



Ph.D. Professor Aluisio Igor Rego Fontes

Gustavo de Freitas Rodrigues

José Matheus Bento

**Autor da apostila**

Ph.D. Professor Aluisio Igor Rego Fontes

**Instrutor do curso**

Larissa Jéssica Alves – Analista de Suporte Pedagógico

**Revisão da apostila**



**Autor**

**Aluisio Igor Rêgo Fontes**

Aluisio I. R. Fontes possui graduação em Engenharia da Computação pela Universidade Federal do Rio Grande do Norte (2010), mestrado (2012) e doutorado (2015) em Engenharia Elétrica e Computação pela Universidade Federal do Rio Grande do Norte. Atualmente é professor do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte campus Pau dos Ferros, onde coordena o Laboratório de Análise de Dados e Inteligência Computacional (NADIC). Por mais de 13 anos, vem se dedicando a criação de soluções inovadoras utilizando inteligência artificial e análise de dados para inserir tecnologias em várias organizações. No contexto de sistemas corporativos, desenvolvi soluções com acesso massivo de usuários, alta disponibilidade e robotização de processos. Na pesquisa acadêmica sou autor/co-autor de mais de 15 publicações em periódicos internacionais nas áreas de Inteligência Artificial e teoria da informação, tendo publicado em veículos de grande reputação nestas áreas. Sou revisor de periódicos internacionais (e.g., IEEE Signal Processing, Expert System with Application, IEEE Access e eurasip journal on advances in signal processing). Tenho experiência na área de Engenharia da Computação, com ênfase em Sistemas de Computação, atuando principalmente nos seguintes temas: Processamento digital de sinais, inteligência Computacional, Teoria da Informação, Correntropia, Processamento em Big Data, desenvolvimento de sistemas corporativos e computação de alto desempenho.



**José Matheus Bento**

Fiz curso técnico em informática no IFRN, campus Pau dos Ferros. Atualmente estou cursando Análise e Desenvolvimento de Sistemas no mesmo instituto. Sempre gostei de tecnologia, e assim que aprendi a programar, procurei aprender diversas tecnologias, para me aprimorar e encontrar aquela que mais gostasse. Desde 2020 participo do NADIC onde trabalhei com desenvolvimento backend utilizando o framework Django, em 2021 iniciei um estágio na JusInvestimens ainda focado em backend. No ano de 2022 fui contratado tanto para dar continuidade ao sistema da empresa como também para desenvolver um aplicativo nativo para Android. Tenho muito interesse em redes neurais e aprendizado de máquina, pois desenvolvi um sistema de reconhecimento de placas de veículos utilizando IA.



**Gustavo de Freitas Rodrigues**

Sou técnico de informática no IFRN - Campus Pau dos Ferros, estudante na graduação de Tecnologia da Informação, com interesse em ciências da computação pela UFRN (Universidade Federal do Rio Grande Do Norte). Tem experiência com circuitos e eletrônica, já tendo trabalhado na construção de robôs, utilizando arduino. Participei no NADIC (Núcleo de Análise de Dados e Inteligência Computacional), na área de Deep Learning, Computer Vision e IoT. Atualmente, estou trabalhando e aprofundando meus conhecimentos em ciência de dados e aprendizado de máquina.

**APRESENTAÇÃO**

A presente apostila é um instrumento teórico que complementa o curso de capacitação de Inteligência Artificial. Nela, veremoso primeiro módulo daementa do curso. Este material é baseado em artigos científicos, periódicos, revistas científicas e livros científicos. É extremamente recomendável ao aluno que, ao final da leitura de cada seção, realize os exercícios propostos e acesse os materiais indicados nas referências bibliográficas, para aprofundar a leitura desse material e complementar o que foi lido aqui.

***Boa Leitura !!***

**Sumário**

[**1 Pré-processamento e Transformação de Dados 6**](#_heading=h.qru63u2dzk2u)

[**1.1 O que é o pré-processamento e limpeza de dados 6**](#_heading=h.rmvynn2qw5jy)

[1.1.1 Quantidade insuficiente 7](#_heading=h.m5aeqzfasoyj)

[1.1.2 Dados Não Representativos 8](#_heading=h.ia16s4nj1r4m)

[1.1.3 Dados de Baixa Qualidade 9](#_heading=h.kqfrvhimyz4f)

[1.1.4 Características Irrelevantes 11](#_heading=h.h2ngff218r8d)

[1.1.5 *Overfitting* (Sobreajuste) e *Underfitting* (Sub Ajuste) 12](#_heading=h.yny5ud7n7wp9)

[**1.2 Análise Exploratória de Dados 14**](#_heading=h.cwckhdgtse76)

[**1.3 Limpeza dos Dados (com o Pandas) 17**](#_heading=h.2r0nprumqij)

[1.3.1 Valores faltantes 18](#_heading=h.7krrz6d8zh1k)

[1.3.2 Lidando com *outliers* 19](#_heading=h.u41zfbfji5h4)

[1.3.3 Lidando com valores duplicados 19](#_heading=h.p4huvfnmoxhp)

[1.3.5 Transformando dados (com o Pandas) 20](#_heading=h.8c2usgnw4ezg)

[1.3.6 Limpeza e adequação de colunas 20](#_heading=h.wdf7stn9kw8m)

[**1.4 Filtrando DataFrames 22**](#_heading=h.43fsxdcfy6j9)

[**1.5 Redução de dimensionalidade 22**](#_heading=h.crj09tdc3ik4)

[**1.6 Análise dos Componentes Principais 25**](#_heading=h.7pf4fmqcxjr1)

# Pré-processamento e Transformação de Dados

No mundo de Machine Learning e Inteligência Artificial, as tarefas essenciais incluem escolher ou adaptar um algoritmo de aprendizado e treiná-lo com dados. Aqui, enfrentamos dois desafios principais: algoritmos ineficazes e dados de baixa qualidade. Este módulo foca em abordar e solucionar problemas relacionados aos dados, utilizando técnicas e conceitos para aprimorar ou transformar esses dados.

## O que é o pré-processamento e limpeza de dados

A eficácia de algoritmos de aprendizado é totalmente relacionada à qualidade dos dados dos quais aprende. Sendo comum a existências de ruídos, imperfeições, valores incorretos, inconsistentes ou ausentes nos conjuntos de dados, por exemplo em um dataset dos clientes de uma loja, nem todos os clientes revelam suas idades. Ou até mesmo *outliers*, que são exceções, são dados que se diferenciam muito dos demais, como na imagem 1 a seguir, onde o ponto A pode ser considerado um *outlier* por se distanciar muito dos outros pontos.

Um outlier, em estatística, é um ponto de dados que se desvia significativamente do padrão geral de um conjunto, sendo uma observação que está fora do esperado em relação às demais. Essas observações atípicas podem distorcer a análise estatística e influenciar a interpretação dos resultados, uma vez que podem indicar a presença de anomalias, erros de medição ou eventos incomuns. Identificar e compreender *outliers* é crucial para a validade das análises estatísticas, pois eles podem impactar negativamente a precisão dos modelos e inferências. Diversas técnicas, como o uso de diagramas de caixa (boxplots) e critérios estatísticos, são empregadas na detecção e gestão de *outliers*, permitindo uma abordagem mais robusta na análise de dados.

Contudo, é possível mitigar ou resolver essas questões por meio do pré-processamento e da limpeza de dados, uma etapa crucial que consiste na remoção ou ajuste de informações que possam comprometer a eficácia do algoritmo. Essas práticas permitem aprimorar a qualidade e confiabilidade dos dados, contribuindo para a robustez e precisão das análises subsequentes. Ao realizar uma cuidadosa limpeza dos dados, é possível minimizar a influência de *outliers*, erros de medição e outras anomalias, promovendo uma base sólida para análises estatísticas mais precisas e resultados mais confiáveis.

Existem alguns cenários principais em que essa etapa é imprescindível, que são: quantidade insuficiente de dados, dados não representativos, viés de amostragem, dados de baixa qualidade, características irrelevantes, *Overfitting* e *Underfitting*.

O *overfitting* ocorre quando um modelo é excessivamente complexo e se ajusta perfeitamente aos dados de treinamento, capturando até mesmo o ruído ou variações insignificantes presentes nesses dados. Esse ajuste excessivo pode resultar em uma perda significativa de generalização, levando o modelo a apresentar desempenho sub ótimo em dados não vistos, ou seja, não utilizados durante o treinamento. Para mitigar o *overfitting*, técnicas como regularização e a inclusão de mais dados de treinamento são frequentemente empregadas. Para exemplificar, imagine que você está estudando para uma prova e decora todas as perguntas de um teste anterior, em vez de entender os conceitos. Se a prova real tiver perguntas diferentes, você provavelmente não se sairá bem. Isso é semelhante ao *overfitting*, onde um modelo de *Machine Learning* aprende os detalhes e ruídos específicos dos dados de treinamento a ponto de prejudicar seu desempenho em novos dados. Para evitar isso, utilizamos técnicas como regularização ou adicionamos mais dados de treinamento.

