype=object)
```



Fonte: O Autor (2021)

Similar ao gráfico demonstrado pela Figura 37, é possível demonstrar as mesmas linhas geradas anteriormente de uma maneira sobreposta, facilitando a comparação entre as áreas. Para obter esse resultado foram importadas as bibliotecas *pyplot* proveniente da biblioteca *matplotlib* e a biblioteca *Seaborn* que será utilizada posteriormente. A Figura 38 ilustra a importação no *Google Colab*, onde a área verde representa os casos recuperados, a área azul os casos confirmados e a área vermelha representa as mortes confirmadas.

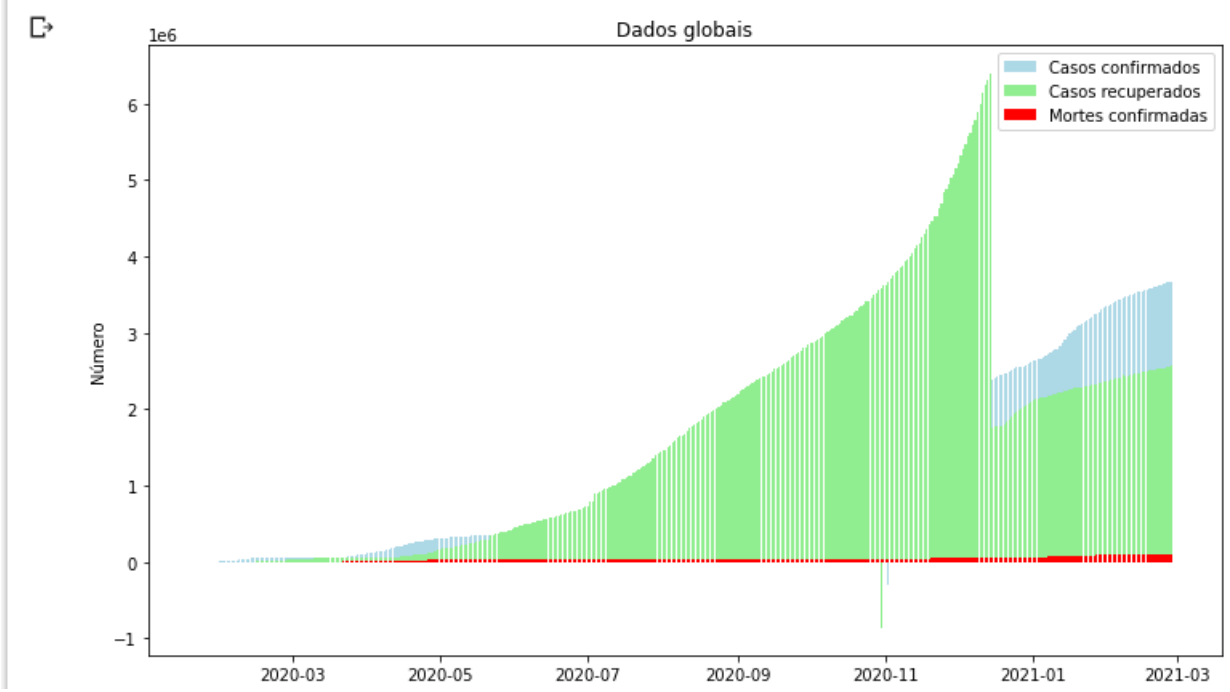
Figura 38 – Importação de bibliotecas de visualização

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Fonte: O Autor (2021)

Figura 39 – Método *plot()* utilizando a biblioteca *pyplot*

```
dates = covid.Data
confirmed = list(covid.Confirmados)
deaths = list(covid.Mortes)
recovered= list(covid.Recuperados)
plt.figure(figsize=(12,7))
plt.bar(dates, confirmed, color='lightblue', label="Casos confirmados")
plt.bar(dates, recovered, color='lightgreen',label="Casos recuperados")
plt.bar(dates, deaths, color='red',label="Mortes confirmadas")
plt.xlabel("Datas")
plt.ylabel("Número")
plt.title("Dados globais")
plt.legend()
plt.show()
```



Fonte: O Autor (2021)

A fim de compreender melhor a queda brusca apresentada nas figuras 37 e 39 foi gerado uma tabela dos 15 países na última data disponível, que no caso é o dia 27 de Fevereiro de 2021. A tabela está ordenada pelo maior número de casos confirmados, contendo também os casos recuperados, ativos e as mortes dos respectivos países. A Figura 40 mostra a tabela que foi gerada, podendo ser analisado algumas discrepâncias nos números de recuperados para países como por exemplo os EUA (Estados Unidos da América) que possui 0 pacientes recuperados e países como o Reino Unido, França e a Espanha que nessa data específica possuem uma diferença muito grande no número de casos ativos e recuperados.

Figura 40 – Os 15 países com mais casos confirmados na data de 27/02/2021

Última data disponível: 2021-02-27 00:00:00

Tabela dos 15 países com mais casos confirmados.

País/Região	Confirmados	Mortes	Recuperados	Ativos
US	28554465	511994	0	28042471
India	11096731	157051	10775169	164511
Brazil	10517232	254221	9371448	891563
Russia	4187166	84330	3756808	346028
UK	4182772	122939	11602	4048231
France	3747263	85741	261649	3399873
Spain	3188553	69142	150376	2969035
Italy	2907825	97507	2398352	411966
Turkey	2693164	28503	2565723	98938
Germany	2444177	70092	2252970	121115
Colombia	2248135	59660	2145450	43025
Argentina	2104197	51946	1899087	153164
Mexico	2084128	185257	1630002	268869
Poland	1696885	43656	1414461	238768
Iran	1623159	59980	1386534	176645

Fonte: O Autor (2021)

A razão por trás da data do dia 27 de Fevereiro de 2021 ter sido escolhida é por conta dos dados deste conjunto de dados serem cumulativos. Portanto, foi atribuída a variável *latest_register* apenas as linhas que tem a data igual ao dia 27/02/2021, que é a data do último registro disponível no conjunto de dados. Após selecionar os registros da última data disponível foi aplicado o método *groupby*, utilizando como parâmetro a coluna *País/Região*, este método junta os dados que pertencerem ao mesmo País e ou Região, logo em seguida foi utilizado o método *sum()*, este método realiza a soma e junta os valores para todas as colunas, esses valores foram atribuídos a variável denominada *countries*. Após agrupar os valores de cada país, foi criado uma nova coluna denominada *Ativos*, que representa os casos ativos de cada país, este valor está sendo calculado com base na diferença entre os casos confirmados menos os casos recuperados e as mortes. A Figura 41 ilustra o código desenvolvido.

Figura 41 – Código para gerar a tabela da Figura 40

```
from datetime import timedelta # bibliotecas para manusear datas
last_date = covid.Data.max()
print('Última data disponível: ' + str(last_date) + '\n')

latest_register = covid[covid.Data == last_date]

countries = latest_register.groupby('País/Região').sum()
countries['Ativos'] = countries['Confirmados'] - countries['Mortes'] - countries['Recuperados']

print('Tabela dos 15 países com mais casos confirmados.\n')
_ = countries.sort_values('Confirmados', ascending=False).head(15)
_.astype('int64').style.background_gradient(cmap='Reds')
```

Fonte: O Autor (2021)

Com base na tabela gerada e ilustrada pela Figura 40 nota-se que os casos recuperados dos EUA estão zerados, portanto utilizou esse país como base para procurar uma data anterior que possua o número de recuperados atualizado. A Figura 42 ilustra o trecho de código utilizado para encontrar a data mais próxima em que os dados de pacientes recuperados esteja atualizado para o país dos EUA e a data encontrada que foi o dia 14/12/2020.

Figura 42 – Código para buscar uma data em que o EUA possua o número de recuperados atualizado

```
max_date = covid.Data.max()
i = 0

while True:
  last_date = max_date - timedelta(i)
  i = i + 1
  latest_register = covid[covid.Data == last_date]
  countries = latest_register.groupby('País/Região').sum()
  countries.reset_index(inplace=True)
  us = countries[countries['País/Região'] == 'US']
  us_recovered = us['Recuperados']
  if((us_recovered > 0.0).bool()):
    break

print('Data: ' + str(last_date))
```

```
Data: 2020-12-14 00:00:00
```

Fonte: O Autor (2021)

Após encontrar uma data em que os EUA possuam casos recuperados foi desenvolvido a mesma tabela da Figura 40, utilizando os registros disponíveis agora para a data de 14/12/2020. Com esta nova data os dados mostram que os EUA possuem casos recuperados, porém os países como França, Reino Unido e Espanha continuam com discrepâncias nos casos ativos, ilustrado pela Figura 43.

Figura 43 – Os 15 países com mais casos confirmados na data de 14/12/2020

Tabela dos 15 países com mais casos confirmados.

País/Região	Confirmados	Mortes	Recuperados	Ativos
US	16601499	305960	6399531	9896008
India	9906165	143709	9422636	339820
Brazil	6927145	181835	6158049	587261
Russia	2656601	46846	2105414	504341
France	2435751	58391	183340	2194020
UK	1874870	64500	4051	1806319
Turkey	1866345	16646	1631944	217755
Italy	1855737	65011	1115617	675109
Spain	1751884	48013	150376	1553495
Argentina	1503222	41041	1340120	122061
Colombia	1434516	39195	1321469	73852
Germany	1357261	22634	1012077	322550
Mexico	1255974	114298	927754	213922
Poland	1140572	22960	869155	248457
Iran	1115770	52447	823231	240092

Fonte: O Autor (2021)

É possível visualizar os dados da tabela da Figura 43 de outras maneiras, como por exemplo através de gráficos de barras. Para gerar os gráficos de barras foi desenvolvido uma função visando diminuir a repetição de código. A Figura 44 mostra o trecho de código desenvolvido, onde os parâmetros necessários são a coluna desejada e título do gráfico, a coluna passada como parâmetro será ordenada do maior valor para o menor e apenas os 15 maiores valores serão mostrados. Para gerar o gráfico de barra foi utilizado a biblioteca denominada *Seaborn* que foi importada anteriormente.

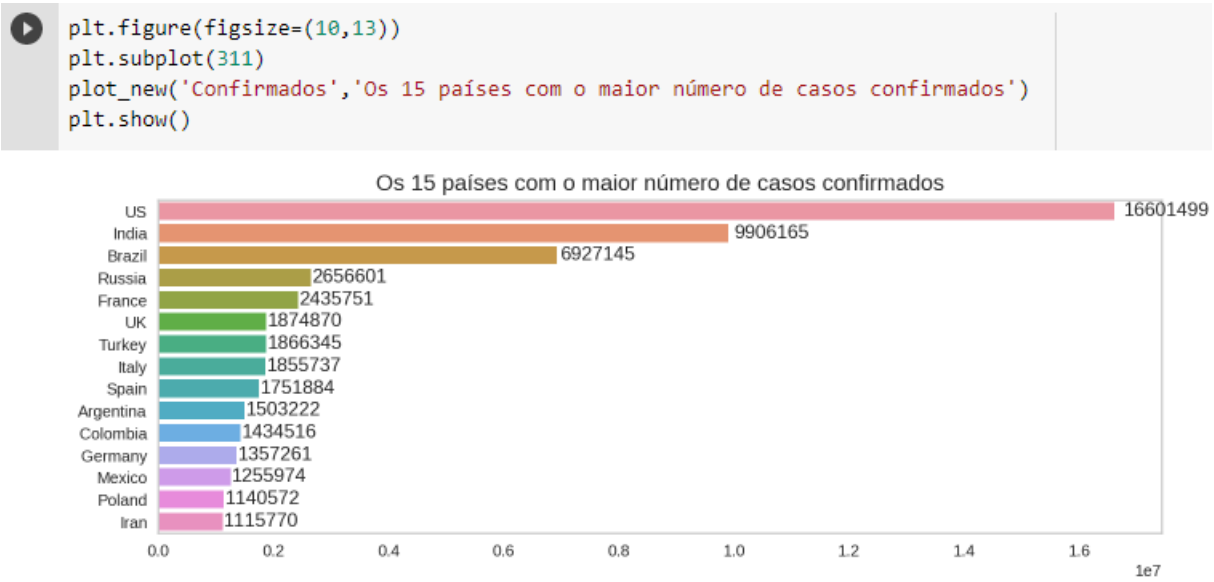
Figura 44 – Função desenvolvida para criar gráficos de barras

```
def plot_new(column, title):
    _ = countries.sort_values(column, ascending=False).head(15)
    g = sns.barplot(_[column], _.index)
    plt.title(title, fontsize=14)
    plt.ylabel(None)
    plt.xlabel(None)
    plt.grid(axis='x')
    for i, v in enumerate(_[column]):
            g.text(v*1.01, i+0.2, str(int(v)))
```

Fonte: O Autor (2021)

A Figura 45 demonstra o gráfico gerado pela função ilustrada na Figura 44, para gerar este gráfico de barra foi utilizado a coluna *Confirmados* e o título *Os 15 países com o maior número de casos confirmados* como parâmetros. Nota-se que na data de 14/12/2020 os EUA é o país com mais casos confirmados no mundo, seguido da Índia e pelo Brasil, além disso é possível analisar que há uma grande diferença no número de casos confirmados quando se compara os três primeiros países com os demais.

Figura 45 – Gráfico de barras gerado para a coluna *Confirmados*

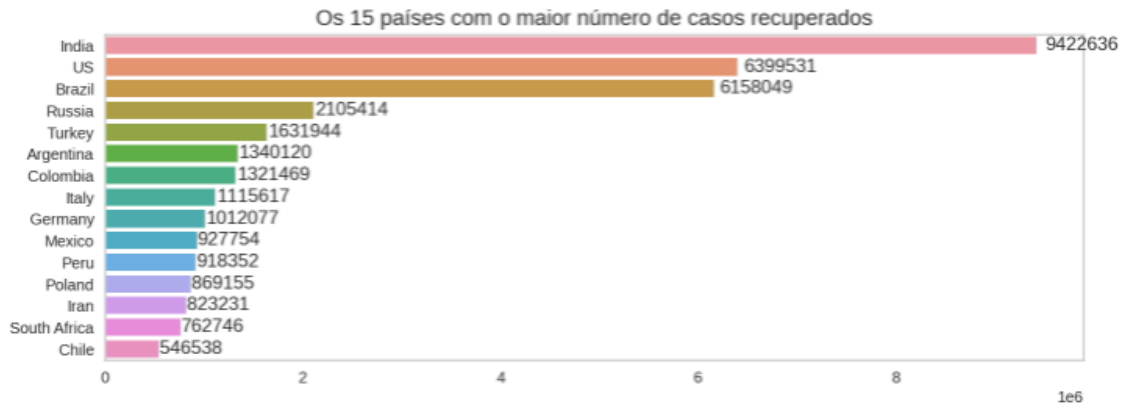


Fonte: O Autor (2021)

Para gerar o gráfico ilustrado pela Figura 46 foi utilizado a mesma função que gerou o gráfico da Figura 45, utilizando como parâmetros a coluna *Recuperados* e o título *Os 15 países com o maior número de casos recuperados*. No gráfico gerado é possível analisar que a Índia possui mais casos recuperados que os EUA e o Brasil segue em terceiro com o maior número de casos recuperados.

Figura 46 – Gráfico de barras gerado para a coluna *Recuperados*

```
plt.figure(figsize=(10,13))
plt.subplot(312)
plot_new('Recuperados','Os 15 países com o maior número de casos recuperados')
```



Fonte: O Autor (2021)

Além dos gráficos de barras, um gráfico de pizza também pode ser criado com o presente conjunto de dados. Para gerar um gráfico de pizza foi desenvolvido uma função, ilustrada pela Figura 47. A função tem como parâmetros a variável *x* que representa a legenda do gráfico, a variável *y* que representa os dados a serem exibidos no gráfico e por último a variável *title* que representa o título apresentado no gráfico.

Figura 47 – Função para gerar um gráfico de pizza

```
def plot_pie_charts(x, y, title):
    c = ['lightcoral', 'rosybrown', 'sandybrown', 'navajowhite', 'gold',
        'khaki', 'lightskyblue', 'turquoise', 'lightslategrey', 'thistle', 'pink']
    plt.figure(figsize=(10,12))
    plt.title(title, size=20)
    plt.pie(y, colors=c,shadow=True, labels=y)
    plt.legend(x, loc='best', fontsize=12)
    plt.show()
```

Fonte: O Autor (2021)

Inicialmente foi atribuído a variável *top_10_countries_confirmed_cases* os 10 primeiros países com mais casos confirmados. Após, para o restante dos países foi feita a soma de todas as colunas e atribuído a variável *others*, adicionando uma coluna *País/Região* com o valor de *Outros*, com isso é possível colocar o restante do mundo em evidência. A Figura 48 ilustra o código.

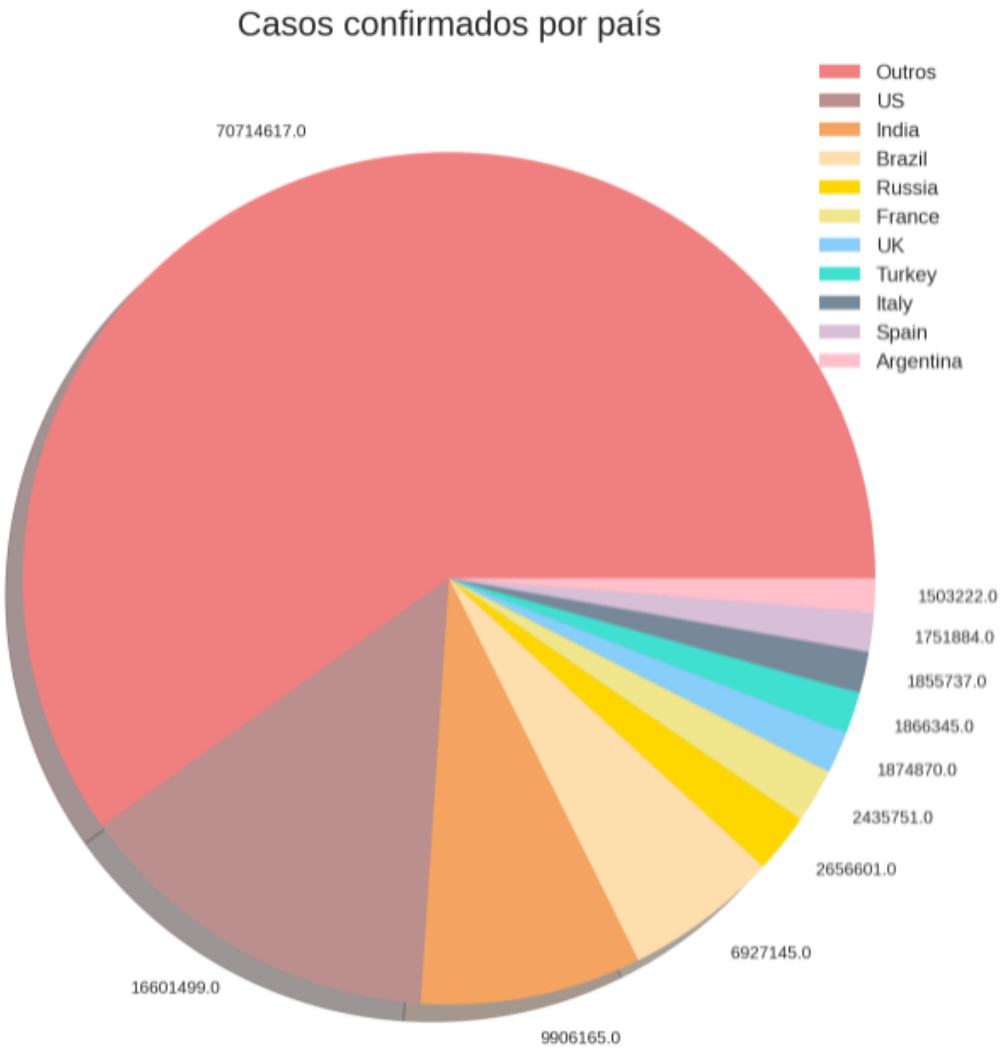
Figura 48 – Código utilizado na preparação dos dados para o gráfico de pizza