Por outro lado, o *underfitting* ocorre quando um modelo é muito simples para capturar adequadamente a complexidade dos dados de treinamento. Isso resulta em uma falta de ajuste, onde o modelo não consegue representar de forma adequada as relações presentes nos dados. O underfitting geralmente leva a baixo desempenho tanto nos dados de treinamento quanto nos dados de teste, indicando que o modelo não consegue aprender com precisão os padrões subjacentes. A solução para o underfitting pode envolver a escolha de modelos mais complexos, a seleção de recursos mais relevantes ou o aumento da quantidade de dados de treinamento para fornecer ao modelo uma visão mais abrangente e precisa do problema em questão. Encontrar o equilíbrio adequado entre complexidade do modelo e capacidade de generalização é fundamental para desenvolver modelos de aprendizado de máquina eficazes. Como exemplo, o *underfitting* é como estudar muito pouco para a prova, não cobrindo todos os tópicos necessários. Aqui, o modelo de *Machine Learning* é muito simples e não consegue captar a complexidade dos dados, resultando em um desempenho ruim tanto nos dados de treinamento quanto nos de teste. Para solucionar o underfitting, podemos usar modelos mais complexos, melhorar a seleção de características ou aumentar a quantidade de dados de treinamento.

### Quantidade insuficiente

Para que uma criança aprenda o que é uma maçã, é preciso que você aponte para uma maçã e diga “maçã” (possivelmente repetindo algumas vezes esse procedimento). Agora, a criança consegue reconhecer maçãs em todos os tipos de cores e formas. No aprendizado de máquina é preciso uma grande quantidade de dados para que a maioria dos algoritmos de Aprendizado de Máquina funcione corretamente. Você precisará de milhares de exemplos mesmo para problemas muito simples, e para problemas complexos, como reconhecimento de imagem ou da fala, precisará de milhões de exemplos (a menos que possa reutilizar partes de um modelo existente). (GÉRON, 2022)

Uma das consequências da falta de dados é a má generalização do aprendizado do algoritmo, nesses casos é importante obter mais dados, extrair mais características dos dados existentes ou simplificar o modelo de aprendizagem.

### Dados Não Representativos

Quando o objetivo é fazer uma boa generalização, os dados para treinamento são essenciais e devem representar bem o motivo da generalização. Para isso, esses dados devem abranger diferentes realidades que podem ser buscadas na generalização. Na imagem 1, no lado esquerdo, o crescimento linear possui grandes chances de não representar bem a realidade, ainda mais se levarmos em consideração a população de cada país e entre outros aspectos. Já no lado direito, existe uma maior chance de termos uma representação mais fidedigna da realidade, onde o gráfico mostra um crescimento exponencial.

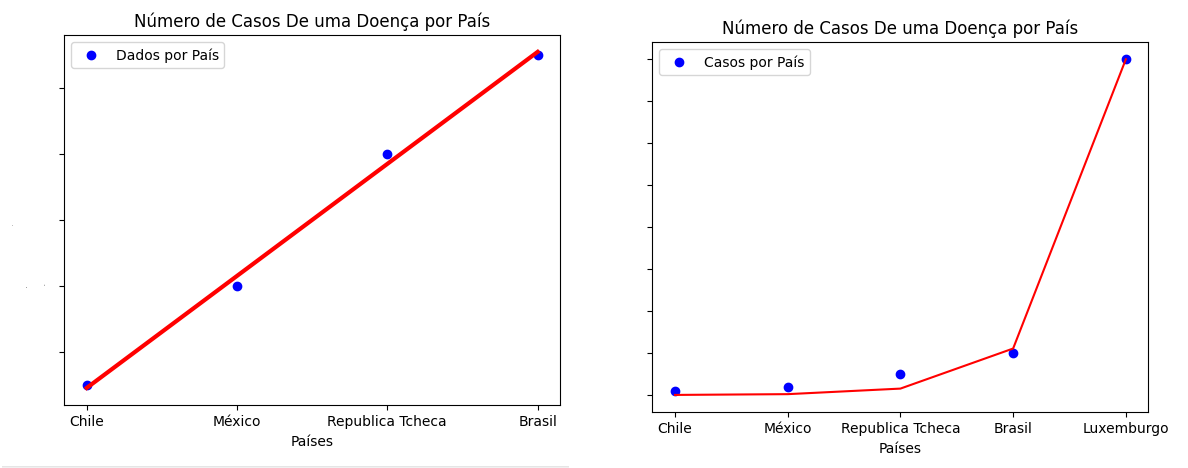


Imagem 1 - Exemplo de representação de dados

Fonte: Autoria Própria

Outro problema importante é o viés na amostragem, ou seja, quando o conjunto de dados favorece uma característica ou uma representação. Um exemplo famoso disso ocorreu na eleição eleitoral dos USA de Landon contra Roosevelt, onde a *Literary Digest* conduziu uma pesquisa com 10 milhões de pessoas, em que apenas 2,4 milhões responderam apontando uma vitória de Landon por 57% dos votos. Porém na realidade Roosevelt ganhou por 62%, tamanho erro se deu devido a escolha do público alvo, que favoreciam pessoas mais ricas, logo propensas a votar em London que era republicano e outro ponto foi o baixo índice de resposta, que caracteriza um viés de falta de resposta.

### Dados de Baixa Qualidade

De forma óbvia, caso os dados de treinamento estejam com erros, *outliers* (por exemplo, erros de digitação ou valores faltantes) e ruídos (por exemplo, devido a medições de baixa qualidade), o algoritmo terá mais dificuldade para detectar os padrões subjacentes, portanto, menos propício a um bom funcionamento. Muitas vezes vale a pena dedicar um tempo explorando e limpando seus dados de treinamento, de forma que:

* Se algumas instâncias são claramente *outliers*, isso pode ajudar a descartá-las ou tentar manualmente a correção dos erros;
* Se faltam algumas características para algumas instâncias, deve-se optar por ignorar as características faltantes, ou as instâncias com dados faltantes ou até mesmo preencher os valores ausentes.

Para tratar os *outliers* existem 2 principais formas:

1. Excluir

Quando estamos tratando um *outlier*, o jeito mais fácil de tratar é simplesmente excluir a instância, contudo, isso pode causar impactos negativos. Quando excluímos algum dado de nossa base, querendo ou não, estamos diminuindo o tamanho da nossa base e isso nunca é bom. Quando temos uma base de dados com poucas instâncias, a exclusão de dados vai alterar muito nossa base, por exemplo, se em uma base com 10 instâncias houver 4 *outliers* e optamos por excluí-los, cerca de 40% da base de dados será perdida e isso é inviável. Por isso, é importante definir um limiar de quantos por cento estamos dispostos a perder de nossa base de dados original. Geralmente, esse percentual é em cerca de 15%.

1. Substituir

Outro jeito de tratar *outliers* é usando a substituição, ou seja, trocando o valor aberrante por outro valor que é conhecido. Aqui, o mais usual é utilizar alguma medida de tendência central, ou seja, média, mediana ou moda. Vale ressaltar que, a substituição por moda é mais usual quando estamos tratando valores categóricos. Para valores numéricos, utiliza-se média ou mediana.

Para identificar *outliers* e *noise* na base de dados, utiliza-se 3 principais métodos: Análise de resíduos utilizando regressão linear, método do desvio padrão e amplitude interquartil. A análise de resíduos usa uma regressão linear para identificar valores que estão distantes dos valores previstos pela regressão. O método de desvio padrão considera *outliers* valores que estão mais do que 3 desvios padrões abaixo ou acima da média.

Contudo, a mais comum é a amplitude interquartil. A amplitude interquartil é uma medida de dispersão que representa a distância entre o primeiro e terceiro quartil. Dado pela fórmula: . Onde IQR é a amplitude interquartil, Q3 é o terceiro quartil representando 75% dos dados e Q1 é o primeiro quartil que representa 25% dos dados. Para ser considerado um *outlier* utiliza-se a seguinte condição: Considere V sendo o valor do dado que deseja ser categorizado como *outlier* ou não.

* Caso . Ele é considerado um *outlier* inferior.
* Caso Ele é considerado um *outlier* superior*.*

Por exemplo, considere o seguinte conjunto de dados: . Nele, temos , logo, . Temos que 6. Assim, de acordo com nossas condições o valor 100 seria considerado um *outlier superior* pois . Assim, poderíamos substituir o valor 100 pela média ou pela mediana, ou então, excluir ele.

O valor 1,5 para multiplicar o IQR é um valor comumente utilizado, contudo, ele pode ser ajustado. Dependerá do quão rigoroso o analista quer ser em relação aos *outliers* e também do balanceamento e do tamanho da base de dados. Caso a base de dados seja extremamente pequena, o valor poderá ser muito rigoroso para a base, levando a considerar *outlier* valores que não são realmente. Caso a base de dados seja bastante assimétrica, o valor poderá ser muito baixo para conseguir captar todos os valores de *outliers*. A seguir podemos ver isso de forma visual em um gráfico de boxplot.

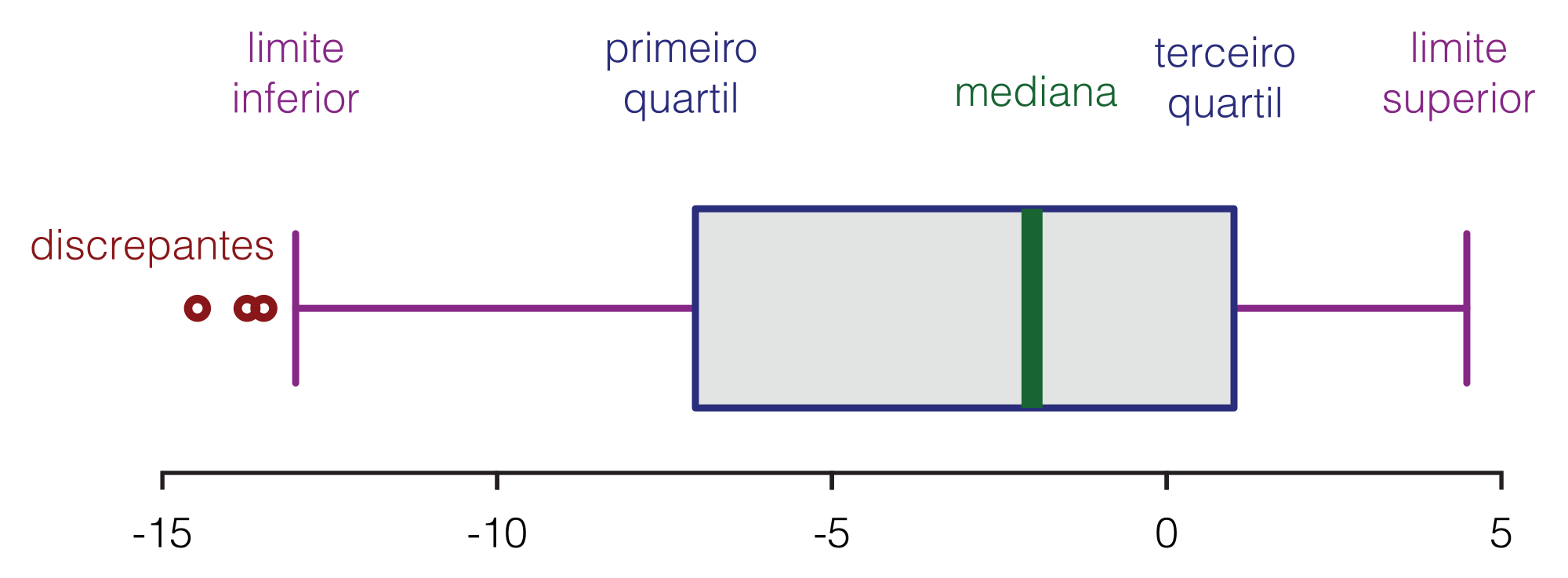


Imagem 2 - Exemplo de Boxplot

Fonte: Autoria Própria

### Características Irrelevantes

Uma parte crítica do sucesso de um projeto de Aprendizado de Máquina é criar um bom conjunto de características para o treinamento, processo chamado de *feature engineering* que envolve:

* Seleção das características: selecionar as mais úteis para treinar entre as existentes;
* Extração das características: combinar características existentes para produzir uma mais útil (como vimos anteriormente, os algoritmos de redução de dimensionalidade podem ajudar);
* Criação de novas características ao coletar novos dados.

A seguir tem-se um exemplo, o gráfico da esquerda possui menos pontos e também uma acurácia menor, porém na direita com muito mais dados existe uma acurácia muito alta.

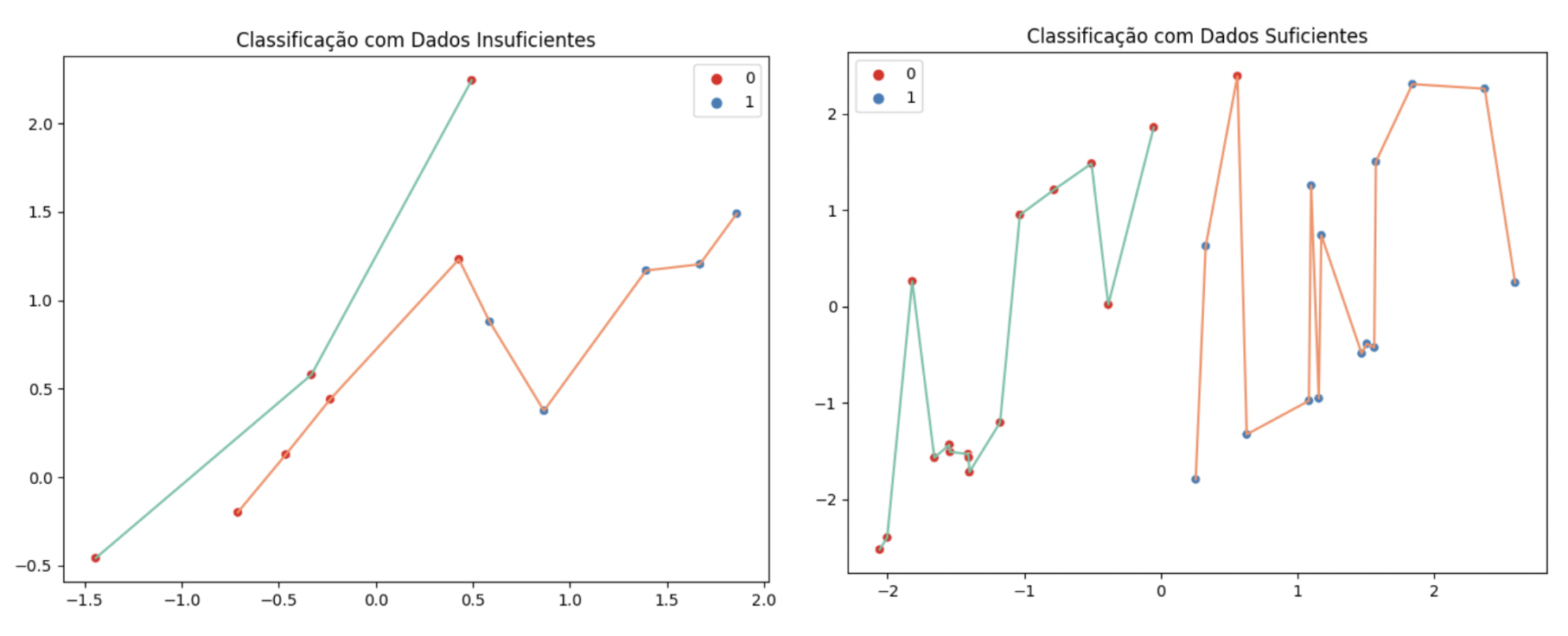


Imagem 3 - Exemplo classificação com dados faltantes

Fonte: Autoria Própria

### *Overfitting* (Sobreajuste) e *Underfitting* (Sub Ajuste)

A generalização faz parte do aprendizado, se todas as casas que conhecemos são azuis então podemos generalizar e pensar que todas as casas que existem são azuis, esse problema é comum na Machine Learning. Modelos complexos como redes neurais profundas podem detectar padrões sutis nos dados, mas se o conjunto de treinamento é ruidoso ou se é muito pequeno (o que introduz ruído na amostragem), então o modelo provavelmente detecta padrões no próprio ruído. É óbvio que esses padrões não serão generalizados para novas instâncias. Ou seja, os ruídos e imprecisões do conjunto de dados podem não se reproduzir na realidade.

Um exemplo disso poderia ser um modelo que classifica o IDH de um país considerando apenas os países europeus, o modelo poderá aprender que **todos** os países com a letra *W* possuem IDH superior a 7, como: Norway (9.6), Sweden (9.4) e Switzerland (9.2). Porém considerando todos os países do mundo, esse padrão se torna falso, por exemplo o Zimbabwe (1.4).

Logo, o *Overfitting* ocorre quando o modelo é muito complexo em relação a quantidade e ruído dos dados de treinamento, para contornar isso pode-se:

* Reduzir o ruído nos dados;
* Coletar mais dados de treinamento;
* Reduzir a complexidade do modelo.

Um contraponto a isso é o *Underfitting*, ocorre quando o modelo é muito simples para o aprendizado da estrutura subjacente dos dados. Podemos voltar ao exemplo do IDH, se nosso modelo for linear não poderá representar a realidade de forma fidedigna pois existem diversos fatores e parâmetros que influenciam no IDH de um país. E para contornar isso, pode-se:

* Selecionar um modelo mais poderoso, com mais parâmetros;
* Alimentar o algoritmo de aprendizado com melhores características (feature engineering);
* Reduzir as restrições no modelo.

Abaixo temos a imagem 4 que demonstra o que acontece com o *overfitting* e *underfitting* em algoritmos de classificação e regressão.

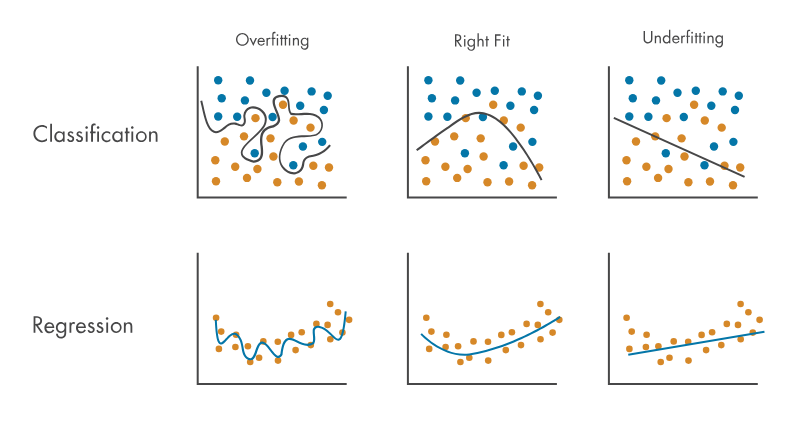


Imagem - Exemplo visual de *overfitting* e *underfitting*

Fonte: Autoria Própria

## Análise Exploratória de Dados

Basicamente existem dois tipos, univariada e multivariada, em uma analisa-se apenas uma variável e na outra analisa-se mais de uma em conjunto, sendo que elas podem ou não ter gráficos para a visualização.