```
top_10_countries_confirmed_cases = countries.sort_values('Confirmados', ascending=False).reset_index().head(10)

others = np.sum(countries[10:])
others['País/Região'] = "Outros"

top_10_countries_confirmed_cases = top_10_countries_confirmed_cases.append(others, ignore_index=True)
top_10_countries_confirmed_cases.sort_values('Confirmados', ascending=False, inplace=True)

plot_pie_charts(
    top_10_countries_confirmed_cases['País/Região'],
    top_10_countries_confirmed_cases['Confirmados'], "Casos confirmados por país")
```

Fonte: O Autor (2021)

Com os dados devidamente preparados, foi utilizado a função desenvolvida e ilustrada pela Figura 47 tendo como resultado a Figura 49. Nota-se que um pouco mais de um quarto de todos os casos confirmados no mundo pertence a 10 países.

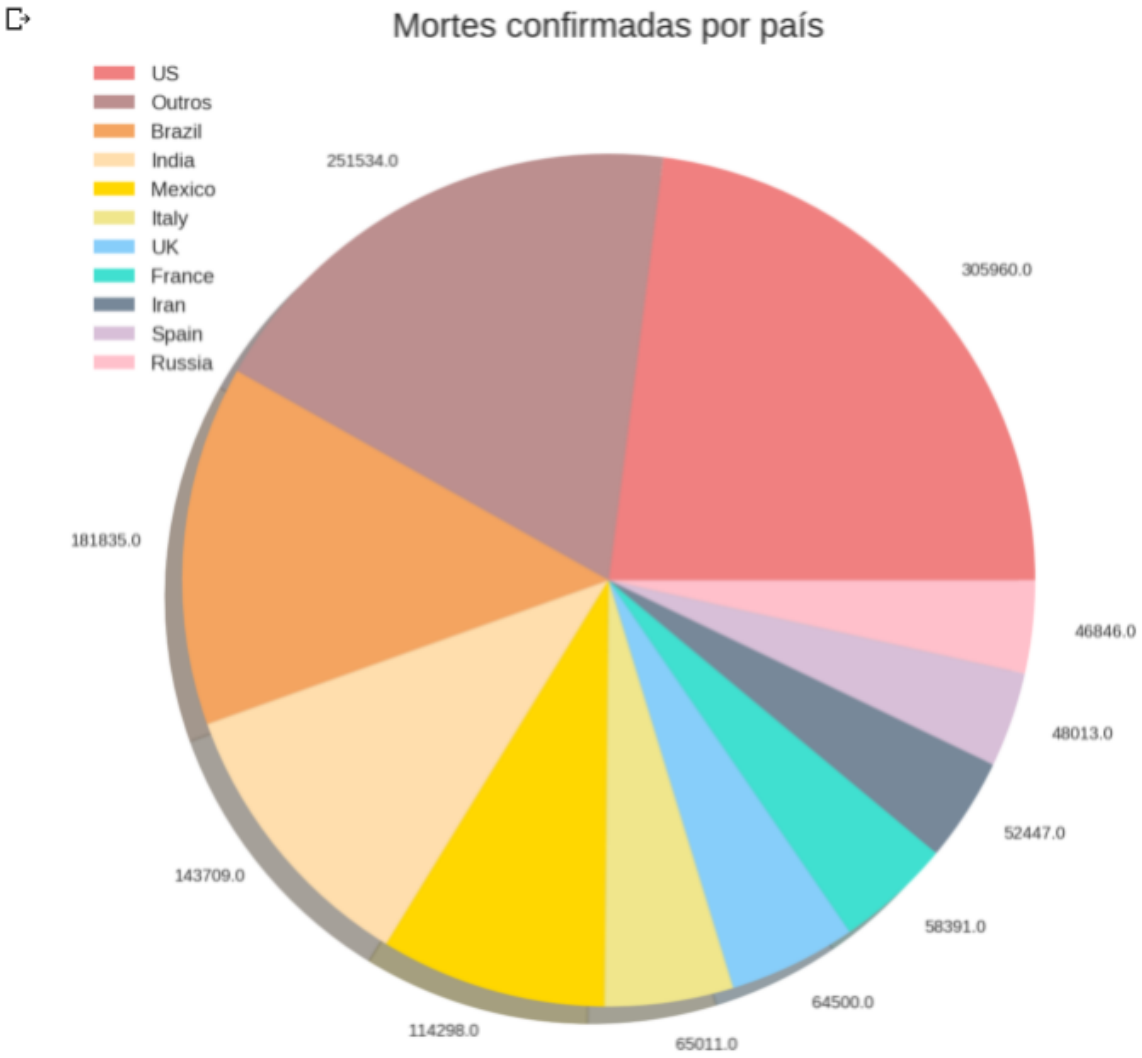
Figura 49 – Gráfio de pizza mostrando os casos confirmados por país



Casos confirmados por país. Fonte: O Autor (2021)

O mesmo gráfico de pizza foi gerado para as mortes confirmadas no mundo ilustrado pela Figura 50. Neste gráfico é possível notar que a maioria das mortes está contida nos 10 países com mais mortes confirmadas e que os EUA possuem mais mortes confirmadas que o restante dos países.

Figura 50 – Gráfio de pizza mostrando as mortes confirmadas por país



Mortes confirmadas por país. Fonte: O Autor (2021)

A biblioteca *Pandas* possui um método chamado *corr()* utilizado para calcular a correlação entre pares de colunas, o método pode ser utilizado diretamente na variável que contém o conjunto de dados. A Figura 51 demonstra o código e o resultado do gráfico gerado.

Figura 51 – Resultado do método *corr()*

```
covid.corr()
```

	Confirmados	Mortes	Recuperados
Confirmados	1.000000	0.858997	0.536756
Mortes	0.858997	1.000000	0.452438
Recuperados	0.536756	0.452438	1.000000

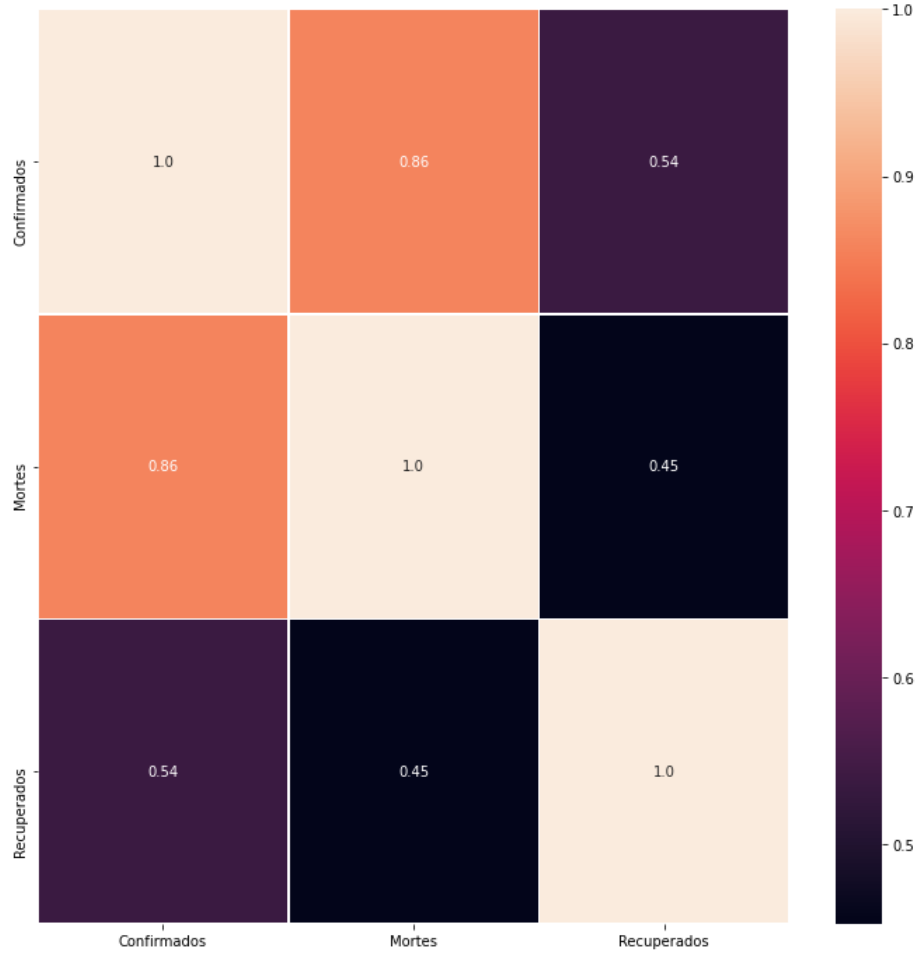
Fonte: O Autor (2021)

Com esses dados gerados utilizando o método *corr()* é possível gerar um mapa de calor através da biblioteca *Seaborn*, possibilitando uma visualização com mais impacto. A Figura 52 demonstra o resultado.

Figura 52 – Mapa de calor gerado através da biblioteca *Seaborn*

```
f,ax=plt.subplots(figsize=(12,12))
sns.heatmap(covid.corr(),annot=True,linewidth=.6,fmt=".2",ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f42d8cdeb10>
```



Fonte: O Autor (2021)

Como este conjunto de dados possui dados do tipo temporais, é possível criar visualizações que demonstram o crescimento dos casos confirmados, recuperados e das mortes causadas por COVID-19 no mundo ao decorrer do tempo. Além disso, como os dados são provenientes de países, é possível gerar um mapa coroplético global. Para isso foi utilizado a biblioteca *Plotly*, mais especificamente o módulo *Express*, Este módulo contém as funções necessárias para criar figuras e gráficos. A Figura 53 ilustra a importação da biblioteca.

Figura 53 – Importação da biblioteca *Ploty*

```
import plotly.express as px
```

Fonte: O Autor (2021)

O primeiro gráfico gerado através da biblioteca *Ploty* é um gráfico linear iterativo, ou seja, ao posicionar-se o ponteiro do *mouse* sobre um ponto da curva, obtém-se o valor representado na devida data. Para isso foi criada uma nova variável denominada de *covid_new_date*, essa variável contém os dados dos países até a data de 14/12/2020, evitando desta maneira os casos recuperados sem valores dos EUA. Após isso, o conjunto de dados foi agrupado por data e para cada data foi realizada a soma de todos países para cada coluna. Com a variável devidamente preparada foi utilizado como o eixo x as datas disponíveis dentro do conjunto de dados e como eixo y os valores das colunas *Confirmados*, *Recuperados* e *Mortes*. A Figura 54 ilustra o código desenvolvido.

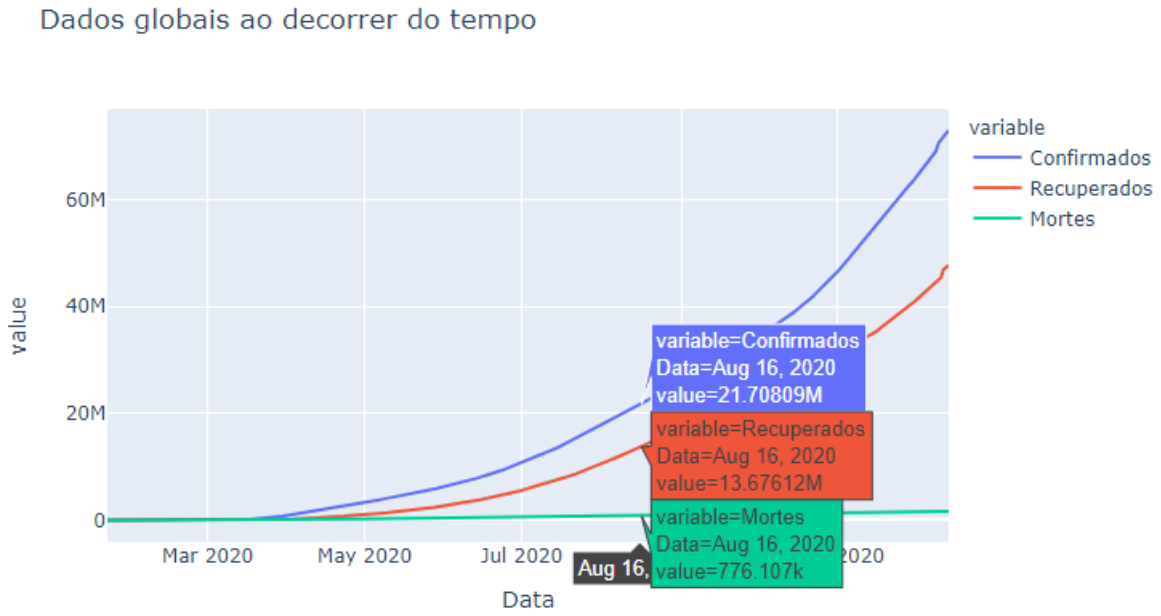
Figura 54 – Código desenvolvido para gerar o gráfico linear iterativo

```
covid_new_date = covid[covid.Data <= last_date]
grouped = covid_new_date.groupby('Data')['Data', 'Confirmados', 'Mortes', 'Recuperados']
grouped = grouped.sum().reset_index()

fig = px.line(grouped, x="Data", y=["Confirmados", "Recuperados", "Mortes"],
              title="Dados globais ao decorrer do tempo", width=750, height=450)
fig.show()
```

Fonte: O Autor (2021)

O resultado é demonstrado pela Figura 55, onde é possível analisar que na data 16/08/20 havia mais de 21 milhões de casos confirmados no mundo, além de mais de 13 milhões de casos recuperados e por fim mais de 776 mil mortes confirmadas ao redor do mundo. Portanto, além da curva de crescimento, é possível analisar isoladamente cada data que o conjunto de dados possui em termos de ocorrências relacionadas ao COVID-19.

Figura 55 – Gráfico linear que ilustra a comparação entre os casos confirmados, recuperados e mortes

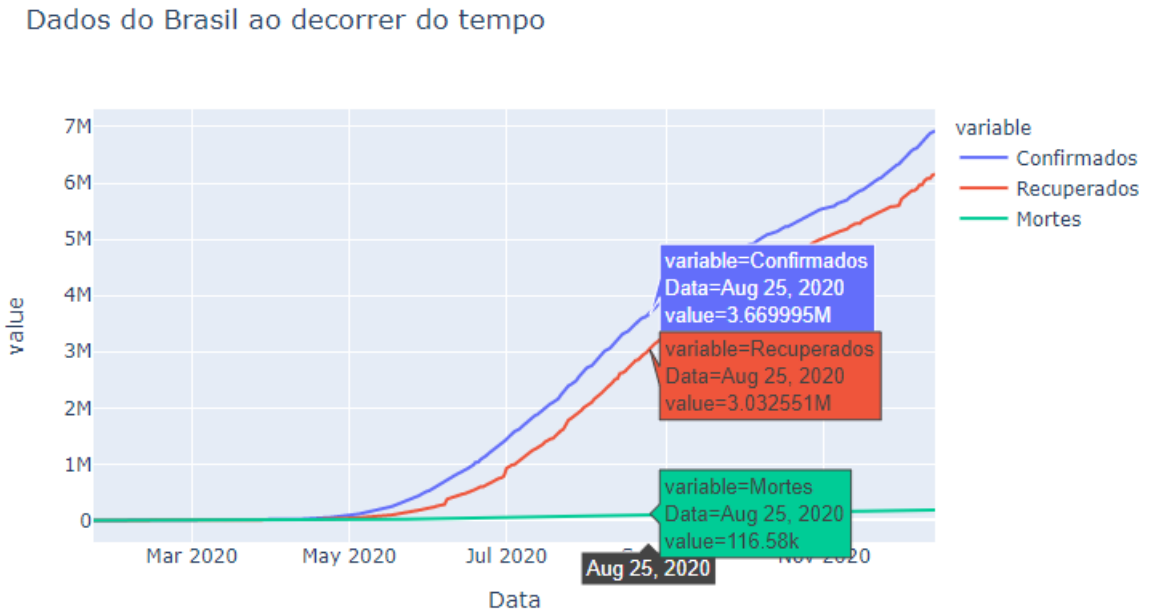


Dados globais ao decorrer do tempo. Fonte: O Autor (2021)

O gráfico gerado pela Figura 55 forma um conjunto que permite comparações entre as curvas de crescimento dos casos, velocidade do crescimento, estabilidade de ocorrências, entre outras verificações numéricas. Com a visualização da sobreposição das curvas, assumindo os mesmos períodos de ano, o usuário pode inferir, ter *insights*, e compreender momentos da pandemia de forma visual.

Similar ao exemplo da Figura 55 também foi gerado um gráfico mostrando os dados do COVID-19 tendo como dados provenientes apenas dados do Brasil. A Figura 56 ilustra o gráfico gerado, onde o ponteiro do *mouse* está em cima da data de 25/08/2020, nesta data o Brasil possuía mais de 3 milhões de casos confirmados e recuperados além de mais de 116 mil mortes.

Figura 56 – Gráfico linear que ilustra a comparação entre os casos confirmados, recuperados e mortes do Brasil apenas



Dados do Brasil ao decorrer do tempo. Fonte: O Autor (2021)

Também foi desenvolvido um gráfico similar contendo os dados dos 10 primeiros países com maior número de casos confirmados, colocando em evidência as diferentes curvas de crescimento de casos para cada país.

Para isso foi desenvolvido o trecho de código ilustrado pela Figura 57. Utilizando a variável *countries* criada anteriormente, ordenou-se pela coluna *Confirmados* e atribui-se o resultado para a variável *top_10_countries_confirmed_cases*. Após, foi selecionado o primeiro país e com base na coluna *País/Região* foram selecionados da variável *covid_new_date* somente os dados deste país, que no caso, o país selecionado foi o EUA. Utilizando esses dados como parâmetro foi criado a primeira linha no gráfico, que é a linha azul, tendo novamente como o eixo x as datas disponíveis e como eixo y os valores da coluna *Confirmados*.

Posteriormente foi realizado o mesmo processo para o restante dos 9 países. A única diferença foi no modo em que os dados foram adicionados ao gráfico. Para poder comparar as curvas no mesmo gráfico foi utilizado a figura gerada anteriormente e concatenado os novos dados. É possível identificar as curvas com o auxílio da legenda que contém as cores e respectivos países no gráfico.

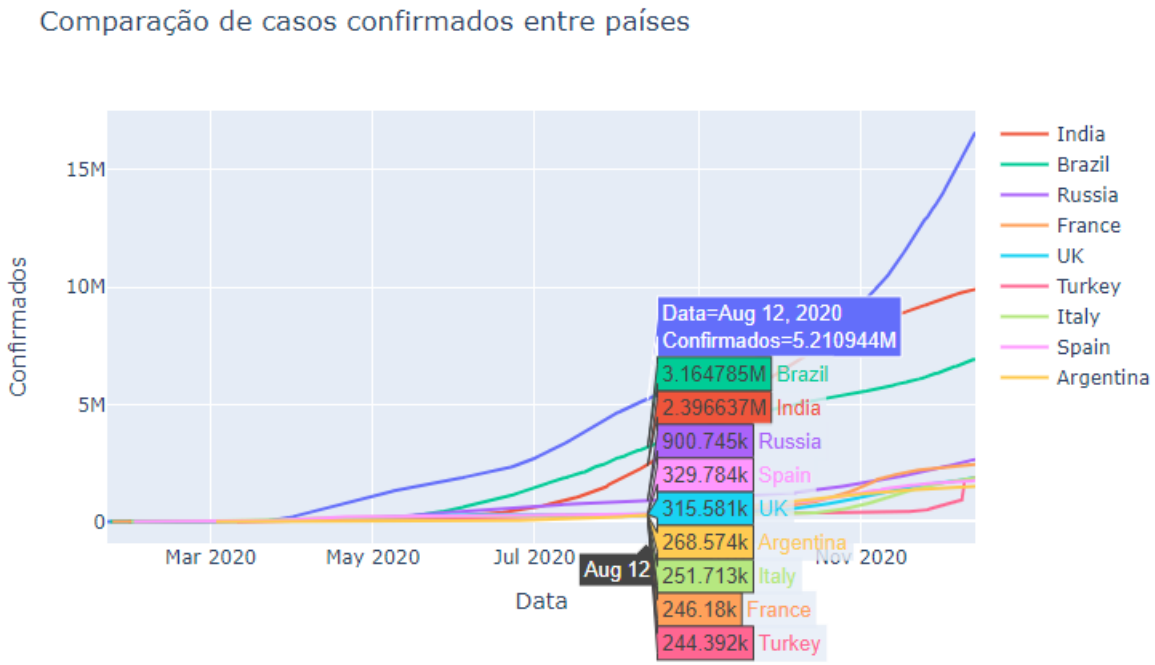
Figura 57 – Código para demonstrar os países com mais casos confirmados