Na univariada, o objetivo é simplesmente entender aquela variável, descrever seus padrões e como ela funciona. Já a Univariada Gráfica, proporciona uma amostra de uma parte dos dados, as formas mais comuns de se representar esse gráficos são histogramas e *box plots*. Na imagem 5 pode-se vê-los, note que o boxplot define os quartis e o histograma mostra a distribuição dos valores.

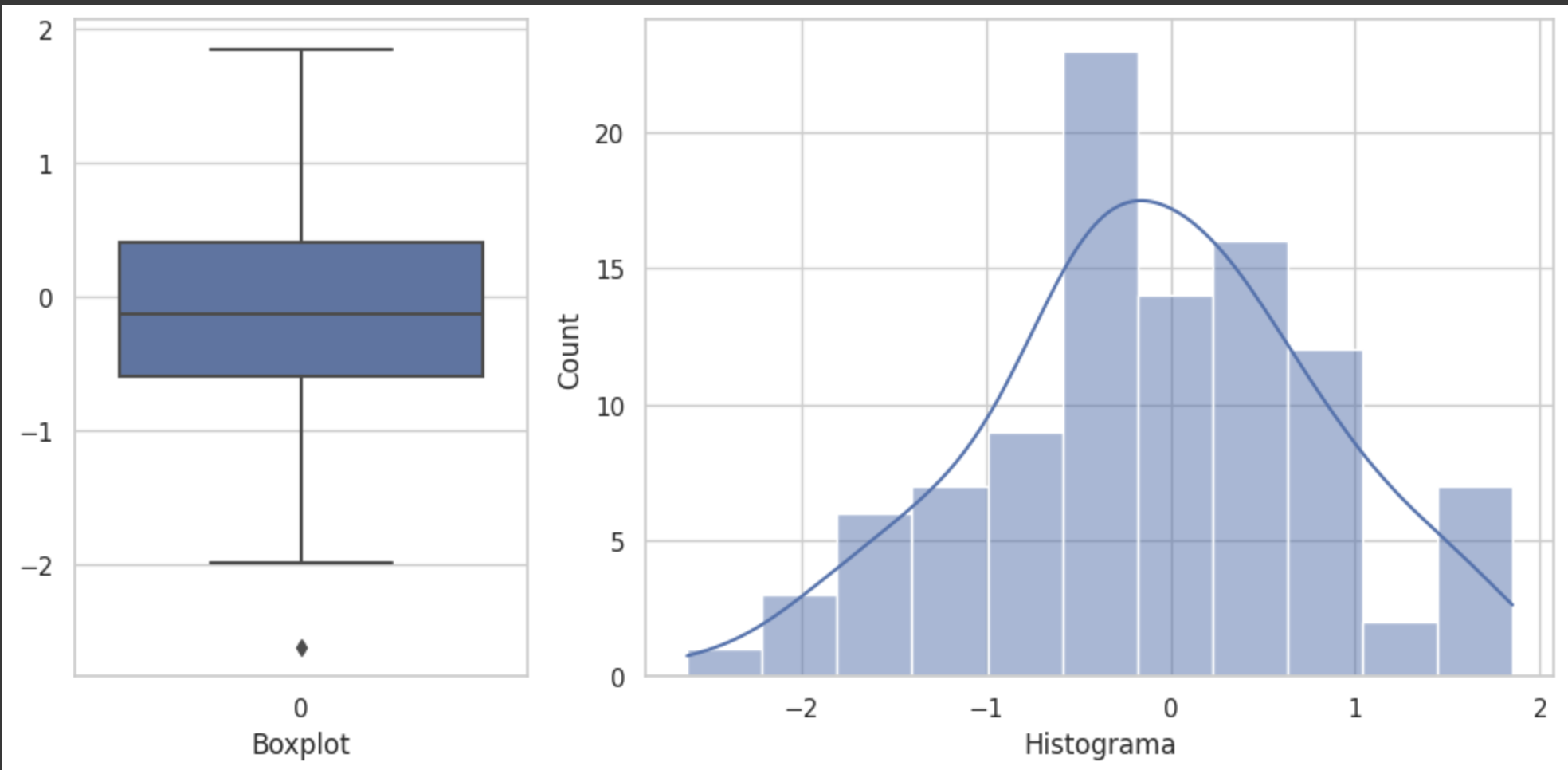


Imagem 5 - Boxplot e Histograma

Fonte: Autoria Própria

Já as análises multivariadas mostram a relação entre duas ou mais variáveis utilizando correlações em seus valores ou estatísticas. Dessa forma, a análise multivariada gráfica tenta trazer comparações dessas variáveis ou como elas se relacionam. Alguns exemplos são o scatter, heat map, gráficos variados de linhas ou barras e bubbles.

O scatter representa a relação de duas colunas ou dimensões valor por valor, fazendo assim uma nuvem de pontos que podem indicar um padrão na relação desses valores. Na imagem 6 mostra um gráfico de *scatter* com um padrão linear, à medida que os valores do eixo X aumentam o eixo Y também aumenta.

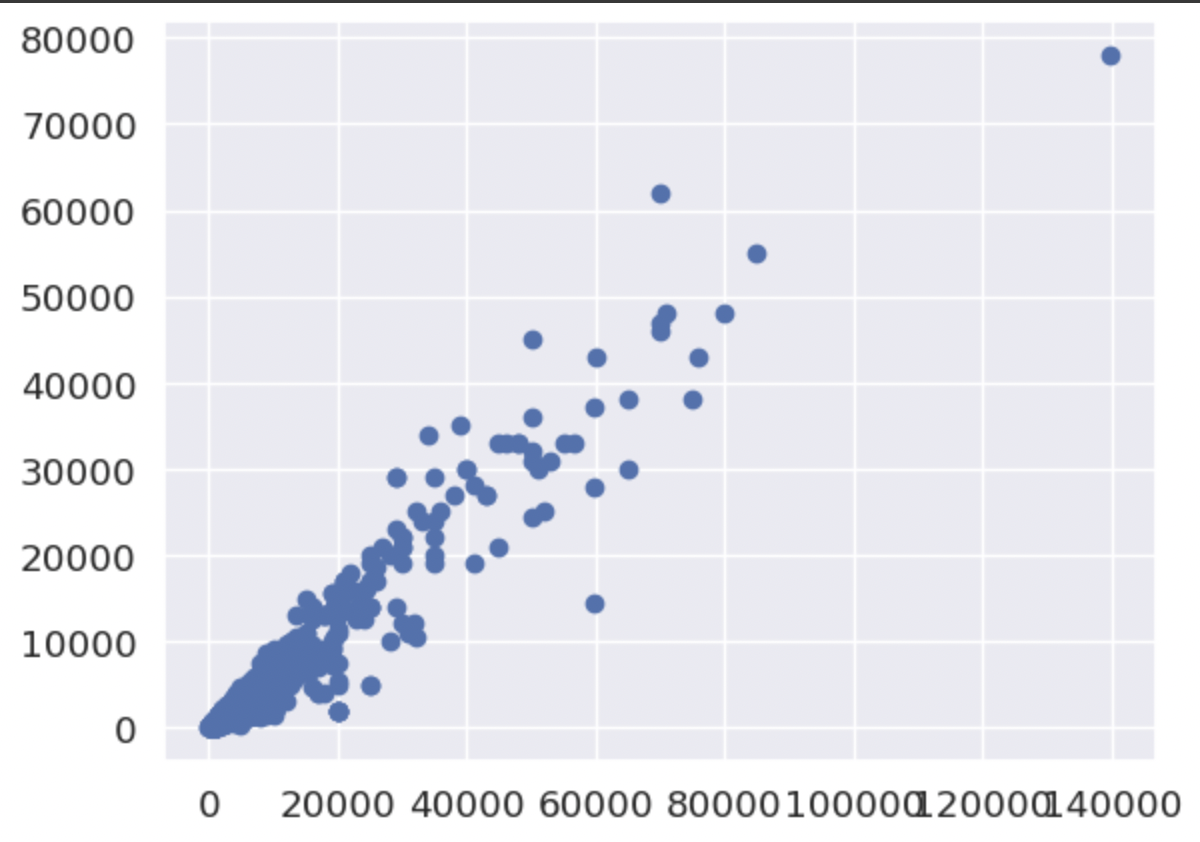


Imagem 6 - Exemplo de Scatter

Fonte: Autoria Própria

Já o heat map mostra a correlação entre todos os pares de dimensões possível, coluna por coluna, de forma que o valor de cada célula pode variar de 1 até -1, representando uma relação diretamente proporcional forte e uma relação inversamente proporcional forte, respectivamente. Esse gráfico nos mostra que se duas variáveis são muito correlacionadas então podemos encontrar uma forma de transformá-las em apenas uma variável. Dessa forma, teremos menos atributos a treinar sem perder informações.