```
top_10_countries_confirmed_cases = countries.sort_values('Confirmados', ascending=False).reset_index().head(10)
first_country = top_10_countries_confirmed_cases.iloc[0]
first_country = covid_new_date[covid_new_date['País/Região'] == first_country['País/Região']]
first_country = first_country.groupby('Data')['Data', 'Confirmados', 'Mortes'].sum().reset_index()
fig = px.line(first_country, x="Data", y="Confirmados",
              title="Comparação de casos confirmados entre países", width=750, height=450)

top_9_countries_confirmed_cases = top_10_countries_confirmed_cases[1:10]

for country_name in top_9_countries_confirmed_cases['País/Região']:
  next_country = covid_new_date[covid_new_date['País/Região'] == country_name]
  next_country = next_country.groupby('Data')['Data', 'Confirmados', 'Mortes'].sum().reset_index()
  fig.add_trace(go.Scatter(
    x=next_country.Data,
    y=next_country.Confirmados,
    name=country_name))

fig.show()
```

Fonte: O Autor (2021)

A Figura 58 ilustra o resultado gerado. É possível analisar que nos países EUA, Índia e Brasil as curvas cresceram consideravelmente mais em comparação ao restante dos países, um provável fator para estes resultados é a densidade populacional desses países.

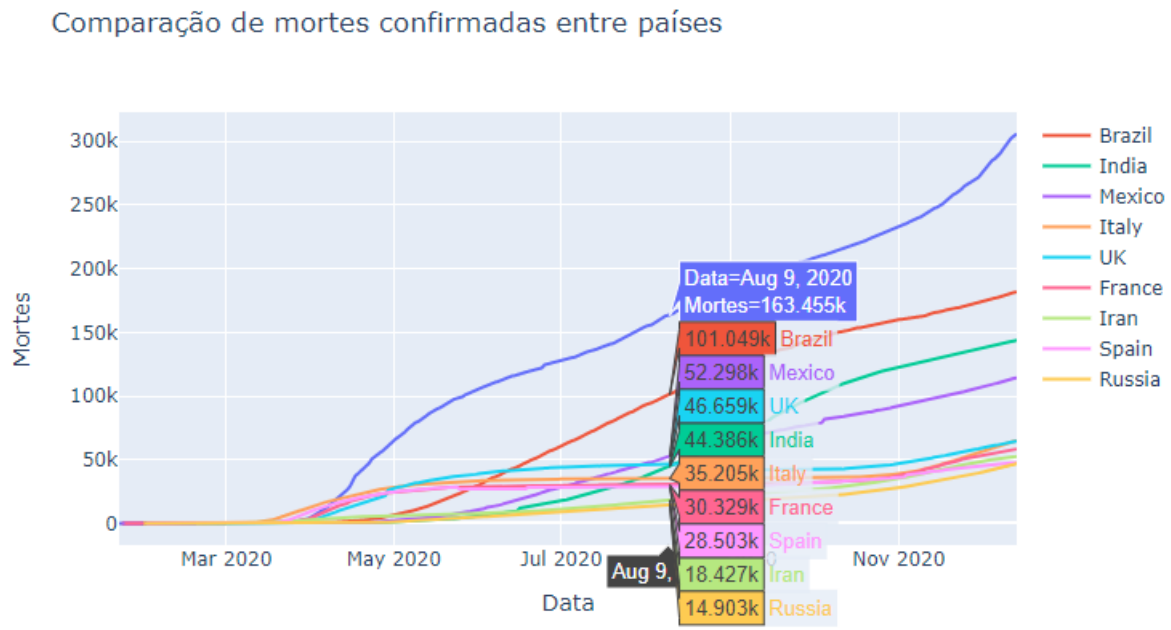
Figura 58 – Gráfico linear interativo que ilustra a comparação de casos confirmados entre os países



Comparação de casos confirmados entre países. Fonte: O Autor (2021)

Utilizando o mesmo processo usado para gerar o gráfico da Figura 58, foi criado um novo gráfico colocando em evidência desta vez a coluna *Mortes*. A Figura 59 ilustra o resultado obtido.

Figura 59 – Gráfico linear interativo que ilustra a comparação das mortes confirmadas entre os países



Comparação de mortes confirmadas entre países. Fonte: O Autor (2021)

Além de gráficos lineares interativos, utilizando a biblioteca *Ploty* é possível desenvolver mapas coropléticos. Para isso foi utilizada a variável criada anteriormente, denominada de *latest_register* contém os dados relacionados ao COVID-19 até a data de 14/12/2020. Esses dados foram agrupados pela coluna *País/Região* e atribuído a uma nova variável chamada de *countries_deaths_confirmed_cases*, essa nova variável foi usada como conjunto de dados no método da biblioteca *Ploty* chamado de *choropleth*. Além do conjunto de dados, entre outros parâmetros importantes estão o *location* que é responsável por identificar a localização no mapa e o parâmetro *color* que utiliza a coluna escolhida para demonstrar através da intensidade de cores a variância entre o valores. A Figura 60 demonstra o código desenvolvido.

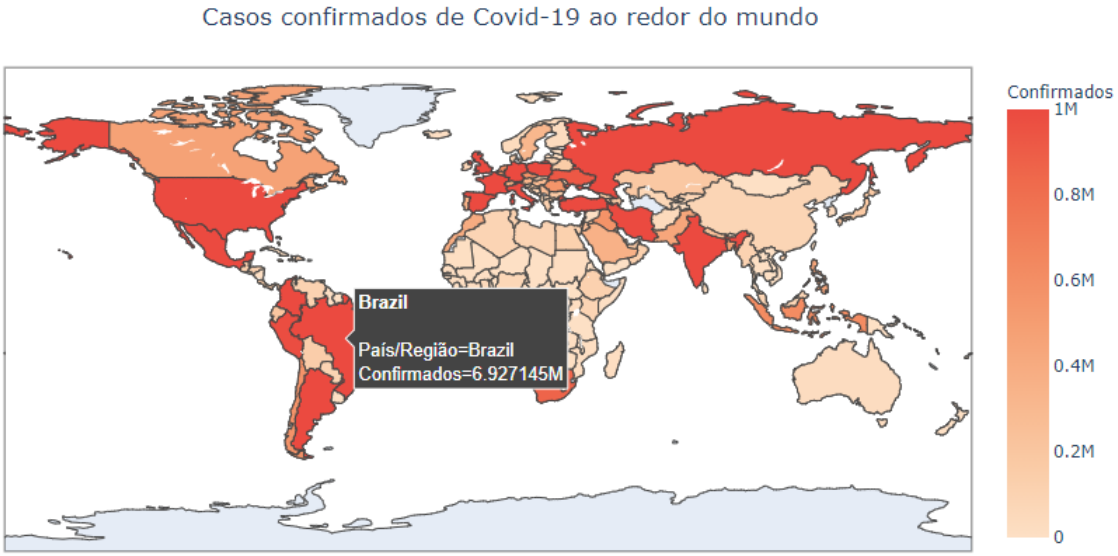
Figura 60 – Código para gerar o mapa coroplético dos casos confirmados

```
countries_deaths_confirmed_cases = latest_register.groupby('País/Região')['Confirmados', 'Mortes']
countries_deaths_confirmed_cases = countries_deaths_confirmed_cases.sum().reset_index()

fig = px.choropleth(countries_deaths_confirmed_cases, locations="País/Região",
                    locationmode='country names', color="Confirmados",
                    hover_name="País/Região", range_color=[1,1000000],
                    color_continuous_scale="peach",
                    title='Casos confirmados de Covid-19 ao redor do mundo')
fig.show()
```

Fonte: O Autor (2021)

A Figura 61 demonstra o resultado gerado. É possível analisar os dados de cada país separadamente, assim como a imagem ilustra, no caso do Brasil haviam quase 7 milhões de casos confirmados de COVID-19 na data de 14/12/2020.

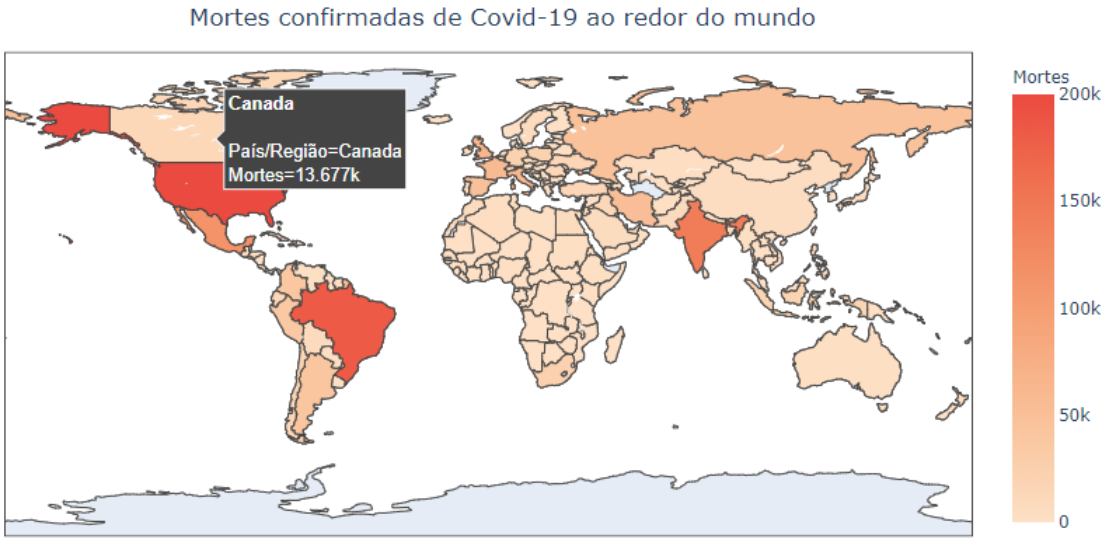
Figura 61 – Mapa coroplético mundial tendo os casos confirmados em evidência



Casos confirmados de Covid-19 ao redor do mundo - Fonte: O Autor (2021)

Similar ao mapa ilustrado pela Figura 61 foi criado um mapa colocando em evidência as mortes confirmadas provenientes do COVID-19. A Figura 62 mostra o resultado, desta vez o ponteiro do *mouse* está em cima do Canadá, que na data de 14/12/2020 possuía quase 14 mil mortes confirmadas.

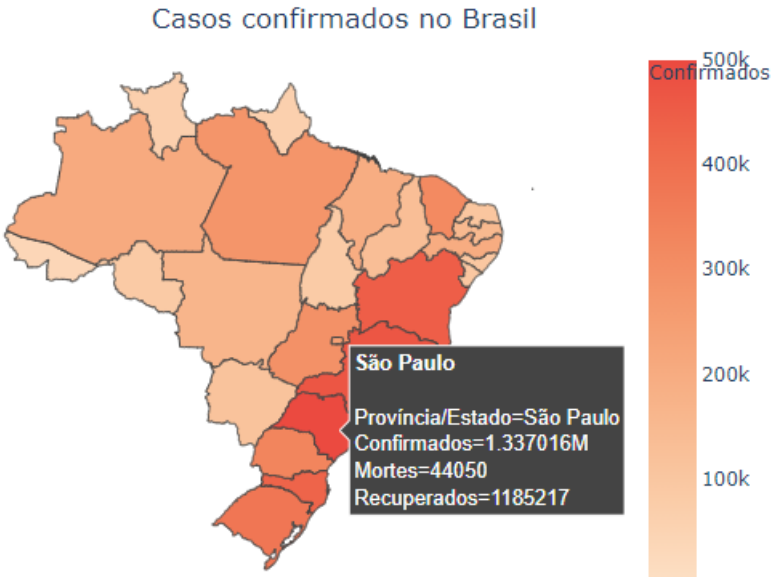
Figura 62 – Mapa coroplético mundial tendo as mortes confirmadas em evidência



Mortes confirmadas de Covid-10 ao redor do mundo. Fonte: O Autor (2021)

O conjunto de dados que está sendo utilizado possui uma coluna chamada *Província/Estado*, com estes dados é possível gerar um mapa dos estados brasileiros e analisar a evolução da doença em nível nacional. Neste mapa, a intensidade das cores representa os casos confirmados para cada estado na data de 14/12/2020. Além disso, é possível analisar também os casos recuperados e as mortes confirmadas ao posicionar o *mouse* em cima dos estados. A Figura 63 ilustra que nesta data no estado de São Paulo havia mais de 1 milhão e 300 mil casos de COVID-19 confirmados, mais de 1 milhão de casos recuperados e mais de 44 mil mortes confirmadas.

Figura 63 – Mapa coroplético do Brasil



Casos confirmados no Brasil. Fonte: O Autor (2021)

Visto que este conjunto de dados constitui uma série temporal, tem-se como objetivo criar previsões de valores futuros, tendo como base os dados atuais. Para aplicar os algoritmos de *Machine Learning* foi utilizado o arquivo de série temporal que possui dados dos casos confirmados. Diferente do arquivo que foi trabalhado até agora, este arquivo é atualizado diariamente. Portanto, possui dados desde o dia 22/01/2020 até os dias atuais.

Este arquivo atualizado pode ser encontrado no repositório do *GitHub*[17] da Universidade *Johns Hopkins*, e possui sempre o mesmo endereço na *web*, ou seja, possui uma *URL* única. Dito isso, o método *read_csv* da biblioteca *Pandas* possibilita a realização da leitura de um arquivo através do endereço de uma *URL* como parâmetro. A Figura 64 ilustra o processo de carregamento da série temporal e o método *head()* que demonstra as 5 primeiras linhas do conjunto de dados. Este conjunto de dados possui as colunas *Province/State* que representam as províncias e estados, *Country/Region* que representam os países e as regiões, *Lat* que representa a latitude, *Long* que representa a longitude e por fim as datas desde o dia 22/01/2020 até os dias atuais.

[17] github.com/cSSEGISandData/COVID-19

Figura 64 – Carregando um arquivo *CSV* diretamente de uma *URL*

```
url = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv'
confirmed_df = pd.read_csv(url)
confirmed_df.head()
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/20	1/30/20	1/31/20	2/1/20	2/2/20
0	NaN	Afghanistan	33.93911	67.709953	0	0	0	0	0	0	0	0	0	0	0	0
1	NaN	Albania	41.15330	20.168300	0	0	0	0	0	0	0	0	0	0	0	0
2	NaN	Algeria	28.03390	1.659600	0	0	0	0	0	0	0	0	0	0	0	0
3	NaN	Andorra	42.50630	1.521800	0	0	0	0	0	0	0	0	0	0	0	0
4	NaN	Angola	-11.20270	17.873900	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 514 columns

Fonte: O Autor (2021)

Para preparar os dados seguintes foi necessário o uso de duas bibliotecas, a primeira denominada *NumPy*, esta biblioteca adiciona suporte durante a preparação de matrizes multidimensionais, junto com uma grande coleção de funções matemáticas de alto nível para operar essas matrizes. A segunda biblioteca é denominada de *sklearn* e dois módulos provenientes desta biblioteca foram utilizados, o primeiro é denominado de *train_test_split* que será utilizado para dividir um conjunto de dados em dados de treinamento e dados de teste dos modelos que serão criados, o segundo módulo é chamado de *PolynomialFeatures* utilizado para gerar recursos polinomiais. A Figura 65 ilustra a importação das bibliotecas.

Figura 65 – Bibliotecas utilizadas no pré-processamento dos dados

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
```

Fonte: O Autor (2021)

Os valores visados a serem previstos dizem respeito aos casos confirmados de todo o mundo. Para isso, primeiramente foi utilizado o método *keys()* proveniente da biblioteca *Pandas*, este método seleciona apenas as colunas do conjunto de dados e atribui-as para a variável *cols*, após isso, através do método *loc* foram selecionadas apenas as colunas das datas e para cada data disponível, foi realizada a soma dos casos confirmados de todos os países e atribuído a uma nova variável denominada de *world_cases*. A Figura 66 demonstra o código desenvolvido.

Figura 66 – Código desenvolvido para preparar os dados para os algoritmos de *Machine Learning*

```
cols = confirmed_df.keys()

confirmed = confirmed_df.loc[:, cols[4]:cols[-1]]

dates = confirmed.keys()

world_cases = []

for i in dates:
    confirmed_sum = confirmed[i].sum()

    world_cases.append(confirmed_sum)
```

Fonte: O Autor (2021)

Como resultado foram obtido dois *arrays*, o primeiro *array* foi atribuído a variável *dates* que possui todas as datas do conjunto de dados e o segundo constitui a variável *world_cases*, que para cada data possui o número total de casos confirmados de COVID-19 de todos países.

Após a criação dos *arrays*, criou-se duas variáveis que serão utilizadas posteriormente pelos modelos para prever resultados e para colocar os dados atuais em evidência. A variável *future_forcast* possui um *array* com as datas disponíveis com um adicional de 10 dias, representando datas futuras, e será utilizadas pelos modelos para prever novos valores. Já a variável *adjusted_dates* possui as datas disponíveis no conjunto de dados e será utilizada nas visualizações com o objetivo de comparar os resultados obtidos. A Figura 67 demonstra o trecho de código desenvolvido.

Figura 67 – Código desenvolvido para preparar as datas

```
days_in_future = 10
future_forcast = np.array([i for i in range(len(dates)+days_in_future)]).reshape(-1, 1)
adjusted_dates = future_forcast[:-days_in_future]
```

Fonte: O Autor (2021)

Utilizando as variáveis criadas até então foram criados dois conjuntos de dados, um conjunto de dados para treinamento dos modelos e um conjunto de dados para testar a eficiência dos modelos. Para isso, foi utilizado o método *train_test_split* proveniente da biblioteca *sklearn*.

Figura 68 – Código desenvolvido para preparar os dados de treinamento e testes

```
days_since_1_22 = np.array([i for i in range(len(dates))]).reshape(-1, 1)
X_train_confirmed, X_test_confirmed, y_train_confirmed, y_test_confirmed
                  = train_test_split(days_since_1_22[50:], world_cases[50:], test_size=0.03, shuffle=False)
```

Fonte: O Autor (2021)

Por último, como posteriormente os algoritmos de regressão linear e de *bayesian ridge* serão utilizados, foi necessário um pré-processamento dos resultados obtidos através do método

train_test_split, para tornar os dados adequados para os algoritmos de função polinomial. Para isso utilizou-se o módulo previamente importado *PolynomialFeatures*, o parâmetro *degree* representa o grau dos recursos polinomiais e o método *fit_transform* ajusta os dados e transforma-os de acordo com o grau polinomial. Para o algoritmo de regressão linear utilizou-se grau 4 e para o algoritmo de *bayesian ridge* utilizou-se grau 5. A Figura 69 ilustra o código desenvolvido.

Figura 69 – Código desenvolvido para preparar os dados para os algoritmos polinomiais

```
poly = PolynomialFeatures(degree=4)
poly_X_train_confirmed = poly.fit_transform(X_train_confirmed)
poly_X_test_confirmed = poly.fit_transform(X_test_confirmed)
poly_future_forcast = poly.fit_transform(future_forcast)

bayesian_poly = PolynomialFeatures(degree=5)
bayesian_poly_X_train_confirmed = bayesian_poly.fit_transform(X_train_confirmed)
bayesian_poly_X_test_confirmed = bayesian_poly.fit_transform(X_test_confirmed)
bayesian_poly_future_forcast = bayesian_poly.fit_transform(future_forcast)
```

Fonte: O Autor (2021)

3.4.3 Aplicação de algoritmos de *Machine Learning* e geração de visualizações correspondentes

Com os dados devidamente preparados, os algoritmos escolhidos a serem implementados foram a regressão linear, máquina de vetores de suporte e por último o *bayesian ridge*. Todos os algoritmos utilizam a técnica de aprendizado supervisionado e resolvem problemas de regressão já que os dados a serem previstos são numéricos.

Para realizar a criação dos modelos foi utilizado novamente módulos da biblioteca *sklearn*. Esses módulos facilitam o processo de desenvolvimento de modelos de *Machine Learning* pois abstraem os algoritmos. Para cada algoritmo importou-se um módulo específico. Além disso, o módulo *RandomizedSearchCV* foi importado e será utilizado na busca dos parâmetros que serão utilizados no algoritmo de máquina de vetores de suporte. A Figura 70 ilustra a importação dos módulos.

Figura 70 – Módulos importados para a criação dos modelos de *Machine Learning*

```
from sklearn.linear_model import LinearRegression, BayesianRidge
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVR
```

Fonte: O Autor (2021)

O primeiro algoritmo a ser implementado será o máquina de vetores de suporte, para isso foi criado o modelo e foi atribuído para a variável *SVM*, o parâmetro *kernel* especifica o algoritmo utilizado e próximo parâmetro *degree* representa o grau da função utilizado pelo algoritmo.

Com o modelo criado, utilizou-se o método proveniente do módulo *RandomizedSearchCV* que realiza uma busca de parâmetros ótimos para um modelo. Este método tem primeiramente como parâmetro o modelo que está sendo feita a busca dos melhores parâmetros, uma lista que contém parâmetros a serem testados e sua variância de valores, que neste caso estão contidos na variável *svm_grid* e o parâmetro *scoring* utilizado para avaliar o desempenho do modelo.

Ao término do método de busca, foi utilizado o método *fit* que realiza o treinamento do modelo, como parâmetros foram utilizados os dados de treinamento criados na seção 3.4.2. Após o fim do treinamento do modelo é possível acessar uma variável denominada *best_estimator_* que contém o melhor estimador que foi encontrado pela pesquisa, ou seja, o estimador que contém a maior pontuação, foi atribuído o melhor estimador para a variável *svm_confirmed*.

Com o melhor estimador disponível foi utilizado o método *predict*, este método aplica a regressão no conjunto de dados passado como parâmetro, que neste caso foi a variável *future_forcast* preparada anteriormente, atribui-se então o resultado para a variável *svm_pred*, que será utilizada posteriormente para comparação de resultados. A Figura 71 ilustra o código desenvolvido.

Figura 71 – Código desenvolvido para implementar o algoritmo de máquina de vetores de suporte

```
c = [0.01, 0.1, 1]
gamma = [0.01, 0.1, 1]
epsilon = [0.01, 0.1, 1]
shrinking = [True, False]

svm_grid = {'C': c, 'gamma' : gamma, 'epsilon': epsilon, 'shrinking' : shrinking}

svm = SVR(kernel='poly', degree=3)
svm_search = RandomizedSearchCV(
    svm, svm_grid, scoring='neg_mean_squared_error',
    cv=3, return_train_score=True, n_jobs=-1, n_iter=30,
    verbose=1)
svm_search.fit(X_train_confirmed, y_train_confirmed)

svm_confirmed = svm_search.best_estimator_
svm_pred = svm_confirmed.predict(future_forcast)
```

Fonte: O Autor (2021)

O segundo algoritmo a ser implementado foi o algoritmo denominado de regressão linear. Primeiro criou-se o modelo utilizando o módulo específico do algoritmo chamado de *LinearRegression*, posteriormente foi realizado o treinamento do modelo utilizando os mesmos dados de treinamento do modelo anterior. Por fim utilizou-se o método *predict* utilizando como parâmetro o conjunto de dados ajustado para algoritmos polinomiais de grau 4 atribuindo os resultados para a variável *linear_pred*. A Figura 72 demonstra o código desenvolvido.

Figura 72 – Código desenvolvido para implementar o algoritmo de regressão linear

```
linear_model = LinearRegression(normalize=True, fit_intercept=False)
linear_model.fit(poly_X_train_confirmed, y_train_confirmed)
linear_pred = linear_model.predict(poly_future_forcast)
```

Fonte: O Autor (2021)

O terceiro e último algoritmo a ser implementado foi o algoritmo denominado de *bayesian ridge*. Para isso utilizou-se o módulo denominado *BayesianRidge* para criar o modelo. Similar ao algoritmo máquina de vetores de suporte utilizou-se a busca *RandomizedSearchCV* para encontrar os melhores parâmetros possíveis.

Após a finalização da busca pelos melhores parâmetros realizou-se o método *fit* que aplica o treinamento do modelo, utilizando como parâmetro os dados preparados especificamente para esta função na seção 3.4.2.

Por último, com o modelo devidamente treinado, foi selecionado o melhor preditor e então realizado o método *predict*, utilizando como parâmetro o conjunto de dados ajustado para algoritmos polinomiais de grau 5, após atribuiu-se os resultados para a variável *bayesian_pred*. A Figura 73 ilustra o código desenvolvido.

Figura 73 – Código desenvolvido para implementar o algoritmo de *bayesian ridge*

```
tol = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2]
alpha_1 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
alpha_2 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
lambda_1 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
lambda_2 = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3]
normalize = [True, False]

bayesian_grid = {'tol': tol, 'alpha_1': alpha_1, 'alpha_2' : alpha_2,
                 'lambda_1': lambda_1, 'lambda_2' : lambda_2,
                 'normalize' : normalize}

bayesian = BayesianRidge(fit_intercept=False)
bayesian_search = RandomizedSearchCV(
    bayesian, bayesian_grid, scoring='neg_mean_squared_error',
    cv=3, return_train_score=True, n_jobs=-1, n_iter=40, verbose=1)
bayesian_search.fit(bayesian_poly_X_train_confirmed, y_train_confirmed)

bayesian_confirmed = bayesian_search.best_estimator_
bayesian_pred = bayesian_confirmed.predict(bayesian_poly_future_forcast)
```

Fonte: O Autor (2021)

3.4.4 Definição de critérios de avaliação para o modelo e aplicação de visualizações

Para análise deste conjunto de dados, utilizou-se os seguintes critérios de avaliação para os modelos: erro médio absoluto (MAE), porcentagem de erro médio absoluto e o coeficiente de determinação (R2). Para obter-se os valores dessas métricas, utilizou-se os módulos *r2_score* e *mean_absolute_error* provenientes da biblioteca *sklearn*, além da definição de uma função para calcular a porcentagem de erro médio absoluto. A Figura 74 ilustra o código utilizado para importar os módulos e definir a função.

Figura 74 – Código utilizado para importar os módulos para realizar as métricas

```
from sklearn.metrics import r2_score, mean_absolute_error

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

Fonte: O Autor (2021)

Para cada modelo criado foi realizado novamente o método *predict*, porém desta vez tendo como parâmetro os dados de teste criados na seção 3.4.2. Atribui-se os resultados provenientes para variáveis correspondentes, sendo elas *svm_test_pred* para o modelo que aplica o algoritmo de máquina de suporte de vetores, *test_linear_pred* para o modelo que aplica o algoritmo de regressão linear e *test_bayesian_pred* para o modelo que aplica o algoritmo de *bayesian bridge*, possibilitando desta maneira a análise das métricas de cada modelo. A Figura 75 ilustra o código desenvolvido.

Figura 75 – Código utilizado para analisar os modelos criados no conjunto de testes

```
svm_test_pred = svm_confirmed.predict(X_test_confirmed)
test_linear_pred = linear_model.predict(poly_X_test_confirmed)
test_bayesian_pred = bayesian_confirmed.predict(bayesian_poly_X_test_confirmed)
```

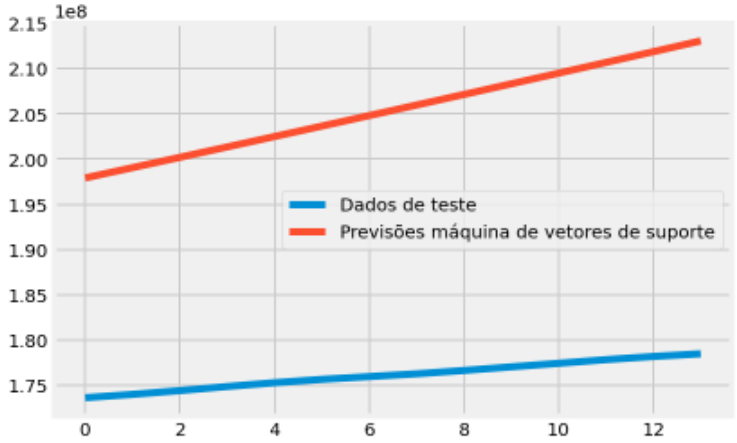
Fonte: O Autor (2021)

Com os resultados de teste obtidos foram criados gráficos colocando em evidência os valores previstos pelos modelos e os valores contidos no conjunto de testes. Para cada modelo foi gerado um gráfico similar, os gráficos são ilustrados pelas Figuras 76, 77 e 78.

Figura 76 – Gráfico de previsões do modelo de máquina de vetores de suporte

```
plt.plot(y_test_confirmed)
plt.plot(svm_test_pred)
plt.legend(['Dados de teste', 'Previsões máquina de vetores de suporte'])
```

```
<matplotlib.legend.Legend at 0x7f87fead0ed0>
```

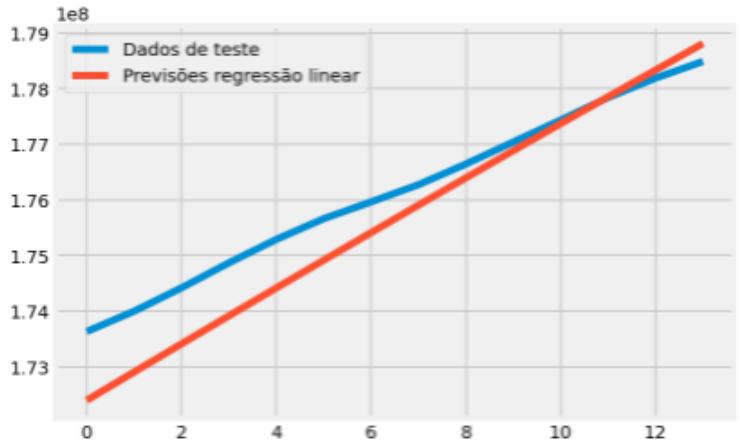


Fonte: O Autor (2021)

Figura 77 – Gráfico de previsões do modelo de regressão linear

```
plt.plot(y_test_confirmed)
plt.plot(test_linear_pred)
plt.legend(['Dados de teste', 'Previsões regressão linear'])
```

```
<matplotlib.legend.Legend at 0x7f87fea54f10>
```

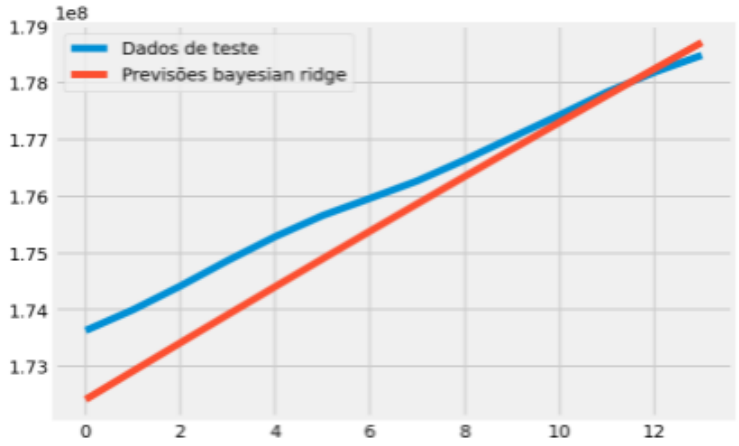


Fonte: O Autor (2021)

Figura 78 – Gráfico de previsões do modelo de *bayseian ridge*

```
plt.plot(y_test_confirmed)
plt.plot(test_bayesian_pred)
plt.legend(['Dados de teste', 'Previsões bayesian ridge'])
```

```
<matplotlib.legend.Legend at 0x7f87fea100d0>
```



Fonte: O Autor (2021)

As Figuras 76, 77 e 78 possibilitam a análise individual de cada modelo. Ao analisar o modelo que utiliza o algoritmo de máquina de vetores de suporte nota-se que as previsões estão muito distantes do conjunto de testes, já os modelos de regressão linear e de *bayesian ridge* possuem valores mais próximos do conjunto de testes, além de parecerem muito similares devido a aparência de suas retas.

Além da comparação visual realizada através dos gráficos, mediu-se para cada modelo a qualidade de suas previsões utilizando as métricas de erro médio absoluto (MAE), porcentagem de erro médio absoluto e o coeficiente de determinação (R^2).

Primeiramente calculou-se o coeficiente de determinação, geralmente denominado como R^2, essa métrica fornece uma medida de quão bem as amostras não vistas podem ser previstas pelo modelo, sendo a melhor pontuação possível de 1,0 e a pontuação também pode ser negativa, porque o modelo pode ser arbitrariamente pior. Para isso utilizou-se o módulo *r2_score* previamente importado, e como parâmetros foram utilizados os valores de testes e os valores previstos para cada modelo. A Figura 79 demonstra os valores obtidos.

Figura 79 – Pontuação R2 dos modelos desenvolvidos

```
print('Máquina de vetores de suporte - Pontuação R2: {0:.4f}'.format(r2_score(y_test_confirmed, svm_test_pred)))
print('Regressão Linear              - Pontuação R2: {0:.4f}'.format(r2_score(y_test_confirmed, test_linear_pred)))
print('Bayesian Ridge                - Pontuação R2: {0:.4f}'.format(r2_score(y_test_confirmed, test_bayesian_pred)))
```

```
Máquina de vetores de suporte - Pontuação R2: -381.8529
Regressão Linear              - Pontuação R2: 0.7919
Bayesian Ridge                - Pontuação R2: 0.7916
```

Fonte: O Autor (2021)

Analisando a Figura 79, observa-se que a pontuação do modelo que utiliza o algoritmo de máquina de vetores esta extremamente negativa e para os outros dois modelos a pontuação está próxima de 1, além de possuírem valores quase idênticos, demonstrando novamente resultados melhores para os modelos de regressão linear e *bayesian bridge*.

Posteriormente calculou-se o erro médio absoluto para cada modelo, esta métrica é uma medida de erros entre valores pareados, neste caso entre o conjunto de teste e os valores previstos pelos modelos. Para realizar o cálculo foi usado o módulo *mean_absolute_error*. A Figura 80 ilustra os resultados obtidos.

Figura 80 – Erro médio absoluto dos modelos desenvolvidos

```
print('Máquina de vetores de suporte - MAE: {0:.2f}'.format(mean_absolute_error(y_test_confirmed, svm_test_pred)))
print('Regressão Linear              - MAE: {0:.2f}'.format(mean_absolute_error(y_test_confirmed, test_linear_pred)))
print('Bayesian Ridge                - MAE: {0:.2f}'.format(mean_absolute_error(y_test_confirmed, test_bayesian_pred)))
```

```
Máquina de vetores de suporte - MAE: 29308313.47
Regressão Linear              - MAE: 555438.69
Bayesian Ridge                - MAE: 561249.62
```

Fonte: O Autor (2021)

A Figura 80 ilustra o erro médio absoluto para cada modelo, sendo possível analisar que os valores obtidos são altos, isso é devido aos valores no conjunto de teste e os valores previstos serem contidos por valores presentes nas centenas de milhões, tendo em vista que os valores que estão sendo analisados representam os casos confirmados de COVID-19 no mundo inteiro.

Visto isso, a porcentagem de erro médio absoluto será calculada para cada modelo, provendo um entendimento melhor dos resultados. A Figura 81 mostra os resultados obtidos, mostrando que o modelo que implementa o algoritmo de máquina de vetores possui uma porcentagem de erro 16.63%, bem elevada em comparação com os outros dois modelos que tem a porcentagem de 0.32%.

Figura 81 – Porcentagem de erro médio absoluto dos modelos desenvolvidos

```
print('Máquina de vetores de suporte - Porcentagem MAE: {0:.2f}%'
      .format(mean_absolute_percentage_error(y_test_confirmed, svm_test_pred)))
print('Regressão Linear              - Porcentagem MAE: {0:.2f}%'
      .format(mean_absolute_percentage_error(y_test_confirmed, test_linear_pred)))
print('Bayesian Ridge                - Porcentagem MAE: {0:.2f}%'
      .format(mean_absolute_percentage_error(y_test_confirmed, test_bayesian_pred)))
```

```
Máquina de vetores de suporte - Porcentagem MAE: 16.63%
Regressão Linear              - Porcentagem MAE: 0.32%
Bayesian Ridge                - Porcentagem MAE: 0.32%
```

Fonte: O Autor (2021)

3.4.5 Seleção do modelo de melhor desempenho e visualizações para justificar a escolha

Com base nos resultados obtidos através dos cálculos das métricas é possível realizar a escolha do melhor modelo desenvolvido. Os modelos que implementam os algoritmos de regressão linear e de *bayesian ridge* foram os modelos que se destacaram, tendo resultados bastante parecidos entre si e superiores ao modelo que implementa o algoritmo de máquina de suporte de vetores.

Utilizando os resultados obtidos ao realizar o método *predict* para cada modelo foram criados gráficos para cada modelo, sendo possível comparar os dados previstos pelos modelos e os dados reais. Foi desenvolvido uma função para evitar a repetição de código, a Figura 82 ilustra o código desenvolvido para gerar esses gráficos. A função tem como parâmetros a variável *x* que representa as datas disponíveis com o adicional das 10 datas extras, a variável *y* que representa o número de casos confirmados na respectiva data, a variável *pred* que representa os valores obtidos pelos modelos ao realizar o método *predict*, a variável *algo_name* que representa o nome do algoritmo utilizado e por último a variável *color* que representa a cor que será imprimida na tela para a respectiva curva do algoritmo.

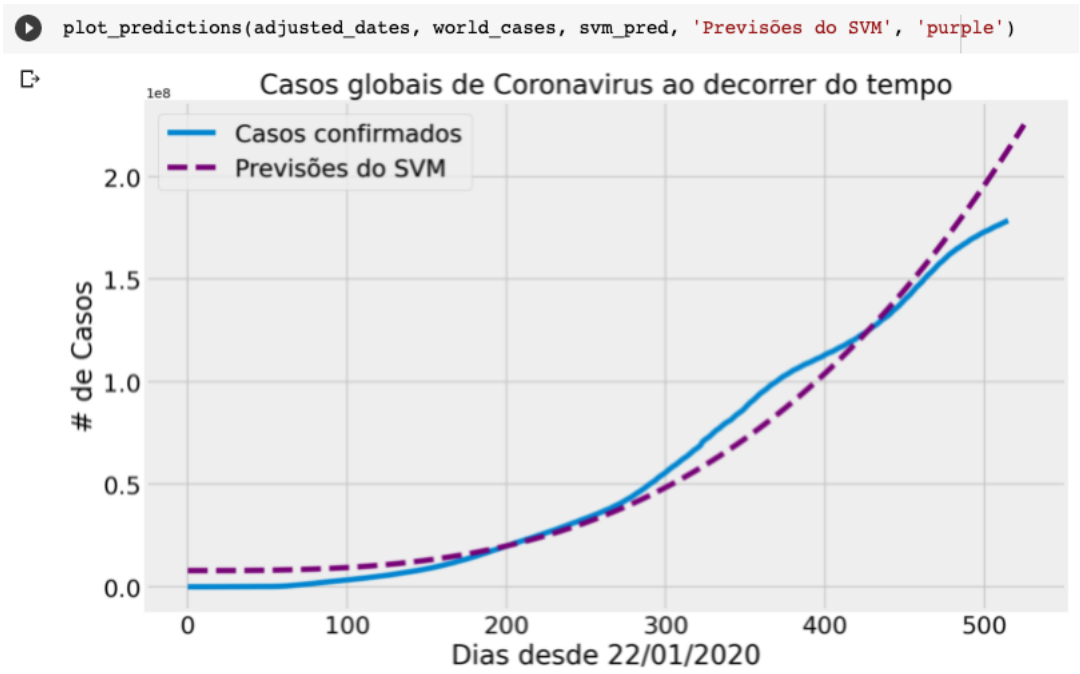
Figura 82 – Função desenvolvida para gerar gráficos dos resultados dos modelos desenvolvidos

```
def plot_predictions(x, y, pred, algo_name, color):
    plt.figure(figsize=(11, 7))
    plt.plot(x, y)
    plt.plot(future_forcast, pred, linestyle='dashed', color=color)
    plt.title('Casos globais de Coronavirus ao decorrer do tempo', size=20)
    plt.xlabel('Dias desde 22/01/2020', size=20)
    plt.ylabel('# de Casos', size=20)
    plt.legend(['Casos confirmados', algo_name], prop={'size': 18})
    plt.xticks(size=18)
    plt.yticks(size=18)
    plt.show()
```

Fonte: O Autor (2021)

Inicialmente utilizou o modelo que implementa o algoritmo de máquina de vetores de suporte para gerar o gráfico. A Figura 83 ilustra o gráfico gerado, observando que as previsões deste modelo estão um tanto quanto distantes na maioria das datas.

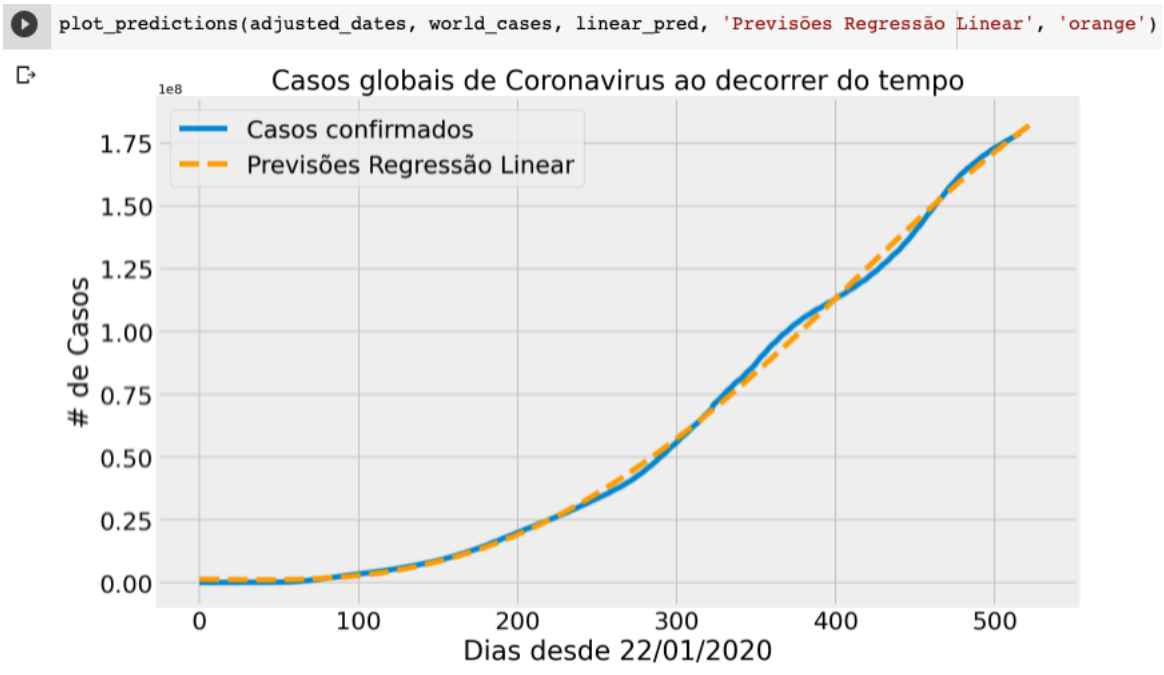
Figura 83 – Comparação das previsões do modelo de máquina de vetores de suporte



Fonte: O Autor (2021)

Posteriormente, foi gerado o mesmo gráfico para o modelo que implementa o algoritmo de regressão linear. O gráfico ilustrado pela Figura 84 demonstra que a curva gerada pelas previsões do modelo estão muito mais próximas em comparação ao algoritmo de máquina de vetores de suporte.