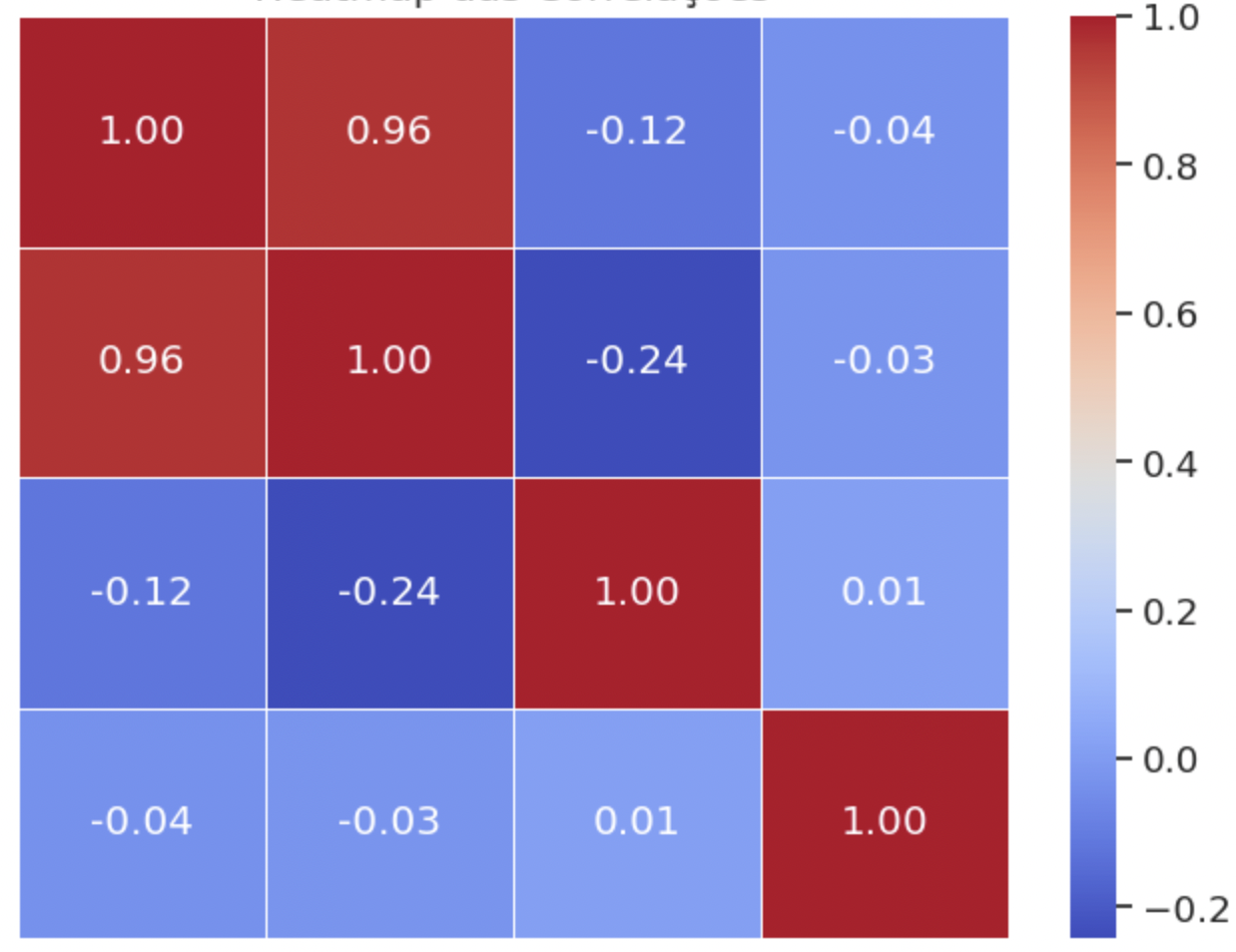


Imagem 7 - Heat map

Fonte: Autoria Própria

Tem-se também na análise não gráfica alguns pontos cruciais para entender como se encontram os dados, o primeiro desses pontos são as propriedades estatísticas de cada dimensão. Com o Pandas pode-se utilizar o método *describe()* para vê-los.

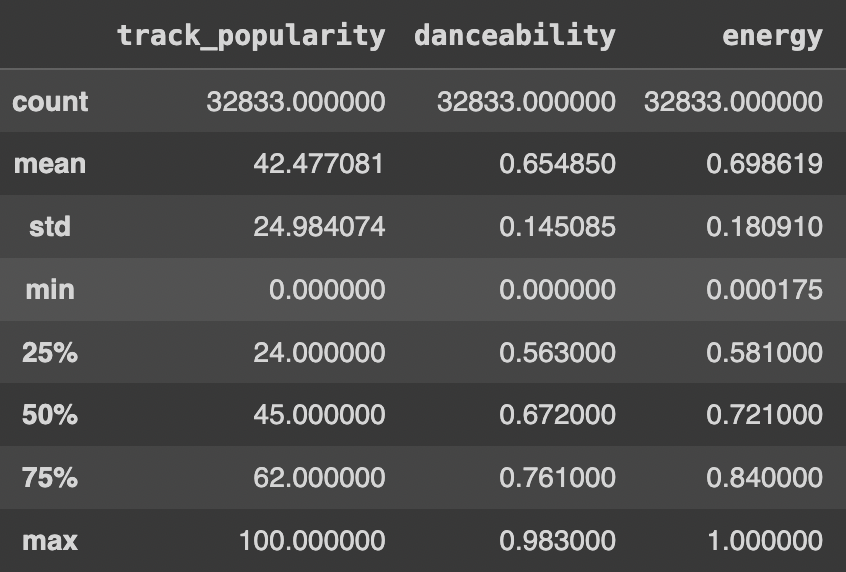


Imagem 8 - *Describe* de um *dataset*

Fonte: Autoria Própria

Na imagem 8 pode-se observar os principais pontos do *describe*, que são:

* *count:* soma de todos os valores de cada dimensão;
* *mean:* média dos valores;
* *std:* desvio padrão dos valores;
* *min:* valor mínimo da dimensão;
* 25%: valor do primeiro quartil;
* 75%: valor do 3 quartil;
* *max:* valor máximo da dimensão;

Outro ponto importante é a análise dos valores faltantes (NaN), que com o Pandas pode ser realizada utilizando a fun *isna()* que retorna os valores NaN em cada dimensão e depois utilizar a função *sum()* que somará o total desses valores como no exemplo: ***data\_frame.isna().sum()***, o que vai resultar em algo como a imagem 9, onde temos o título de cada coluna e o total de valores faltantes.

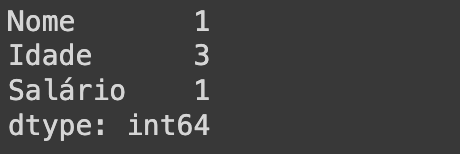
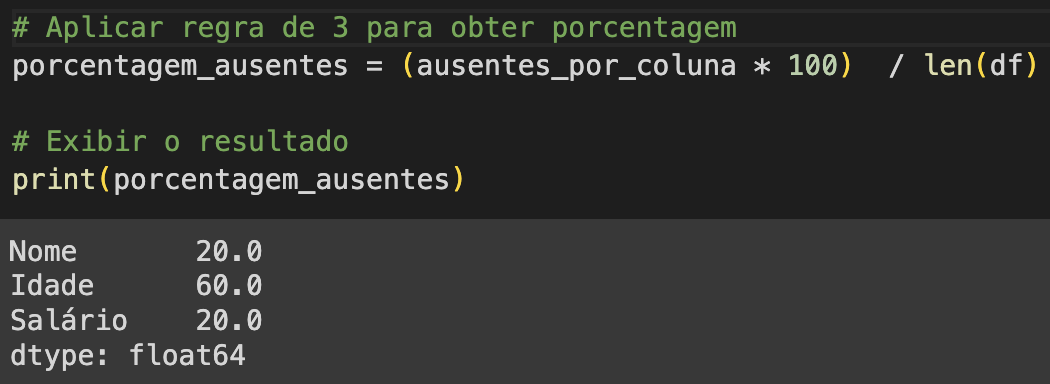


Imagem 9 - Valores faltantes de um *dataset*

Fonte: Autoria Própria

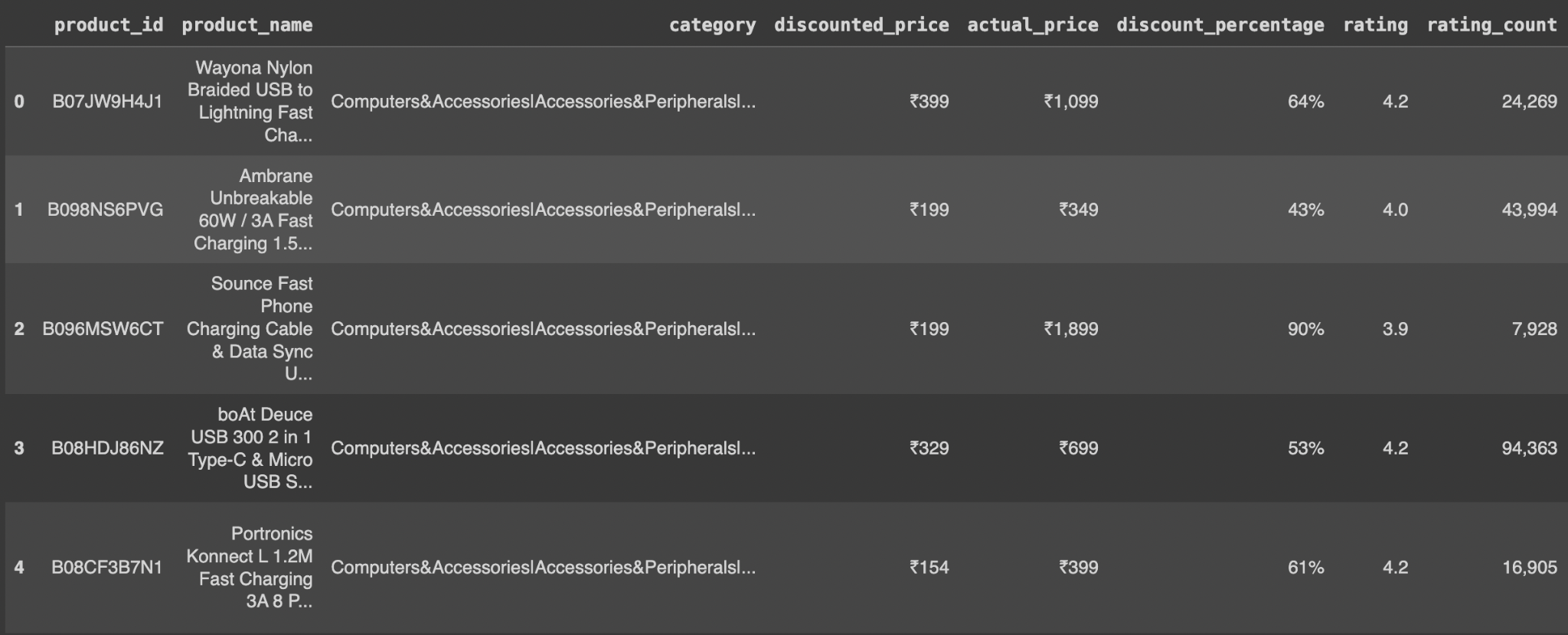
Isso pode ser ainda mais incrementado para mostrar a porcentagem e assim conseguir visualizar melhor o impacto que esses dados podem causar, e isso pode ser feito da seguinte forma:



Outro ponto importante é utilizar a função *info()* para obter os tipos de cada dimensão, pois assim se saberá quais delas são numéricas e quais não, pois muitas vezes algumas colunas precisam passar a ser numéricas para que se possa analisar de forma estatística, ou até mesmo ajustar os formatos de datas por exemplo.

## Limpeza dos Dados (com o Pandas)

Para explorar a limpeza de dados com o pandas utilizaremos inicialmente o *dataset* [amazon.csv](https://raw.githubusercontent.com/Matheus51bento/datasets/main/amazon.csv), esse conjunto de dados foi criado a partir de vendas e avaliações de produtos na Índia.



### Valores faltantes

Inicialmente vamos tratar os valores faltantes, geralmente são representados por NA, NaN, ou até mesmo None.

Nesse dataset pode-se obtê-los da seguinte forma:

nullvalues = data.isna().sum()

print(nullvalues)

product\_id 0

product\_name 0

category 0

discounted\_price 0

actual\_price 0

discount\_percentage 0

rating 0

rating\_count 2

E existem algumas técnicas principais para lidar com esses valores que são:

1. Deixá-los como estão

Dependendo da quantidade, e do objetivo pode não ser interessante para o algoritmo aprender (ou deixar de aprender) de dados faltantes.

1. Deletar

Em pequenas quantidades de dados essa pode ser uma opção viável pois podem não fazer falta, porém em grandes quantidades isso pode ser um problema pois no geral nunca é bom diminuir o tamanho do *dataset.*

1. Preenchê-los com média, mediana ou moda

Dessa forma se preservam os outros dados que a instância pode conter, e a alteração dos valores faltantes será um valor médio então terá um peso menor, não afetando muito na generalização.

1. Utilizar machine learning para preenchê-los

É possível utilizar algum algoritmo de aprendizado para preencher os valores faltantes com padrões que podem ser percebidos nos dados existentes. Dessa forma criando mais variabilidade no *dataset* o que gera mais aprendizado.

No caso deste dataset, como apenas duas linhas são afetadas por dados faltantes pode-se excluí-las, umas que não se perderá muita informação. Isso pode ser feito da seguinte forma:

data[data["rating\_count"].isnull()]

data = data.loc[~data['rating\_count'].isnull()]

Neste caso, selecionamos as entradas NaN com a função *isnull()* e em seguida utilizamos o operador *tilde* (~) para negar a condição *isnull()* que dessa forma retornará apenas os valores que não são nulos.

### Lidando com *outliers*

Como explicado anteriormente, os *outliers* podem ser facilmente identificados utilizando a análise dos quartis do dado, utilizando a amplitude inter quartil. desse modo podemos também excluí-los como uma função como a seguinte:

# São passados o dataframe e coluna a se tratar

def remove\_outliers(df, column):

# método quantile() retorna os valores do

# primeiro e terceiro quartis

q1 = df[column].quantile(0.25)

q3 = df[column].quantile(0.75)

# calcula-se a amplitude interquartil

iqr = q3 - q1

# define-se quais são os limiares inferior

# e superior de outliers

upper\_boundary = q3 + 1.5 \* iqr

lower\_boundary = q1 - 1.5 \* iqr

# filtra o dataset

new\_df = df.loc[(df[column] > lower\_boundary) & \

(df[column] < upper\_boundary)]

return new\_df

### Lidando com valores duplicados

Outro ponto importante é lidar com valores duplicados, como eles não agregam novas informações ao *dataset* podem ser simplesmente apagados, deixando é claro uma de suas cópias. Com pandas isso pode ser feito facilmente com a função: dataframte.drop\_duplicates(inplace=True).

### Transformando dados

Transformar dados é importante em alguns momentos, como por exemplo, quando existem classes ou valores em notações diferentes das ideais. Um exemplo de transformar classes seria uma coluna **desempenho** que tem valores "bom", "médio" e "ruim", se esses valores forem numéricos tanto a análise quanto o treinamento se beneficiaram mais deles, e isso pode ser feito da seguinte forma:

desempenho\_dict = {'ruim': -1, 'medio': 0, 'bom': 1}

dataframe['desempenho'].replace(desempenho\_dict)

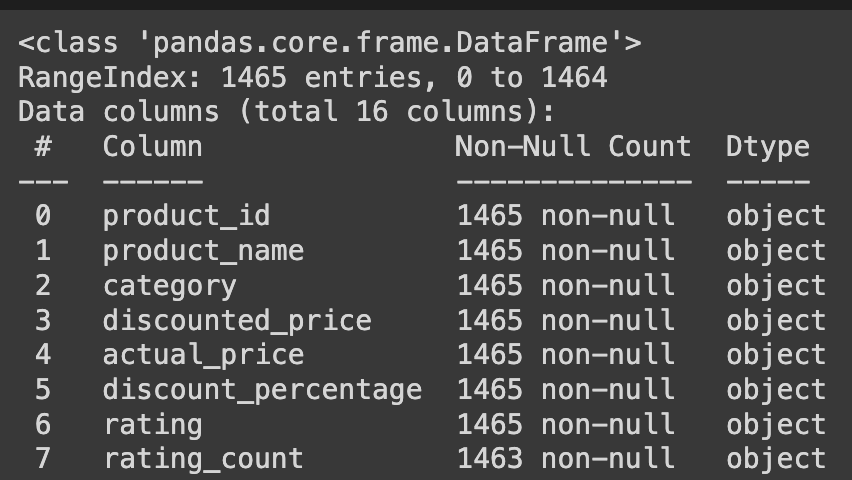
Dessa forma, cria-se um dicionário com os valores da troca e se aplica no data frame usando a função *replace().*

Outro exemplo utilizando o *dataset* de vendas seria a conversão do valores monetários de Rupias para Reais, e uma vez que cada real vale 17 rupias isso poderia ser feito da seguinte forma:

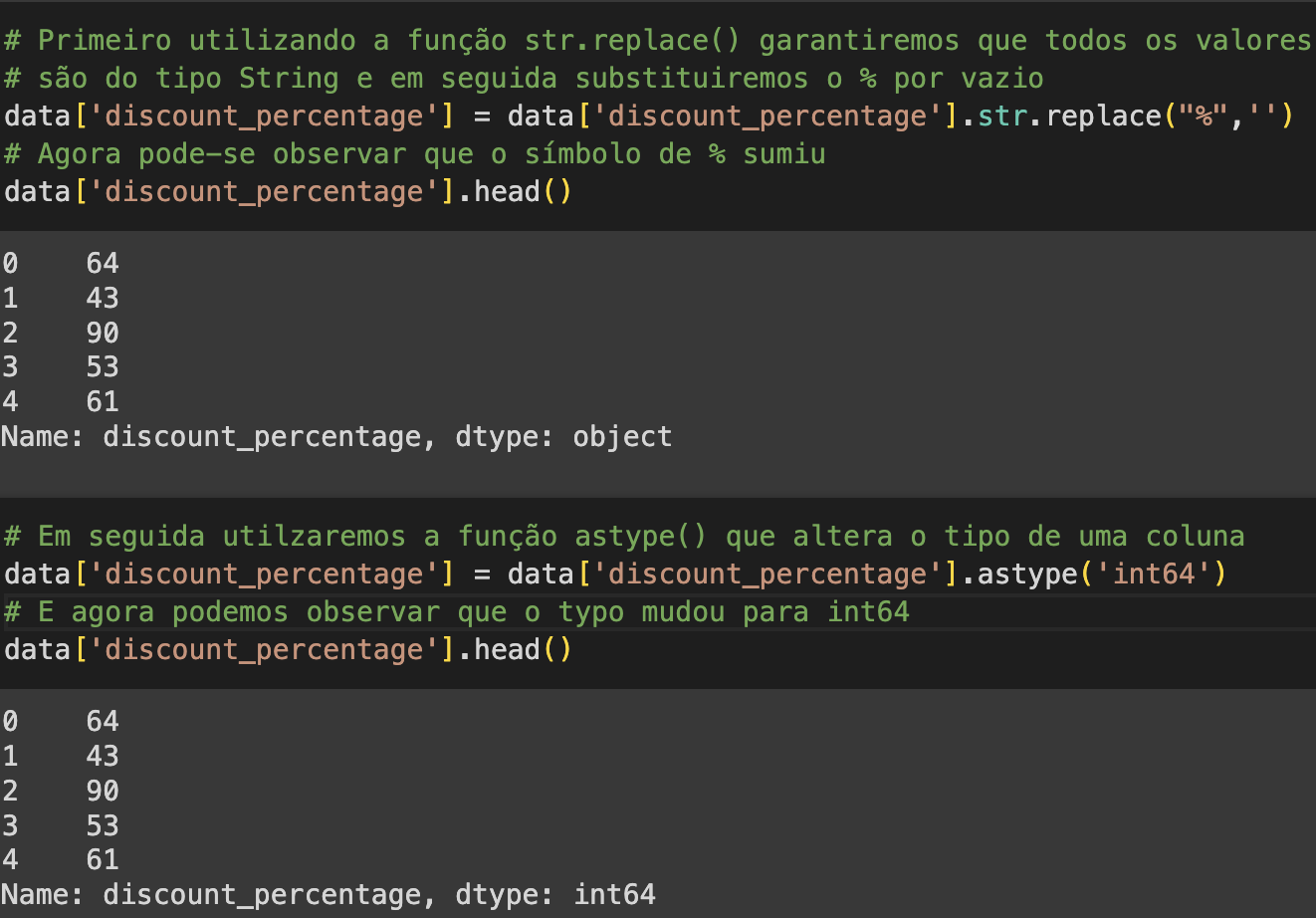
df['actual\_price\_reais'] = df['actual\_price'] / 17