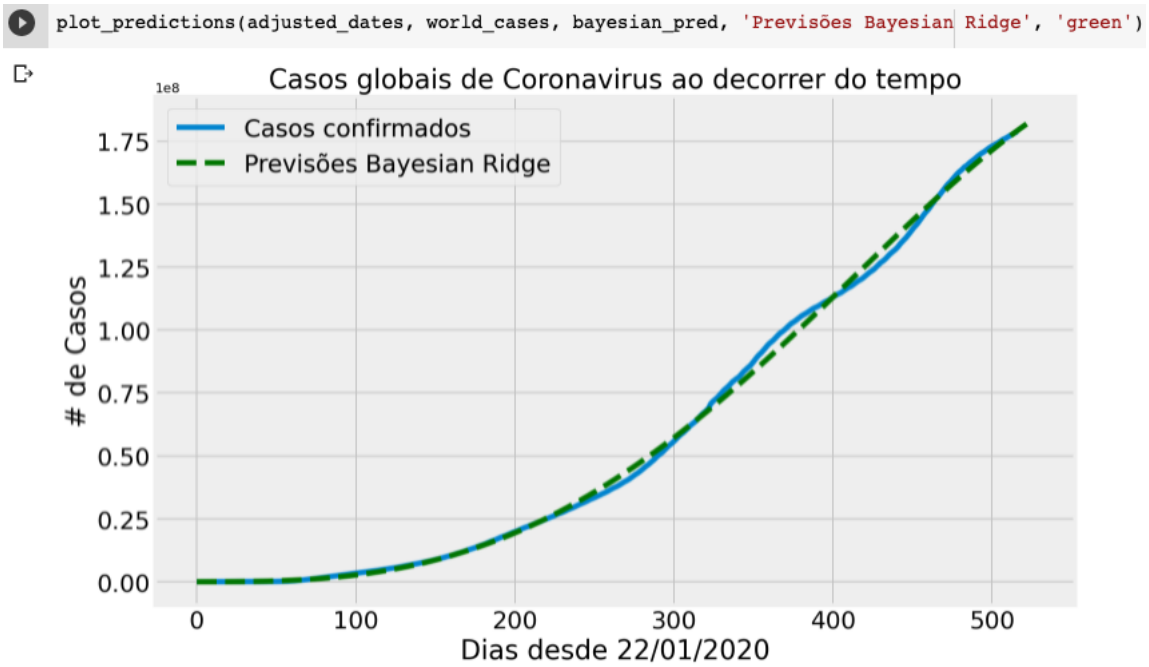
Figura 84 – Comparação das previsões do modelo de regressão linear



Fonte: O Autor (2021)

Por último, foi gerado o mesmo gráfico para o modelo que implementa o algoritmo de *bayesian ridge*, como visto anteriormente nos resultados das métricas, este modelo obteve resultados bastante similares ao modelo de regressão linear. A Figura 85 ilustra o gráfico gerado, possibilitando uma comparação entre as previsões dos modelos.

Figura 85 – Comparação das previsões do modelo de *bayesian ridge*



Fonte: O Autor (2021)

Através das métricas obtidas para cada modelo, somado com as visualizações desenvolvidas, fica claro que os modelos de regressão linear e de *bayesian ridge* obtiveram resultados superiores ao modelo de máquina de vetores de suporte.

3.5 SEGUNDO ESTUDO DE CASO: CONJUNTO DE DADOS IRIS

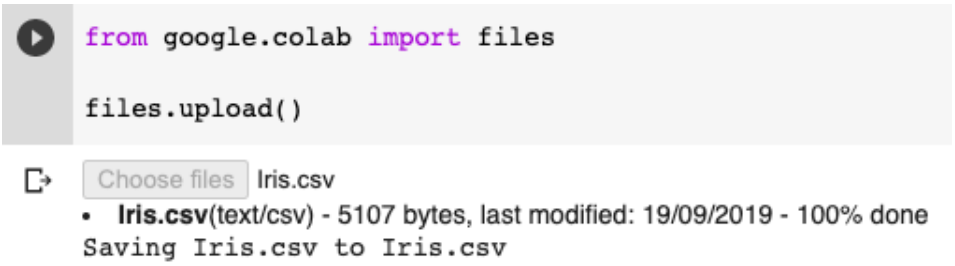
Este conjunto de dados pode ser encontrado no repositório da UCI *Machine Learning*, um banco de dados de problemas de *Machine Learning* que pode ser acessado gratuitamente. O conjunto de dados consiste em três espécies de iris com 50 amostras cada, bem como algumas propriedades sobre cada flor. As espécies são denominadas de Iris Setosa, Iris Versicolor e Iris Virgínica. Cada amostra contém quatro atributos que contribuem para a identificação única como uma das três espécies, sendo elas: o comprimento da sépala, a largura da sépala, o comprimento da pétala e a largura da pétala, além de um identificador único, representado por um número.

3.5.1 Exploração dos dados e aplicação de visualizações

Primeiramente foi feito *download* do conjunto de dados no site do *Kaggle*, o arquivo está no formato *CSV*. Posteriormente, foi realizado o *upload* deste arquivo para um novo ambiente

do *Google Colab*, como ilustra a Figura 86.

Figura 86 – Upload do conjunto de dados relacionado às plantas iris para o *Google Colab*



Fonte: O Autor (2021)

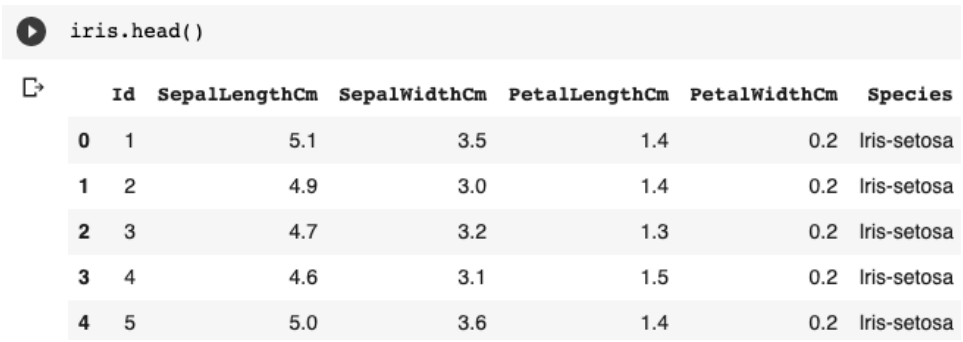
Subsequentemente utilizando a biblioteca *Pandas*, foi realizado o carregamento do arquivo *CSV* e atribui-se os resultados para a variável denominada de *iris*, para isso utilizou-se o método chamado *read_csv*, como a Figura 87 ilustra.

Figura 87 – Carregando o conjunto de dados iris

Fonte: O Autor (2021)

Utilizando o método *head()* proveniente da biblioteca *Pandas* é possível analisar as 5 primeiras amostras presentes no conjunto de dados. A Figura 88 ilustra o resultado. É possível observar as colunas presentes no conjunto de dados e que as mesmas estão em inglês.

Figura 88 – Primeiras 5 amostras do conjunto de dados iris



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Fonte: O Autor (2021)

Em seguida utilizou-se o método *describe()* que possibilita visualizar alguns detalhes estatísticos básicos de um conjunto de dados como a média, os valores máximos e mínimos, a média, etc. A Figura 89 ilustra o resultado obtido, podendo ser confirmado que o conjunto de dados possui 150 amostras.

Figura 89 – Descrição estatística do conjunto de dados iris

```
iris.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Fonte: O Autor (2021)

Utilizando métodos como o *shape()* também é possível identificar o número de linhas e colunas presentes num conjunto de dados. Procurou-se por valores nulos presentes no conjunto de dados, utilizando o método *isnull().sum()*, contrário do conjunto de dados do COVID-19 não foram encontrados nenhum valor nulo. Por último, foi verificado quais os tipos de dados presentes para cada coluna utilizando o método *dtypes*. A Figura 90 ilustra o resultado obtido.

Figura 90 – Descrição geral do conjunto de dados iris

```
print("Número de linhas/Número de colunas do dataset:", iris.shape)
print("\n")
print("Procurando por valores nullos:")
print(iris.isnull().sum())
print("\n")
print("Verificando os tipos de dados de cada coluna:")
print(iris.dtypes)
```

```
Número de linhas/Número de colunas do dataset: (150, 6)


Procurando por valores nullos:
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64


Verificando os tipos de dados de cada coluna:
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

Fonte: O Autor (2021)

3.5.2 Pré-processamento dos dados e aplicação de visualizações

Como observado até o momento, este conjunto de dados contém poucas modificações necessárias antes de ser utilizado, isso é devido ao fato de ser um conjunto de dados amplamente utilizado no meio acadêmico para estudos na área de *Machine Learning*.

Primeiramente foi realizado a tradução das colunas utilizando o método *rename* proveniente da biblioteca *Pandas*, os parâmetros são o nome da coluna atual e o nome desejado para ser renomeado, além disso, também deletou-se a coluna *Id* que representa um identificador único para cada amostra dentro do conjunto de dados que não será utilizado no decorrer do processo. A Figura 91 ilustra o resultado obtido depois de renomear as colunas e de deletar a coluna *Id*.

Figura 91 – Pré-processamento do conjunto de dados iris

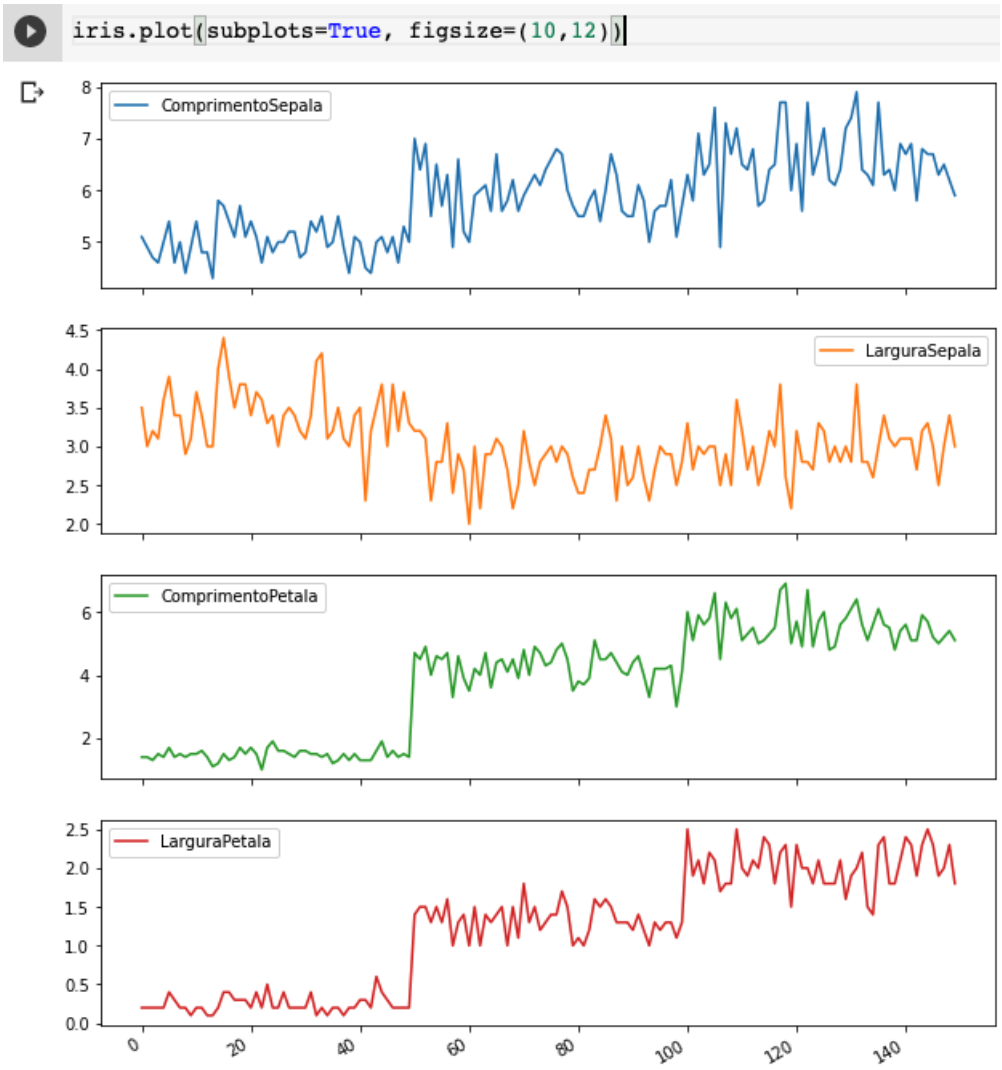
```
iris.rename(columns={'SepalLengthCm': 'ComprimentoSepala',
                     'SepalWidthCm':'LarguraSepala',
                     'PetalLengthCm':'ComprimentoPetala',
                     'PetalWidthCm':'LarguraPetala',
                     'Species':'Especie'
                    }, inplace=True)

iris.drop(["Id"], 1, inplace=True)

iris.head()
```

	ComprimentoSepala	LarguraSepala	ComprimentoPetala	LarguraPetala	Especie
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Fonte: O Autor (2021)

Sucessivamente foi gerado um gráfico utilizando a biblioteca *Pandas*, o gráfico constitui sub-gráficos para cada coluna e seus valores correspondentes no conjunto de dados. A Figura 92 ilustra o resultado obtido. Os gráficos que representam o comprimento da pétala e a largura da pétala são bastante similares, indicando uma alta correlação.

Figura 92 – Gráfico das colunas presentes no conjunto de dados iris



Fonte: O Autor (2021)

A fim de analisar melhor essa possível correlação entre a largura e o comprimento da pétala, foi desenvolvido um trecho de código que cria um gráfico de dispersão. Este conjunto de dados está organizado pelas espécies presentes, ou seja, há 50 amostras de iris-setosa, seguido de 50 amostras da espécie iris versícula e por último 50 amostras da espécie iris virgínica. Visto isso, utilizando o *iloc*, proveniente da biblioteca *Pandas*, é possível escolher o número de linhas e as colunas específicas que se quer utilizar. Com isso foi possível para cada classe especificar uma legenda e uma cor para diferenciar no gráfico. A biblioteca que facilitou a criação deste gráfico é a biblioteca *Ploty*. A Figura 93 ilustra o código desenvolvido.

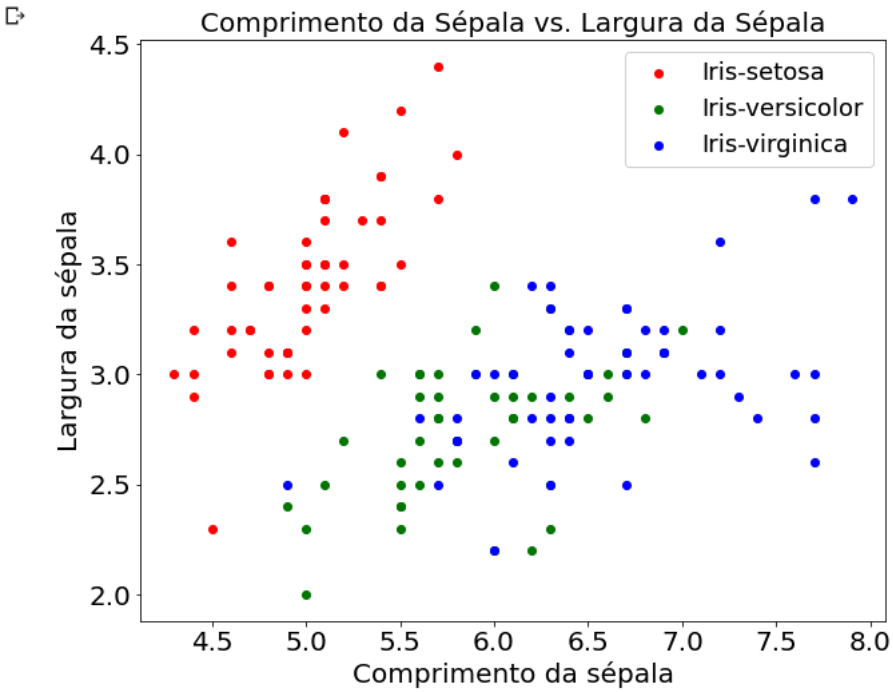
Figura 93 – Código desenvolvido para gerar o gráfico de dispersão

```
plt.figure(4, figsize=(10, 8))

plt.scatter(iris.iloc[:50, 0], iris.iloc[:50, 1], c='r', label='Iris-setosa')

plt.scatter(iris.iloc[50:100, 0], iris.iloc[50:100, 1], c='g',label='Iris-versicolor')

plt.scatter(iris.iloc[100:, 0], iris.iloc[100:, 1], c='b',label='Iris-virginica')

plt.xlabel('Comprimento da sépala',fontsize=20)
plt.ylabel('Largura da sépala',fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.title('Comprimento da Sépala vs. Largura da Sépala',fontsize=20)
plt.legend(prop={'size': 18})
plt.show()
```

Fonte: O Autor (2021)

Analisando o gráfico gerado que é ilustrado pela Figura 94, é possível observar que fica muito aparente a correlação das flores de iris setosa entre o seu comprimento e a largura das sépalas, tendo os pontos no gráfico bem densos. Por outro lado, há menos correlação entre a iris versicolor e a iris virgínica. Os pontos de dados das espécies versicolor e virgínica estão mais espalhados do que os da espécie setosa.

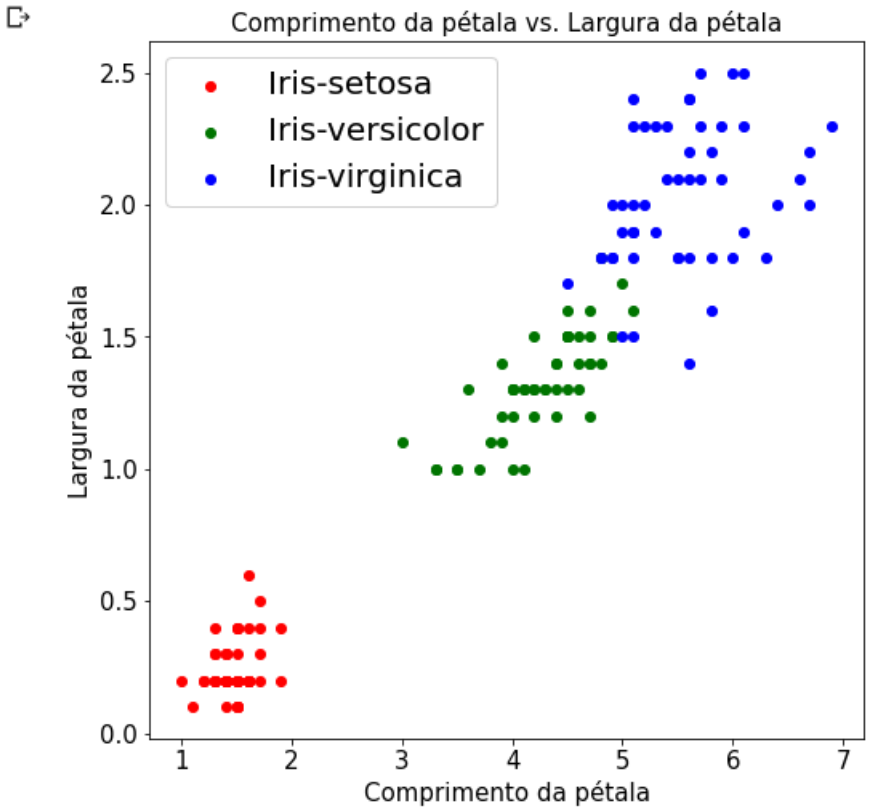
Figura 94 – Gráfico de dispersão do comprimento e largura da sépala



Comprimento da Sépala vs. Largura da sépala. Fonte: O Autor (2021)

O mesmo gráfico foi gerado, porém agora colocando em evidência a largura e o comprimento da pétala. A Figura 95 ilustra o gráfico obtido. Mesmo quando se trata do comprimento e largura das pétalas, o gráfico indica uma forte correlação para as flores de espécie setosa que estão densamente agrupadas.

Figura 95 – Gráfico de dispersão do comprimento e largura da pétala



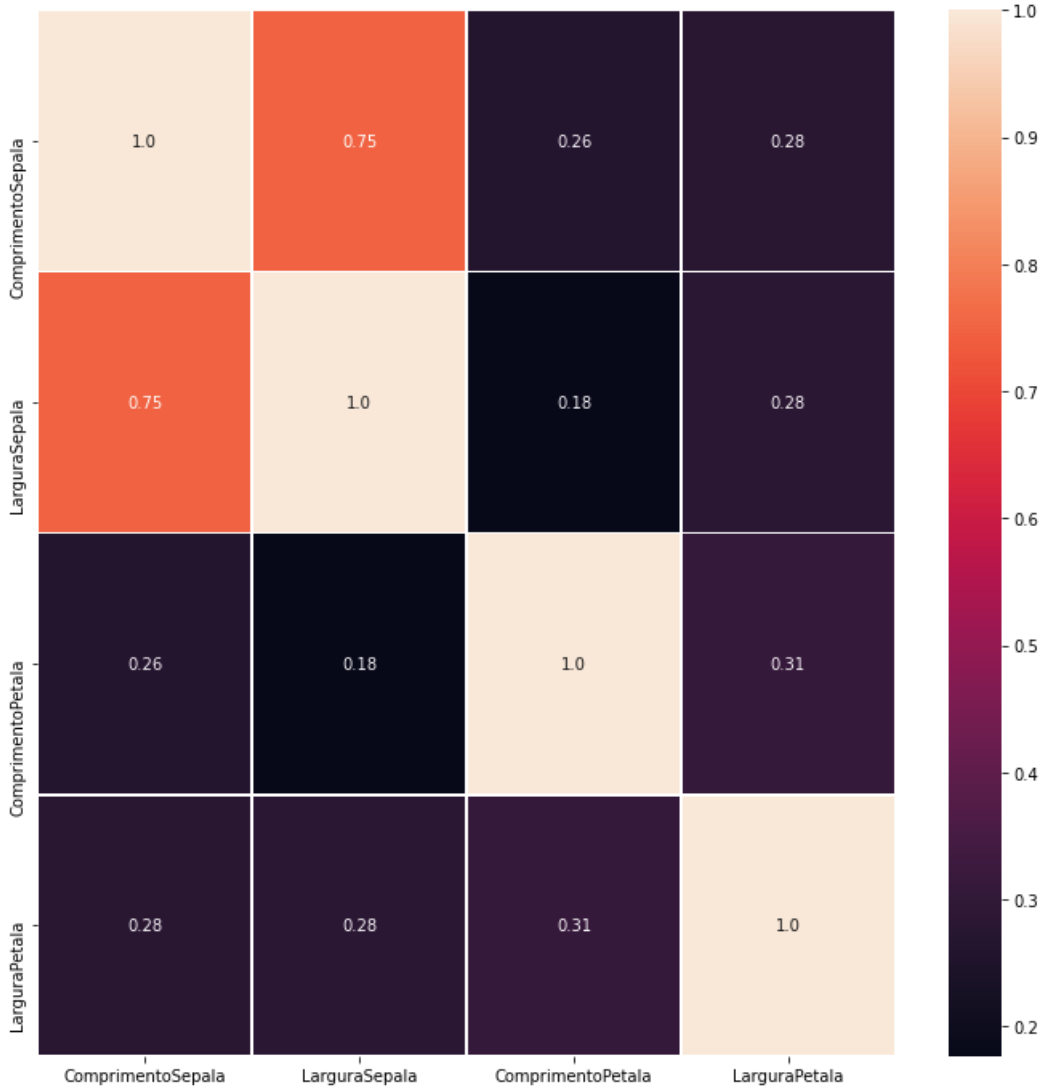
Comprimento da pétala vs. Largura da pétala. Fonte: O Autor (2021)

Além do gráfico de dispersão, a fim de analisar as correlações entre os atributos das amostras, um gráfico de calor foi gerado para cada espécie separadamente. Para isso foi utilizado o método *corr()* proveniente da biblioteca *Pandas*, que calcula a correlação entre os atributos. Com os valores da correlação calculados foi utilizado a biblioteca *Seaborn* parar gerar os gráficos de calor. As Figuras 96, 97 e 98 ilustram os resultados. A partir dos gráficos gerados, é possível analisar que a correlação entre o comprimento e a largura da pétala das espécies setosa e virgínica é de 0,33 e 0,32, respectivamente, e que, para a espécie versicolor é de 0,78.

Figura 96 – Gráfico de calor da espécie setosa

```
setosa_corr = iris.iloc[:50,:].corr() #setosa
f,ax = plt.subplots(figsize=(12, 12))
sns.heatmap(setosa_corr, annot=True,linewidth=.6, fmt=".2",ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c717a0c90>
```

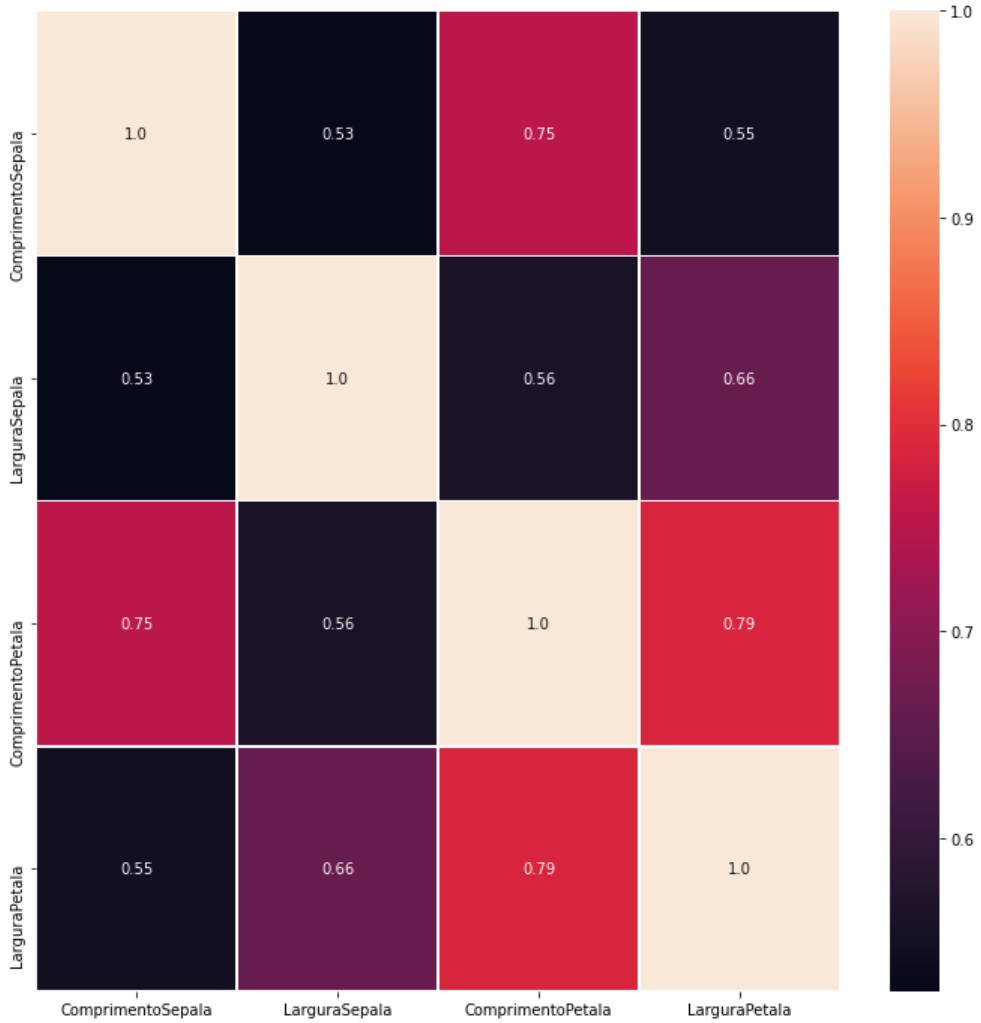


Fonte: O Autor (2021)

Figura 97 – Gráfico de calor da espécie versicolor

```
f,ax = plt.subplots(figsize=(12, 12))
sns.heatmap(versicolor_corr, annot=True,linewidth=.6, fmt=".2",ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c71753e50>
```

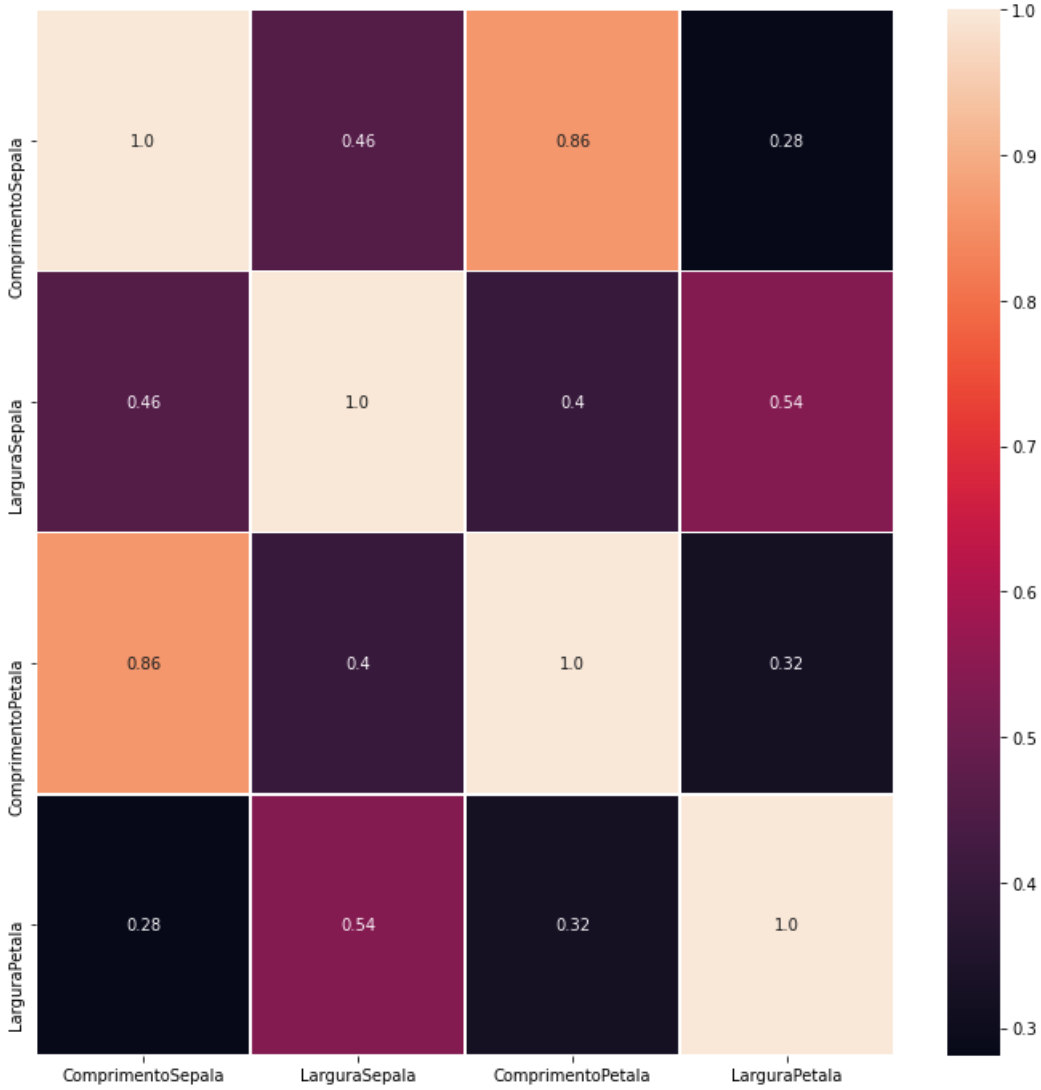


Fonte: O Autor (2021)

Figura 98 – Gráfico de calor da espécie virgínica

```
virginica_corr = iris.iloc[100:,:].corr() #virginica
f,ax = plt.subplots(figsize=(12, 12))
sns.heatmap(virginica_corr, annot=True,linewidth=.6, fmt=".2", ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1c708c2450>
```

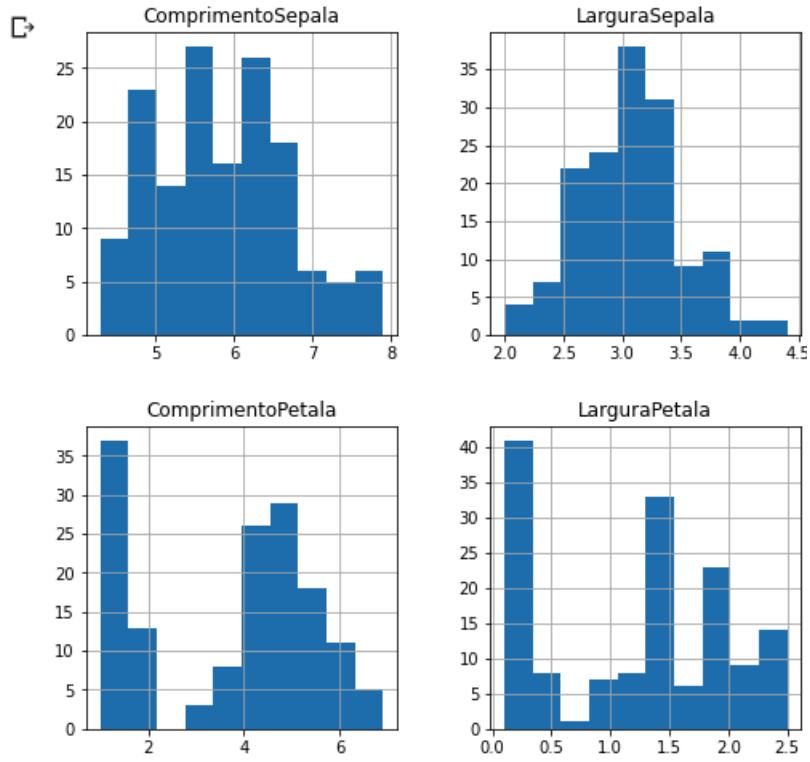


Fonte: O Autor (2021)

A fim de analisar a distribuição dos atributos presentes no conjunto de dados foram gerados histogramas para cada atributo individualmente. A Figura 99 ilustra o resultado obtido, podendo ser observado que o comprimento da pétala, a largura da pétala e o comprimento da sépala mostram uma distribuição unimodal, enquanto a largura da sépala mostra um tipo de distribuição gaussiana.

Figura 99 – Histograma dos atributos presentes no conjunto de dados iris

```
fig = plt.figure(figsize = (8,8))
ax = fig.gca()
iris.hist(ax=ax)
plt.show()
```



Fonte: O Autor (2021)

3.5.3 Aplicação de algoritmos de *Machine Learning* e aplicação de visualizações

Com os dados devidamente preparados, os algoritmos escolhidos a serem implementados foram o *k-nearest neighbors* (KNN) e máquina de vetores de suporte. Estes algoritmos utilizam a técnica de aprendizado supervisionado e resolvem problemas de classificação, tendo em vista que os dados a serem previstos são do tipo categóricos, neste caso a classe das amostras é a espécie das flores.

Primeiramente foram importadas as bibliotecas e os módulos necessários, como os módulos que implementam os algoritmos que serão usados, o módulo *train_test_split* que divide o conjunto de dados em conjunto de treinamento e conjunto de testes, e a biblioteca *numpy* que auxilia nas operações matemáticas realizadas em matrizes. A Figura 100 ilustra o código utilizado.

Figura 100 – Bibliotecas e módulos importados para auxiliar no desenvolvimento dos modelos

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import numpy as np
```

Fonte: O Autor (2021)

Posteriormente, dividiu-se o conjunto de dados em dados de treinamento e dados de teste com ajuda do módulo *train_test_split*. As variáveis de treinamento foram atribuídas para duas variáveis, sendo elas a *train_data* que possui os valores das colunas e a variável *train_label* possui a respectiva classe. Já as variáveis de teste foram atribuídas para outras duas variáveis *test_data* possuindo os valores das colunas e *test_label* que possui as respectivas classes.

Figura 101 – Código utilizado para dividir o conjunto de dados em dados de treinamento e testes

```
train_data, test_data, train_label, test_label
            = train_test_split(iris.iloc[:,:3], iris.iloc[:,4], test_size=0.2, random_state=42)
```

Fonte: O Autor (2021)

Após separar os dados de treinamento e dados de testes, será implementado primeiramente o algoritmo KNN. O algoritmo KNN possui como parâmetro a variável *n_neighbors* que representa um número, esse número diz respeito ao número de amostras próximas que o algoritmo irá considerar para tomar uma decisão de classificação. Para escolher o melhor valor a ser utilizado para este conjunto de dados foi desenvolvido um trecho código que testa o desempenho do modelo para os valores entre 1 a 9. A Figura 102 ilustra o código. Para cada valor foi criado um modelo, treinado o modelo com os dados de treinamento e posteriormente testado o modelo nos dados de teste, salvando os resultados provenientes de cada modelo.

Figura 102 – Código utilizado para encontrar o melhor parâmetro para o algoritmo KNN

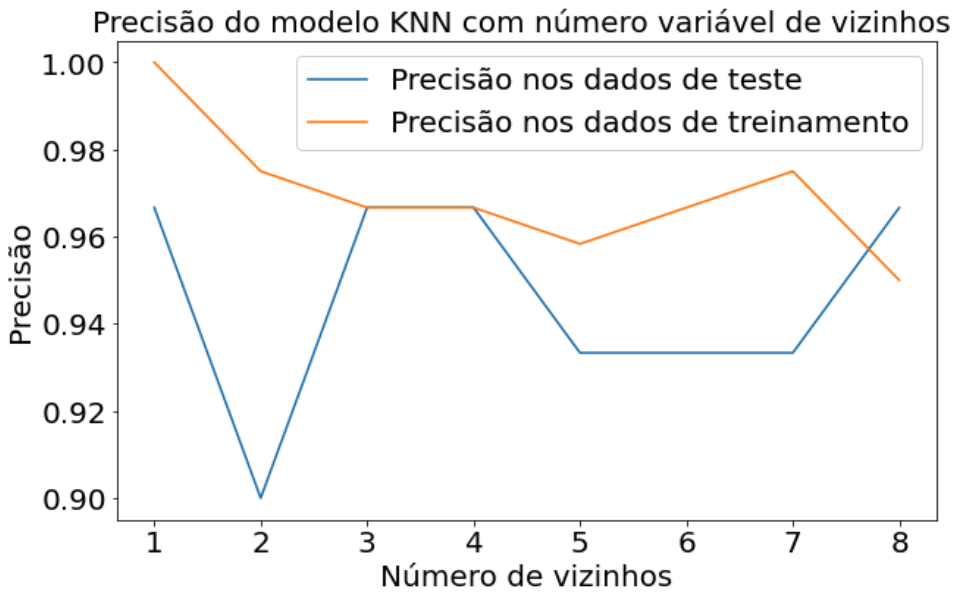
```
neighbors = np.arange(1,9)
train_accuracy =np.zeros(len(neighbors))
test_accuracy = np.zeros(len(neighbors))

for i,k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(train_data, train_label)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(train_data, train_label)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(test_data, test_label)
```

Fonte: O Autor (2021)

Com os dados dos modelos desenvolvidos anteriormente criou-se um gráfico com o intuito de encontrar o melhor parâmetro encontrado. A Figura 103 ilustra o resultado, podendo ser observado que o valor que obteve maiores resultados de precisão nos dados de teste e de treinamento foi o número 3.

Figura 103 – Gráfico que mostra a comparação de precisão entre modelos com diferentes parâmetros de número de vizinhos

```
plt.figure(figsize=(10,6))
plt.title('Precisão do modelo KNN com número variável de vizinhos ',fontsize=20)
plt.plot(neighbors, test_accuracy, label='Precisão nos dados de teste')
plt.plot(neighbors, train_accuracy, label='Precisão nos dados de treinamento')
plt.legend(prop={'size': 20})
plt.xlabel('Número de vizinhos',fontsize=20)
plt.ylabel('Precisão',fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```



Precisão do modelo KNN com número variável de vizinhos. Fonte: O Autor (2021)

Após encontrar o parâmetro que possui o melhor desempenho criou-se novamente o modelo utilizando como parâmetro o valor 3. Foi realizado o treinamento e novamente os testes, atribuindo os resultados para a variável *knn*. A Figura 104 ilustra o código.

Figura 104 – Código desenvolvido para criar o modelo KNN

```
knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(train_data, train_label)

train_accuracy = knn.score(train_data, train_label)

test_accuracy = knn.score(test_data, test_label)
```

Fonte: O Autor (2021)

Para criar o modelo que implementa o algoritmo de máquina de vetores de suporte foi utilizado o mesmo processo, a única diferença foi na utilização dos módulos. A Figura 105 ilustra o trecho de código desenvolvido para criar o modelo de máquina de vetores de suporte.

Figura 105 – Código desenvolvido para criar o modelo de máquina de vetores de suporte

```
svm = SVC(kernel='rbf', random_state=0, gamma=.10, C=1.0)

svm.fit(train_data, train_label)

svm_train_accuracy = knn.score(train_data, train_label)

svm_test_accuracy = knn.score(test_data, test_label)
```

Fonte: O Autor (2021)

3.5.4 Definição de critérios de avaliação para o modelo e aplicação de visualizações

Os critérios de avaliação deste conjunto de dados são diferentes do conjunto de dados do COVID-19, isso é devido a diferença entre os problemas que cada conjunto de dados possibilita resolver. Métricas como a precisão, *recall*, *f1 score*, *support* foram calculadas para medir o desempenho de cada modelo, além da criação de uma matriz de confusão e de um gráfico mostrando os erros de previsão de classe para cada modelo. Para isso foi utilizado uma biblioteca denominada de *Yellowbrick*.

A biblioteca *Yellowbrick* possui um pacote de ferramentas de visualização e diagnóstico que possibilita a seleção mais rápida de modelos. A Figura 106 ilustra a importação dos módulos da biblioteca.

Figura 106 – Módulos importados para realizar as métricas

```
from yellowbrick.target import ClassBalance
from yellowbrick.classifier import ROCAUC
from yellowbrick.classifier import PrecisionRecallCurve
from yellowbrick.classifier import ClassificationReport
from yellowbrick.classifier import ClassPredictionError
```

Fonte: O Autor (2021)

Antes de tudo foi desenvolvido um relatório de classificação. Esse relatório contém métricas como a precisão, o *recall* que indica a capacidade do modelo de encontrar corretamente todas as instâncias positivas, o *f1 score* que é uma média harmônica ponderada entre a precisão e o *recall*, por último o *support* que representa o número de ocorrências de determinada classe no conjunto de dados. Para isso, foi utilizado o módulo *ClassificationReport*, que utiliza como parâmetro o modelo e as classes presentes, a Figura 107 ilustra o código desenvolvido.

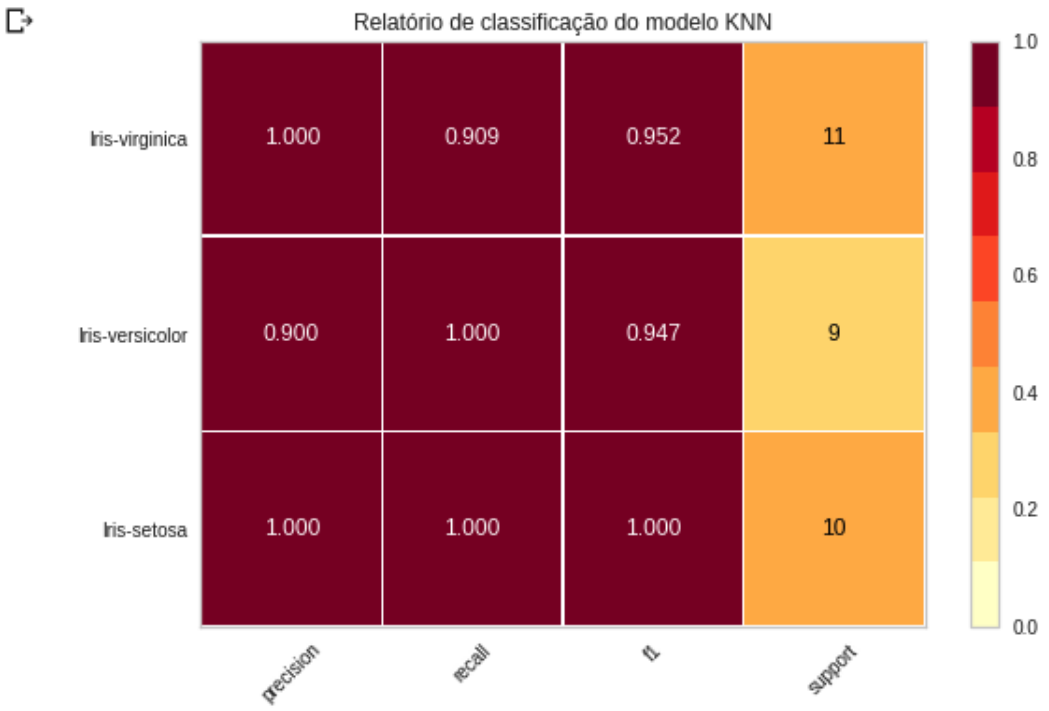
Figura 107 – Código desenvolvido para gerar o relatório de classificação

```
classification_report = ClassificationReport(knn, support=True)

classification_report.fit(train_data, train_label)
classification_report.score(test_data, test_label)
classification_report.title = "Relatório de classificação do modelo KNN"
classification_report.finalize()
```

Fonte: O Autor (2021)

Primeiramente foi gerado o relatório de classificação para o modelo KNN. A Figura 108 ilustra o relatório, analisando o relatório é possível analisar que o modelo teve apenas um falso positivo, identificou uma iris-versicolor como iris-virgínica, isso afetou a precisão de acerto na classe iris-versicolor, além de ter afetado o *recall* da classe iris-virgínica e é possível analisar o *support* que a classe iris-virgínica possui 11 previsões e a iris-versicolor apenas 9.

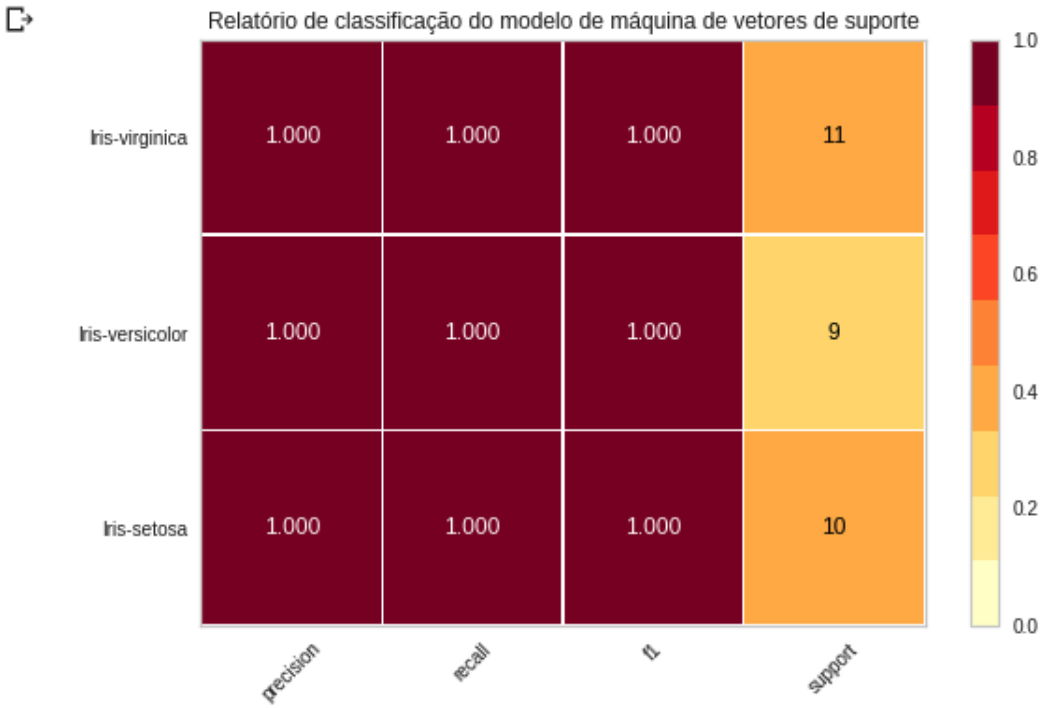
Figura 108 – Relatório de classificação do modelo KNN



Relatório de classificação do modelo KNN. Fonte: O Autor (2021)

O segundo relatório foi gerado para o modelo de máquina de vetores de suporte. A Figura 109 ilustra o relatório. Este modelo teve um relatório perfeito, sem erros nos dados de teste.

Figura 109 – Relatório de classificação do modelo de máquina de vetores de suporte



Relatório de classificação do modelo de máquina de vetores de suporte. Fonte: O Autor (2021)

Após foi criado uma matriz de confusão para cada um dos modelos, a matriz de confusão mostra como cada uma das classes previstas nos valores de teste se compara às suas classes reais. Para isso o módulo *ConfusionMatrix* foi utilizado, tendo como parâmetro o modelo a ser analisado, a Figura 110 ilustra o código desenvolvido.

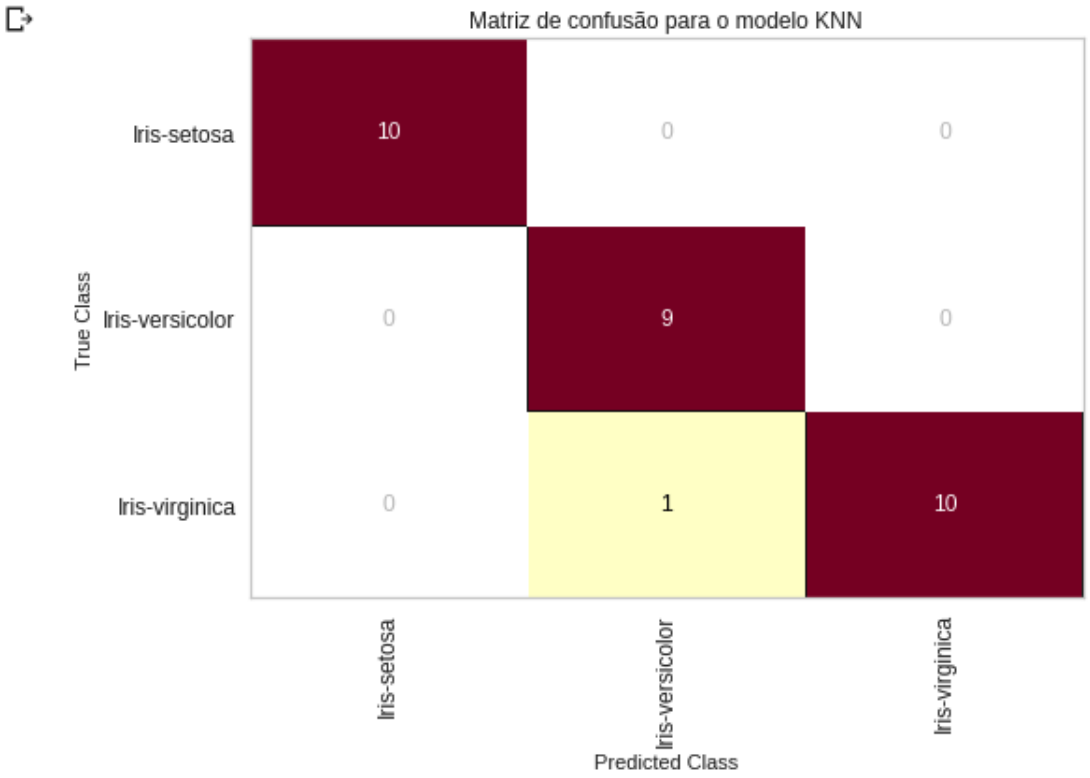
Figura 110 – Código desenvolvido para gerar a matriz de confusão

```
confusion_matrix = ConfusionMatrix(knn)

confusion_matrix.fit(train_data, train_label)
confusion_matrix.score(test_data, test_label)
confusion_matrix.title = "Matriz de confusão para o modelo KNN"
confusion_matrix.finalize()
```

Fonte: O Autor (2021)

Inicialmente foi gerado a matriz de confusão para o modelo KNN, é possível analisar novamente pela Figura 111 que o modelo previu a classe iris-versicolor como uma iris-virgínica.

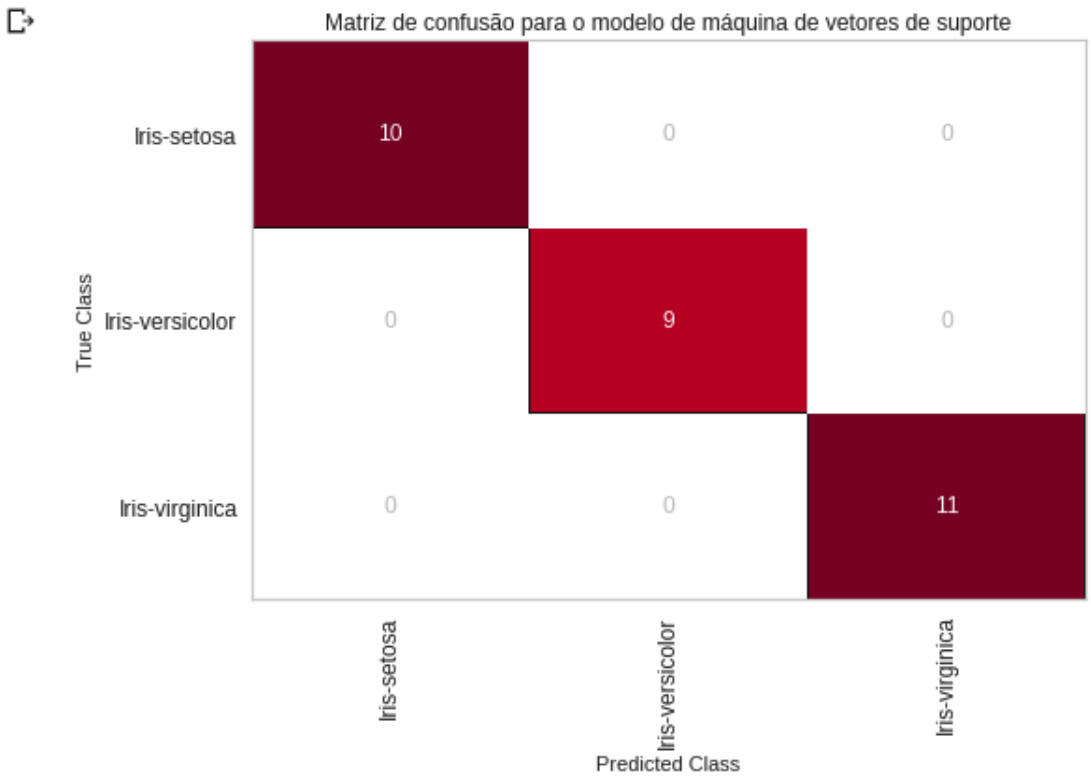
Figura 111 – Matriz de confusão para o modelo KNN



Matriz de confusão para o modelo KNN. Fonte: O Autor (2021)

Posteriormente foi gerado a matriz de confusão para o modelo de máquina de vetores de suporte, como o modelo não teve erros não haverá dados a não ser nas diagonais, a Figura 112 ilustra a matriz de confusão gerada.

Figura 112 – Matriz de confusão para o modelo de máquina de vetores de suporte



Matriz de confusão para o modelo de máquina de vetores de suporte. Fonte: O Autor (2021)

Por último foi criado um gráfico que mostra os erros de previsão de classe para cada modelo, ilustrando para cada classe disponível no conjunto de dados qual classe foi dita como acerto. Para isso foi utilizado o módulo *ClassPredictionError*, que utiliza como parâmetro um modelo. A Figura 113 ilustra o código desenvolvido.

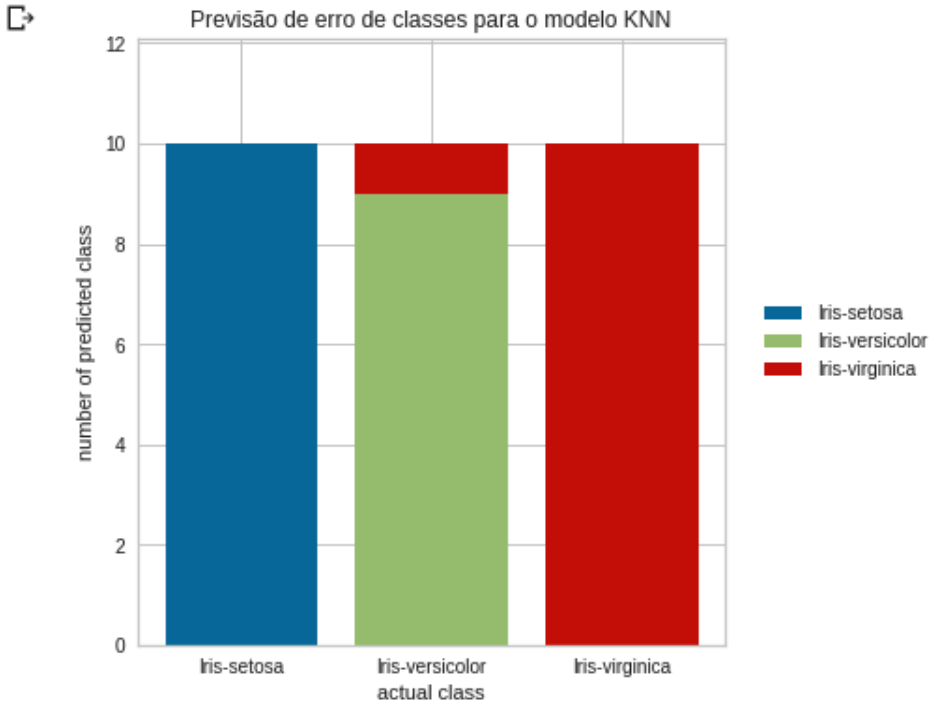
Figura 113 – Código desenvolvido para criar gráficos de erro de previsão de classe

```
class_prediction_report = ClassPredictionError(knn)

class_prediction_report.fit(train_data, train_label)
class_prediction_report.score(test_data, test_label)
class_prediction_report.title = "Previsão de erro de classes para o modelo KNN"
class_prediction_report.finalize()
```

Fonte: O Autor (2021)

No gráfico de erro de previsão de classe para o modelo KNN é possível visualizar erros do classificador. A Figura 114 ilustra o resultado, no caso deste modelo, é possível analisar que uma amostra iris-virgínica foi identificada como iris-versicolor.

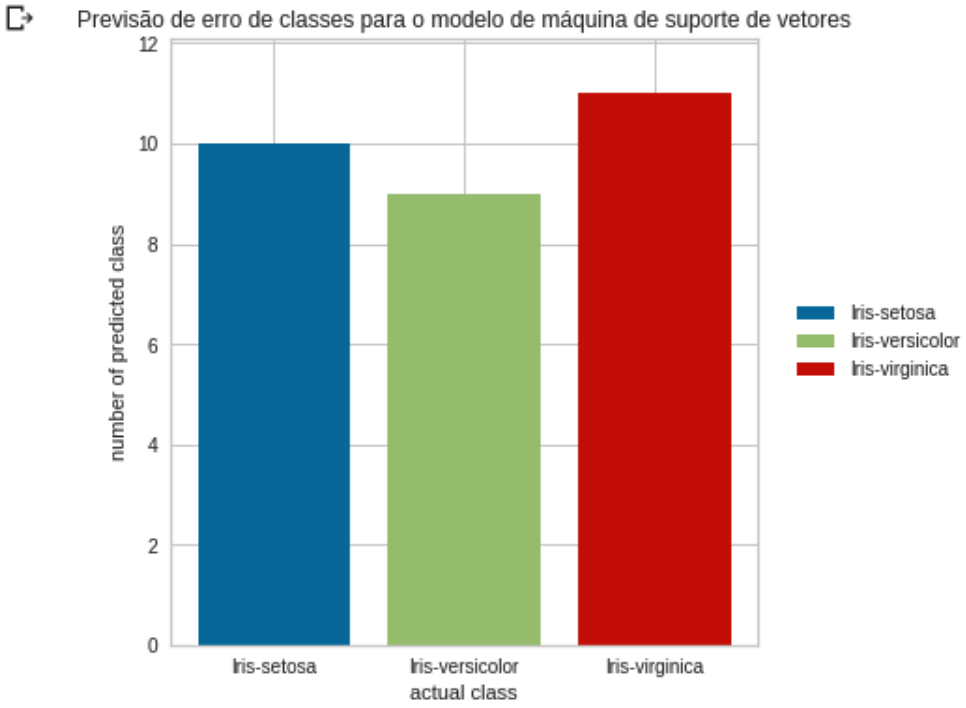
Figura 114 – Previsão de erro de classes para o modelo KNN



Previsão de erro de classes para o modelo KNN. Fonte: O Autor (2021)

Posteriormente foi gerado o mesmo gráfico para o modelo de máquina de vetores de suporte, que como não teve nenhum erro nos dados de teste, as barras no gráfico serão de uma única cor. A Figura 115 ilustra o resultado.

Figura 115 – Previsão de erro de classes para o modelo de máquina de vetores de suporte



Previsão de erro de classes para o modelo de máquina de vetores de suporte. Fonte: O Autor (2021)

3.5.5 Seleção do modelo de melhor desempenho e visualizações para justificar a escolha

Além das dos resultados obtidos calculando as métricas, para ajudar na escolha do modelo final foi realizada a comparação dos limites de decisão.

O limite de decisão de um modelo nada mais é que o espaço que cada classe ocupa. O modelo irá classificar todos os pontos de um lado da fronteira de decisão como pertencentes a uma classe e os pontos restantes do outro lado como pertencentes à uma outra classe. A fronteira de decisão é a região de um espaço em que a predição de saída do modelo é ambígua.

Para desenvolver o limite de decisão dos modelos foi utilizado a biblioteca denominada de *mlextend* [18], que contém funções úteis para as tarefas de ciência de dados. Foi utilizado uma versão modificada de um trecho de código que está disponível nos exemplos desta biblioteca [19].

Primeiramente foram criados mais dois modelos para realizar a comparação em múltiplos modelos, os modelos utilizam os algoritmos de *Random Forest* e *Naive Bayes*, ambos classificadores. A Figura 116 ilustra o código.

[18] http://rasbt.github.io/mlxtend/
[19] http://rasbt.github.io/mlxtend/user_guide/plotting/plot_decision_regions

Figura 116 – Código desenvolvido para criar os dois modelos extras para comparação

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn import datasets
import numpy as np

clf1 = RandomForestClassifier(random_state=1, n_estimators=100)
clf2 = GaussianNB()

# Loading some example data
iris = datasets.load_iris()
X = iris.data[:, [0,2]]
y = iris.target
```

Fonte: O Autor (2021)

Posteriormente, importou-se o módulo *plot_decision_regions* da biblioteca *mlxtend* que é responsável por gerar os gráficos contendo o limite de decisão. Para cada modelo foi realizado o treinamento e gerado um gráfico do limite de decisão. A Figura 117 ilustra o código desenvolvido.

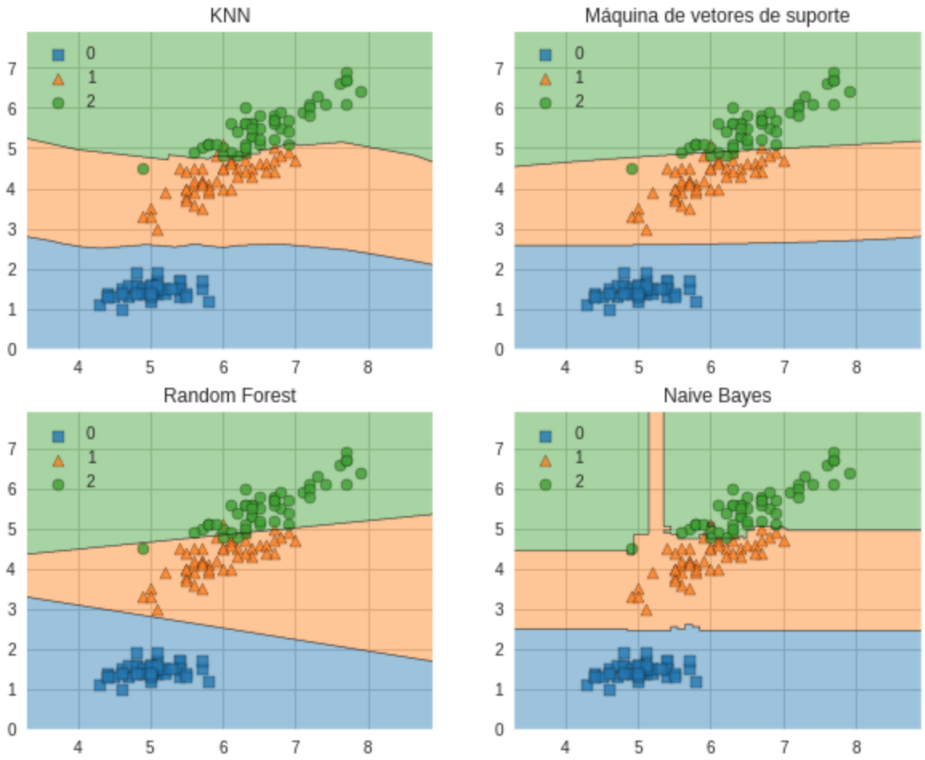
Figura 117 – Código desenvolvido gerar os limites de decisão dos modelos

```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
import matplotlib.gridspec as gridspec
import itertools
gs = gridspec.GridSpec(2, 2)

fig = plt.figure(figsize=(10,8))

labels = ['KNN', 'Máquina de vetores de suporte', 'Random Forest', 'Naive Bayes']
for clf, lab, grd in zip([knn, svm, clf1, clf2],
                         labels,
                         itertools.product([0, 1], repeat=2)):

    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y, clf=clf, legend=2)
    plt.title(lab)

plt.show()
```

Fonte: O Autor (2021)

O resultado obtido é demonstrado pela Figura 118. Onde é possível analisar que para cada modelo tem-se um limite de decisão diferente. A legenda está em forma numérica por conta de ser um requisito da biblioteca, o quadrado azul representa a iris-setosa, o triângulo laranja representa a iris-versicolor e a bolinha verde representa a iris-virgínica.

Figura 118 – Limites de decisão dos modelos



Fonte: O Autor (2021)

Os limites de decisão são diagnóstico popular para entender as decisões tomadas por um modelo de classificação. O gráfico mostra como o modelo de *Machine Learning* treinado prevê os dados de entrada. Mostrando ser uma ferramenta poderosa para entender como um determinado modelo "vê" a tarefa de previsão e como ele decidiu dividir o espaço do recurso de entrada por rótulo de cada classe.

Com base na Figura 118 é possível identificar, de uma forma visual, que o melhor modelo entre os 4 presentes é o modelo que implementa o algoritmo *Naive Bayes*. Pois, aparentemente é o único que teve seus limites de decisão certeiros.

4 CONCLUSÃO

Neste capítulo apresenta-se uma síntese das atividades desenvolvidas durante o desenvolvimento deste trabalho, descreve também os resultados analisados e ideias para trabalhos futuros.

4.1 SÍNTESE DO TRABALHO

No presente trabalho foram abordados conceitos de *Machine Learning* e Visualização de Dados. A partir desse estudo foi possível evidenciar a complexidade de um processo de *Machine Learning* e a importância de cada etapa durante o processo. Além disso, cabe salientar que a visualização de dados é um recurso que desempenha um papel fundamental em todas as etapas do processo de *Machine Learning*. Ela facilita a descoberta de padrões, além de oferecer diferentes maneiras de exibir os resultados de forma mais amigável e compreensível.

Adicionalmente, foi realizado um estudo de ferramentas ou bibliotecas que pudessem ser utilizadas para implementar um processo de *Machine Learning*, sendo possível produzir visualizações durante o processo. Com isso, foram aplicados os recursos de visualização de dados, obtidos a partir das ferramentas de software, em dois conjuntos de dados identificando o potencial destas ferramentas.

Avaliando-se os objetivos específicos deste estudo, considera-se relevante apontar que as etapas do estudo de caso contribuíram para que eles pudessem ser desenvolvidos e alcançados. A primeira parte do estudo de caso consistiu em analisar o conjunto de dados sobre a contaminação por Coronavírus no mundo. Por se tratar de uma pandemia ainda presente em todo mundo, os dados são atualizados diariamente. Foi possível então acompanhar tanto o processo de aprendizagem e predição diariamente, quando testar os recursos de visualização sobre um cenário em tempo real. O segundo conjunto de dados selecionado foi o Iris, que mesmo sendo bastante explorado em estudos relacionados, nos permite compreender os processos e recursos de visualização sob circunstâncias conhecidas. Sendo o foco deste trabalho os recursos de visualização, os dados usados constituem elemento secundário, servindo para demonstrar o potencial dos componentes visuais testados.

Em relação ao objetivo sobre estabelecer relações entre modelos de visualizações e suas aplicações no processo de *Machine Learning*, identificou-se que em especial as bibliotecas *Yellowbrick*, *Seaborn*, *Ploty*, *MatplotLib* e *Pandas* estão bem estruturadas e auxiliam nas diversas etapas. As tabelas a seguir ilustram os principais recursos de visualização empregados neste trabalho em cada uma das etapas de *Machine Learning*.

A primeira tabela diz respeito a etapa de exploração de dados, onde os métodos disponí-

veis para visualizar o conjunto de dados proveram uma descrição estatística geral, além de terem ajudado a identificar valores nulos e os tipos presentes para cada coluno. Os métodos utilizados e suas respectivas bibliotecas são ilustrados pela Figura 119.

Figura 119 – Tabela referente a etapa de exploração dos dados

Etapa de exploração dos dados	
Visualizações recomendadas	**Biblioteca(s)**
.head() - ilustra os cinco primeiros valores presentes no conjunto de dados	Pandas
.isnull().sum() - demonstra a quantidade de valores nulos para cada coluna do conjunto de dados	Pandas
.dtypes - demonstra os tipos de dados contidos no conjunto de dados	Pandas
.describe() - ilustra detalhes estatísticos básicos do um conjunto de dados como a média, os valores máximos e mínimos, etc.	Pandas

Fonte: O Autor (2021)

A segunda tabela é ilustrada pela Figura 120 e se refere a etapa de pré-processamento dos dados. Nesta etapa, com o conjunto de dados devidamente preparado, foi possível gerar visualizações que possibilitaram uma comparação dos atributos presentes, o encontro de correlações dos atributos do conjunto de dados, além de ser possível visualizar os dados separadamente para cada país presente.

Figura 120 – Tabela referente a etapa de pré-processamento dos dados

Pré-processamento dos dados	
Visualizações recomendadas	**Biblioteca(s)**
.plot(subplots=True) - para cada coluna disponível no conjunto de dados um gráfico será gerado, possibilitando a análise das colunas separadamente ou a realização de uma comparação entre as outras colunas.	Pandas
sns.barplot() - gráfico de barra, útil para comparar categorias e medir valores entre os dados presentes.	Seaborn
plt.pie() - gráfico de pizza, útil para comparar categorias e medir proporções entre os dados presentes.	Ploty
sns.heatmap() - gráfico de calor, extremamente útil para encontrar correlações entre as propriedades do conjunto de dados.	Seaborn
plt.line() - gráfico linear iterativo, este gráfico permite uma análise detalhada para por exemplo cada data disponível em um conjunto de dados, podendo ainda ser colocado em evidência múltiplas colunas.	Ploty
px.choropleth() - gráfico que representa dados em relação a algum país, ou até mesmo podendo ser separado por estados ou cidades, útil para realizar comparações e analisar a evolução, decadência, variância de dados em países/estados/cidades.	Plotly.Express
plt.scatter() - gráfico de dispersão, possibilita a identificação de correlações entre os dados presentes no conjunto de dados.	Ploty
plt.hist() - histograma, pode ajudar a detectar dados incomuns (outliers) ou lacunas nos dados.	Ploty

Fonte: O Autor (2021)

A próxima tabela se refere a etapa de definição de critérios de avaliação para os modelos desenvolvidos. Esta tabela é ilustrada pela Figura 121 e tem como principais métodos presentes métodos derivados da biblioteca *Yellowbrick*, como por exemplo o *ClassificationReport* que demonstra um relatório de classificação de um modelo mostrando diversas métricas e também os métodos *ConfusionMatrix* e *ClassPredictionError* que ajudam a identificar o desempenho do modelo.

Figura 121 – Tabela referente a etapa de definições de critérios de avalição

Definição de critérios de avaliação para o modelo	
Visualizações recomendadas	**Biblioteca(s)**
plt.plot() - comparação entre dados previstos por um modelo e dados de treinamento	Ploty
ClassificationReport - relatório de classificação, contendo diversas métricas como a precisão, o recall e o f1 score	Yellowbrick
ConfusionMatrix - matriz de confusão que mostra como cada uma das classes previstas pelo modelo se compara às classes reais	Yellowbrick
ClassPredictionError - gráficos de erro de previsão de classe, possibilitando identificar de uma maneira visual qual classe foi prevista erroneamente	Yellowbrick

Fonte: O Autor (2021)

Por fim, a quarta tabela é ilustrada pela Figura 122. Esta tabela se refere a última etapa do processo de *Machine Learning*, a seleção do modelo de melhor desempenho, nos estudos de caso foram usados dois diferentes recursos, visto que os modelos tinham finalidade diferentes (regressão e classificação). Para os modelos criados para o conjunto de dados relacionado ao COVID-19 foi utilizado um simples gráfico linear que possibilitou a comparação entre os dados reais e os resultados previstos pelo modelo. Já para os modelos criados para o conjunto de dados iris, foram criados os limites de decisões dos modelos, podendo visualizar como o modelo identificou cada classe no conjunto de dados.

Figura 122 – Tabela referente a etapa de seleção do modelo de melhor desempenho

Seleção do modelo de melhor desempenho	
Visualizações recomendadas	**Biblioteca(s)**
plot_decision_regions() - cria gráficos representando o limite de decisão de um modelo, que facilita o entendimento das decisões tomadas por um modelo de classificação. Além disso, o gráfico mostra como um modelo de *Machine Learning* treinado prevê os dados de entrada.	mlxtend
plt.plot() - gráfico linear utilizado para comparar as predições entre modelos com dados reais.	Ploty

Fonte: O Autor (2021)

Esses recursos descritos se revelaram fundamentais para o entendimento das etapas e compreensão dos resultados.

Como demonstrado neste trabalho, as técnicas de visualização constituem uma ferramenta útil no kit de ferramentas de aprendizado de máquina, e o *Yellowbrick* oferece uma ampla seleção de visualizadores para atender às necessidades em cada etapa e estágio do pipeline do projeto de

ciência de dados. Variando de análise e seleção de recursos à seleção e otimização de modelo, os visualizadores *Yellowbrick* tornam mais fácil decidir quais recursos manter no modelo, qual modelo tem melhor desempenho e como ajustar os hiper-parâmetros de um modelo para atingir seu desempenho ideal para uso futuro. Além disso, a visualização da saída algorítmica também facilita a apresentação de *insights* para o público e as partes interessadas e contribui para a interpretabilidade simplificada dos resultados do aprendizado de máquina.

4.2 RESULTADOS E CONTRIBUIÇÕES DO TRABALHO

O objetivo deste trabalho consistiu em identificar ferramentas ou bibliotecas que pudessem ser utilizadas para produzir visualizações de dados em processos de *Machine Learning*, explorando-as por meio de estudos de casos com dois conjuntos de dados.

No decorrer do desenvolvimento dos casos de uso presentes neste trabalho, foi possível identificar que o uso das ferramentas e suas respectivas visualizações forneceram a possibilidade de novas descobertas, além de ajudarem a produzir conclusões coerentes.

Cita-se como exemplos, a etapa de avaliação dos modelos do conjunto de dados relacionado ao COVID-19, em que foi possível comparar o desempenho dos modelos desenvolvidos utilizando gráficos, comparando as previsões dos modelos utilizando dados de testes e dados reais. Além da etapa de pré-processamento dos dados do conjunto de dados iris, onde foi possível identificar uma forte correlação entre os atributos presentes no conjunto de dados.

É possível concluir que a aplicação das visualizações de dados nas diversas etapas de *Machine Learning* pode ser proveitosa nos mais diversos conjuntos de dados, contribuindo para a análise e descoberta de informações.

1. Identificar como a visualização de dados como ser aplicada nas etapas de um processo de *Machine Learning*.
2. Examinar o uso de recursos de visualização de dados sobre conjunto de dados.
3. Estabelecer relações entre modelos de visualizações e suas aplicações no processo de *Machine Learning*.
4. Implementar visualizações para dois conjuntos de dados, compondo um estudo de caso.

4.3 TRABALHOS FUTUROS

Este trabalho abordou algumas maneiras de se utilizar as ferramentas de visualização escolhidas, em dois conjuntos de dados distintos.

Como sugestão para trabalhos futuros, tem-se a utilização de outras bibliotecas, e até mesmo o uso das mesmas bibliotecas utilizadas neste trabalho, porém em conjuntos de dados

diferentes. Visto que, foram trabalhados apenas um problema de regressão utilizando o conjunto de dados sobre COVID-19 e um problema de classificação utilizando o conjunto de dados iris.

REFERÊNCIAS

BELLINI, R. **Aplicação de máquinas de suporte vetorial na classificação textual. Trabalho de Conclusão de Curso, Caxias do Sul**. 2020. 66 p.

BINU. **Seaborn Matplotlib plot to visualize Iris data**. 2020. Disponível em: <https://www.kaggle.com/biphili/seaborn-matplotlib-plot-to-visualize-iris-data>. Acesso em: 13 nov. 2020.

BRINK, H. **Real-World Machine Learning**. [S.l.]: Manning Publications, 2016. 264 p.

BURKOV, A. **The HundredPage Machine Learning Book**. [S.l.]: Andriy Burkov, 2019. 160 p.

CADY, F. **The data science handbook**. 1. ed. [S.l.]: John Wiley & Sons, 2017. 372 p.

CHEN WOLFGANG HäRDLE, A. U. Chun-houh. **Handbook of Data Visualization**. 2. ed. [S.l.]: Springer, 2008. 920 p.

CHOWDARY, D. H. **Decision Trees Explained**. 2020. Disponível em: <https://towardsai.net/p/programming/decision-trees-explained-with-a-practical-example-fe47872d3b53>. Acesso em: 10 out. 2020.

CODECADEMY. **The Machine Learning Process**. 2020. Disponível em: <https://www.codecademy.com/articles/the-ml-process>. Acesso em: 20 jul. 2020.

DEVELOPERS, S. yb. **Yellowbrick: Machine Learning Visualization**. 2020. Disponível em: <https://www.scikit-yb.org/en/latest>. Acesso em: 13 nov. 2020.

ELHAMRAOUI, Z. **What is Machine Learning?** 2020. Disponível em: <https://medium.com/@zahraelhamraoui1997/what-is-machine-learning-4f5e1e9015d7>. Acesso em: 03 ago. 2020.

ENSHENG, D.; HONGRU, D.; LAUREN, G. **An interactive web-based dashboard to track COVID-19 in real time**. 2020. Disponível em: <https://www.thelancet.com/journals/laninf/article/PIIS1473-3099(20)30120-1/fulltext>. Acesso em: 04 dez. 2020.

FAYYAD, U. The kdd process for extracting useful knowledge from volumes of data. ACM, p. 27–34, 1996. Disponível em: <https://sceweb.uhcl.edu/boetticher/ML_DataMining/p27-fayyad.pdf>. Acesso em: 18 nov. 2020.

FISHER, R. **Iris Data Set**. 1936. Disponível em: <https://archive.ics.uci.edu/ml/datasets/iris>.

FRAJ, M. B. **In Depth: Parameter tuning for SVC**. 2018. Disponível em: <https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769>. Acesso em: 04 dez. 2020.

GAVRILOVA, Y. **Classification Overview**. 2020. Disponível em: <https://serokell.io/blog/machine-learning-algorithm-classification-overview>. Acesso em: 10 out. 2020.

GERRY, N. **Logistic Regression Case Study**. 2017. Disponível em: <http://www.masteringdatascience.co.uk/CaseStudy_LogisticRegression.html>. Acesso em: 03 nov. 2020.

GODARD, T. **Detailed exploratory data analysis with python**. 2016. Disponível em: <https://www.kaggle.com/ekami66/detailed-exploratory-data-analysis-with-python/execution>. Acesso em: 23 jul. 2020.

GUK, Y. **Visualize Data Science pipeline with Yellowbrick**. 2020. Disponível em: <https://www.districtdatalabs.com/visualize-data-science-pipeline-with-yellowbrick>. Acesso em: 08 out. 2020.

GÉRON, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow Concepts, Tools, and Techniques to Build Intelligent Systems**. 2. ed. [S.l.]: O'Reilly Media, 2019. 856 p.

JEFFREY, H.; MICHAEL, B.; VADIM, O. A tour through the visualization zoo. **acmqueue**, v. 8, p. 22, 2010. Disponível em: <https://queue.acm.org/detail.cfm?id=1805128>. Acesso em: 18 out. 2020.

KEIM, D. *et al.* Challenges in visual data analysis. **Tenth International Conference on Information Visualisation (IV'06)**, p. 9–16, 2006. Disponível em: <https://www.semanticscholar.org/paper/Challenges-in-Visual-Data-Analysis-Keim-Mansmann/61dd71f9c2c877651a50a5422f23739aef42f09f>. Acesso em: 18 out. 2020.

KERR, G. **Visualizing KNN, SVM, and XGBoost on Iris Dataset**. 2016. Disponível em: <https://www.kaggle.com/mgabrielkerr/visualizing-knn-svm-and-xgboost-on-iris-dataset>. Acesso em: 15 nov. 2020.

KOTSIANTIS, S.; KANELLOPOULOS, D.; PINTELAS, P. Data preprocessing for supervised learning. **International Journal of Computer Science**, v. 1, p. 111–117, 01 2006. Disponível em: <https://www.researchgate.net/publication/228084519_Data_Preprocessing_for_Supervised_Learning>. Acesso em: 18 out. 2020.

L'HEUREUX, A. Machine learning with big data: Challenges and approaches. IEEE, p. 7776 – 7797, 2017. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7906512>. Acesso em: 18 out. 2020.

MCGREGOR, M. **Clustering Algorithms in ML**. 2020. Disponível em: <https://www.freecodecamp.org/news/8-clustering-algorithms-in-machine-learning-that-all-data-scientists-should-know>. Acesso em: 18 out. 2020.

MISHRA, S. **ML Clustering Algorithm**. 2019. Disponível em: <https://www.tutorialandexample.com/ml-clustering-algorithm>. Acesso em: 13 ago. 2020.

MITCHELL, T. M. **Machine Learning**. [S.l.]: McGraw-Hill Science/Engineering/Math, 1997. 432 p.

MOHRI, M.; AFSHIN, R.; AMEET, T. **Foundations of Machine Learning**. [S.l.]: The MIT Press, 2012. 504 p.

MÜLLER, S. G. A. C. **Introduction to Machine Learning with Python A Guide for Data Scientists**. 1. ed. [S.l.]: O'Reilly Media, 2017. 400 p.

PANT, A. **Workflow of a Machine Learning project**. 2011. Disponível em: <https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94>. Acesso em: 08 set. 2020.

RASCHKA, S. **Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow**. 2. ed. [S.l.]: Packt Publishing, 2017. 624 p.

RODRIGUES, V. **AUC e ROC Machine Learning**. 2020. Disponível em: <https://medium.com/bio-data-blog/entenda-o-que-é-auc-e-roc-nos-modelos-de-machine-learning-8191fb4df772>. Acesso em: 10 out. 2020.

SAID, V. T. A. **Visual Data Analysis**. 8. ed. [S.l.]: Springer, 2018. v. 46. 195 p.

SCHAPIRE, R. **COS 511: Theoretical Machine Learning**. 2008. 6 p. Disponível em: <https://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf>. Acesso em: 18 out. 2020.

SHARMA, A. **Introduction to Machine Learning in Python**. 2018. Disponível em: <https://www.datacamp.com/community/tutorials/introduction-machine-learning-python>. Acesso em: 21 nov. 2020.

SHARMA, P. **K-Means Clustering**. 2019. Disponível em: <https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering>. Acesso em: 18 out. 2020.

SIDANA, M. **Machine Learning Types of Classification**. 2017. Disponível em: <https://medium.com/sifium/machine-learning-types-of-classification-9497bd4f2e14>. Acesso em: 10 out. 2020.

TAYO, B. **Role of Data Visualization in Machine Learning**. 2020. Disponível em: <https://medium.com/towards-artificial-intelligence/role-of-data-visualization-in-machine-learning-a6dd62ad1082>. Acesso em: 21 nov. 2020.

TURKAY, C.; LARAMEE, R.; HOLZINGER, A. On the challenges and opportunities in visualization for machine learning and knowledge extraction: A research agenda. In: . [S.l.: s.n.], 2017. p. 191–198. Disponível em: <https://www.researchgate.net/publication/319248120_On_the_Challenges_and_Opportunities_in_Visualiza
Acesso em: 18 out. 2020.

WARE, C. **Information Visualization: Perception for Design (Interactive Technologies)**. 3. ed. [S.l.]: Morgan Kaufmann, 2012. 536 p.

WILLEMS, K. **Python Exploratory Data Analysis Tutorial**. 2017. Disponível em: <https://www.datacamp.com/community/tutorials/exploratory-data-analysis-python>. Acesso em: 18 nov. 2020.

WULFF, D. **What is Machine Learning?** 2019. Disponível em: <https://therbootcamp.github.io/ML_2019May/_sessions/WhatIsML/WhatIsML.html>. Acesso em: 13 ago. 2020.