### Limpeza e adequação de colunas

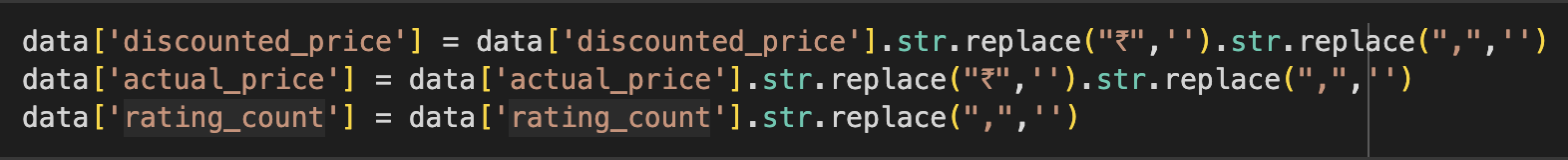
As colunas **discounted\_price**, **actual\_price**, **discount\_percentage**, **rating** e **rating\_count** possuem valores numéricos que podem ser interessantes para a obtenção de informações deste dataset, mas ainda precisam passar por uma limpeza. Inicialmente pode-se aplicar o método *info()* para entender melhor como estão essas colunas.



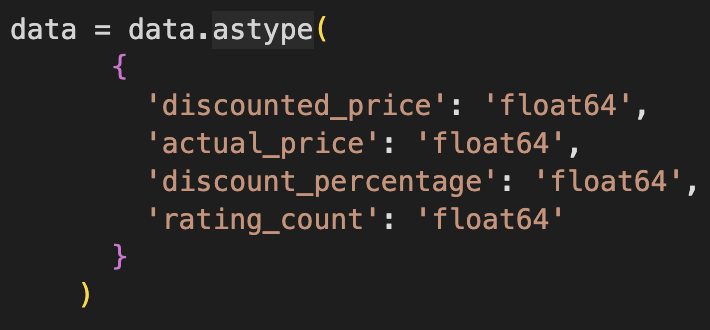
Todas as colunas são do tipo "object", o que não é interessante pois não existem muitas coisas que podem ser extraídas assim, principalmente tratando-se de dados numéricos. Então o primeiro passo será passar essas colunas para o tipo correto, porém observa-se na coluna **discount\_percentage** que os valores são acompanhados pelo símbolo "%", logo deve ser removido. Isso pode ser feito da seguinte forma:



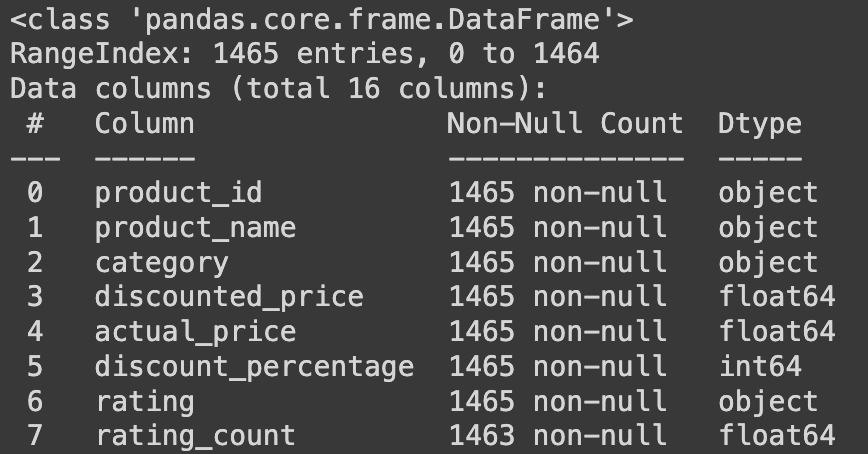
Nota-se em seguida que nas colunas **discounted\_price**, **actual\_price** e **rating\_count** que a divisão dos milhares é feita utilizando "," e não ".", podemos utilizar o método anterior para fazer essa correção, assim como apagar o símbolo ₹ (Rupias), da seguinte forma:



Após feita as correções pode-se mudar o tipo dessas colunas para float64, também utilizando a função *astype(),* mas faremos isso de uma vez passando um dicionário com o título da coluna e seu novo tipo:



Agora utilizando a função *info()* novamente pode-se observar que o *dataset* agora possui diversas colunas numéricas:



Em seguida pode-se retirar as colunas que não serão utilizadas da seguinte forma, isso deve ser feito levando em conta os propósitos que se busca com esses dados, a seguir vamos excluir **product\_name** considerando que **product\_id** já representa essa informação:

#removendo colunas desnecessárias

df\_clean = data[

[

'product\_id',

'actual\_price',

'discounted\_price',

'discount\_percentage',

'rating',

'rating\_count']

].copy()

## Filtrando DataFrames

Analogamente ao processo de consultas em tabelas de bancos de dados, a filtragem de DataFrames no pandas é uma abordagem semelhante para selecionar dados específicos. Em SQL, a cláusula *WHERE* é usada para extrair linhas com base em condições específicas. No pandas, a filtragem é realizada mediante a aplicação de condições lógicas para selecionar as linhas desejadas. Por exemplo, uma consulta SQL que seleciona todas as linhas onde o valor na coluna 'idade' é maior que 25 pode ser comparada a uma operação no pandas como *df[df['idade'] > 25]*.

Assim como em SQL, onde operadores de comparação, como =, >, <, >=, <=, são empregados para definir condições, no pandas, operadores lógicos como & (E), | (OU) e ~ (NÃO) são combinados para criar critérios de filtragem mais complexos.

Para ilustrar, considere um DataFrame de exemplo com informações sobre pessoas, incluindo nomes, idades e salários. A filtragem desse DataFrame para incluir apenas pessoas com idade acima de 25 e salário superior a 60000 pode ser expressa no pandas como *df[(df['Idade'] > 25)* & *(df['Salário'] > 60000)]*. O resultado seria um novo DataFrame contendo apenas as linhas que atendem a essas condições específicas.

## Redução de dimensionalidade

A primeiro momento, podemos pensar no porquê reduzir a dimensionalidade de nossas bases de dados, contudo, precisamos passar por essa etapa para evitar a maldição da dimensionalidade. A maldição da dimensionalidade refere-se aos desafios e problemas que surgem quando lidamos com conjuntos de dados de alta dimensionalidade. À medida que o número de dimensões aumenta, a quantidade de dados necessária para preencher o espaço torna-se exponencialmente maior e também a complexidade de treinamento se torna exponencialmente maior. Como podemos ver na imagem abaixo:

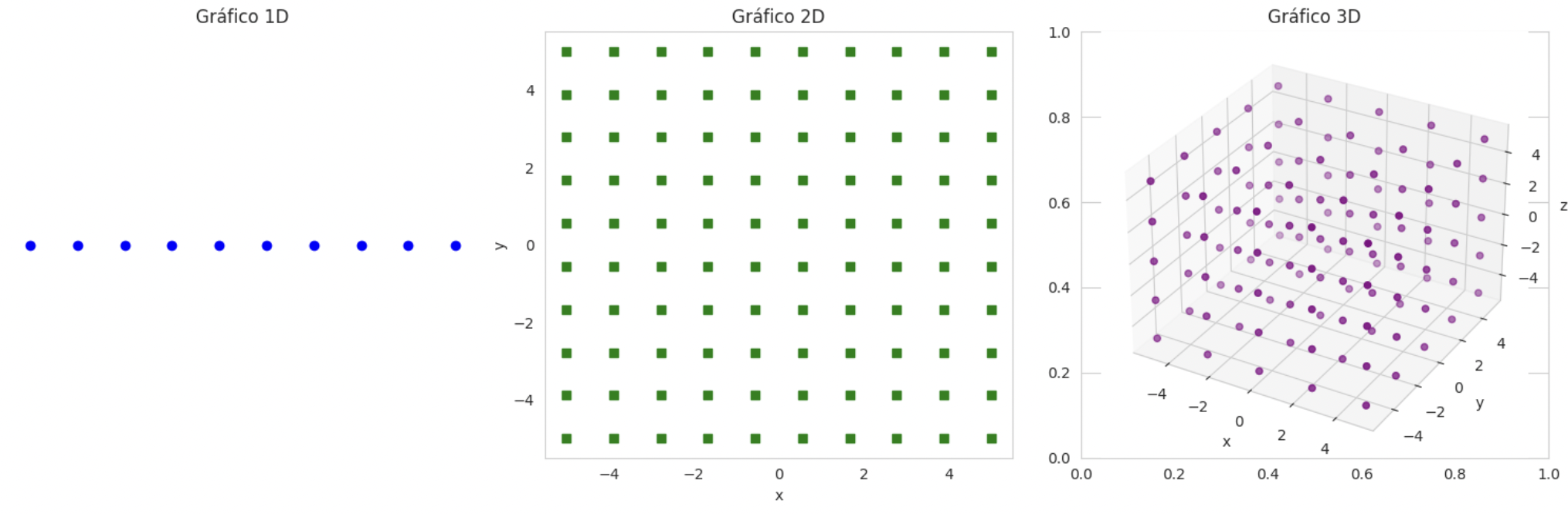


Imagem 10 - Exemplificação do aumento de dimensões

Na imagem 10, podemos ver um exemplo de como o aumento de dimensões influencia no tamanho e na complexidade da base de dados. No primeiro gráfico, vemos um exemplo com 1 dimensão, onde é apenas uma linha. No segundo gráfico, podemos visualizar uma base de dados em duas dimensões, o plano cartesiano,. No terceiro gráfico, podemos ver uma figura tridimensional no , onde sua complexidade é a maior das três, apenas aumentando a dimensão. Tendo esse problema em vista, tornou-se necessário cada vez mais técnicas de redução de dimensionalidade.

A redução da dimensionalidade é uma tarefa relacionada na qual o objetivo é simplificar os dados sem perder muita informação. Uma maneira de fazer isso é mesclar várias características correlacionadas em uma. Por exemplo, a quilometragem de um carro pode estar muito relacionada com seu tempo de uso, de modo que o algoritmo da redução de dimensionalidade irá mesclá-los em uma característica que representa o desgaste do carro. Isso é chamado de extração de características. Isso não apenas facilita a visualização e interpretação dos dados, mas também pode melhorar o desempenho dos modelos, tornando-os menos propensos ao *overfitting*.

*Overfitting* ocorre quando um modelo se ajusta excessivamente aos dados de treinamento, capturando padrões que podem não ser generalizáveis para novos dados. Isso é especialmente problemático em conjuntos de dados de alta dimensionalidade, onde modelos podem se tornar excessivamente complexos. A redução de dimensionalidade pode ajudar a mitigar o *overfitting* ao simplificar a representação dos dados.

*Underfitting*, por outro lado, ocorre quando um modelo é muito simples para capturar os padrões presentes nos dados. A seleção adequada de variáveis e a engenharia de features podem ajudar a abordar o underfitting, garantindo que o modelo tenha informações relevantes para fazer previsões precisas.

Para fazer essa redução de dimensionalidade, envolve métodos de seleção de features. Métodos de seleção de features fazem a escolha das variáveis mais relevantes para um problema específico. Esses métodos podem ser univariados, avaliando cada variável independentemente, ou multivariados, considerando interações entre variáveis. A seleção adequada de features não apenas simplifica os modelos, mas também pode melhorar a interpretabilidade e o desempenho preditivo.

Essa parte, é uma área da engenharia de features para buscar o melhor dimensionamento e melhor comportamento. É o processo de criação de novas variáveis a partir das existentes, com o objetivo de melhorar o desempenho do modelo. Isso pode envolver a combinação de variáveis, a criação de variáveis *dummy* (Variável *dummy* é uma variável binária, que representa a presença ou ausência de uma característica categórica) para representar categorias, a normalização de dados, entre outras técnicas.

Há várias técnicas matemáticas para redução de dimensionalidade:

* Análise de Componentes Principais (PCA)
* Decomposição de Valor Singular (SVD, às vezes também chamada de LSA)
* Análise de Componentes Independentes (ICA)
* Factorização de Matriz Não-Negativa (NMF)
* t-SNE e UMAP (melhores para estruturas não lineares, como linguagem natural e imagens)
* Autoencoders (uma técnica de redes neurais que acaba sendo semelhante ao PCA)

Cobrir todos esses métodos é inviável no curso, assim, vamos focar apenas no entendimento e aplicação do PCA.

## Análise dos Componentes Principais

O PCA, ou Análise de Componentes Principais (em inglês, *Principal Component Analysis*), é uma técnica estatística utilizada para simplificar conjuntos de dados de alta dimensionalidade, preservando as informações mais importantes enquanto reduz a complexidade do conjunto de dados. Transformando um conjunto de variáveis correlacionadas em um conjunto de variáveis não correlacionadas.

Para entendermos o que é PCA, primeiramente, precisamos entender o que significa variância, covariância, autovetor e autovalor.

* Variância

A variância é uma medida estatística que representa a dispersão ou a extensão dos valores em um conjunto de dados. Em outras palavras, a variância indica o quão longe os valores individuais de uma média. Se os valores estiverem todos próximos à média, a variância será baixa; se os valores estiverem espalhados em torno da média, a variância será alta

* Covariância

A covariância é uma medida que indica como duas variáveis diferentes variam juntas. Se a covariância é positiva, significa que, em geral, quando uma variável aumenta, a outra também tende a aumentar. Se a covariância é negativa, significa que quando uma variável aumenta, a outra tende a diminuir. Se a covariância é próxima de zero, não há uma relação linear clara entre as duas variáveis.

* Autovalores e Autovetores

Os autovalores são números que escalam os autovetores durante uma transformação linear. Em termos mais simples, quando multiplicamos uma matriz por seu autovetor, o resultado é o mesmo autovetor, mas escalado pelo autovalor correspondente.

Os autovetores são vetores que mantêm a mesma direção (ou uma direção linear) durante uma transformação linear, embora possam ser escalados pelo autovalor associado.

Tendo isso em mente, podemos mostrar o passo a passo do PCA:

1. Calcula-se a média e normaliza-se os dados
2. Calcula-se uma matriz de covariância
3. Calcula-se os autovetores e autovalores da matriz de covariância
4. Escolhem uma quantidade de autovetores com maior quantidade de informação (Os autovalores associados expressam a quantidade de informação)
5. Monta-se a matriz de projeção baseado nos autovetores selecionados
6. Projeta-se a imagem normalizada obtida na etapa 1 pela matriz de projeção obtida na etapa 5
   1. ( representa a média)
7. Assim, o vetor de dimensão será a nova representação do padrão original .

Podemos aplicar o PCA utilizando a biblioteca [Scikit-learn](https://scikit-learn.org/stable/), além do PCA essa biblioteca possui diversas outras ferramentas para análise de dados e predição. Podemos importar a classe PCA da seguinte forma:

from sklearn.decomposition import PCA

E aplicá-la assim:

pca\_obj = PCA(n\_components = 0.9,whiten=True)

pca\_result = pca\_obj.fit\_transform(X)

Onde, ***n\_components*** representa funciona da seguinte forma: caso ***n\_components*** esteja entre 0 e , representará a porcentagem que deve ser preservada do dataset original com base na variância. Caso ***n\_components*** seja maior que 1 representa explicitamente o total de colunas do novo dataset. E se não for definido o número de dimensões se manterá os mesmos.

Em seguida o método ***fit\_transform()*** recebe as colunas numéricas do dataset original e retorna uma numpy array de dimensão Y igual ao número de entradas e dimensão X igual o número de dimensões atual.

**Considerações finais**

Dessa forma terminamos o conteúdo do curso de Inteligência Artificial. Nele foi possível se aprofundar nessa área, vendo desde a preparação dos dados, as formas e estratégias de treinamento, o funcionamento e criação de redes neurais, as tecnologias utilizadas na área e os principais algoritmos utilizados atualmente.

Esperamos que a partir dos desafios e projetos desenvolvidos possa se levar uma experiência real de como se desenvolve a Inteligência Artificial atualmente. Assim como entender a importância dessas tecnologias atualmente, automatizando tarefas repetitivas ou solucionando problemas práticos da realidade.

Ter esses conhecimentos pode abrir um futuro novo, pois é uma área com muitas possibilidades e que avança constantemente, cada vez mais presente no nosso cotidiano. Logo, dominar seu desenvolvimento pode abrir novas oportunidades profissionais e pessoais.

**Referências**

FAYYAD, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). **From data mining to knowledge discovery in databases.** AI Magazine, 17(3), 37–54.

GÉRON, A. (2022). **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow**. Porto Alegre: Editora Artmed.

HAYKIN, Simon S. **Neural Networks and Learning Machines**. 3rd ed. Pearson Education, 2009.

WITTEN, I. H.; FRANK, E.; HALL, M. A.; PAL, C. J. **Data Mining: Practical Machine Learning Tools and Techniques.** 4. ed., Morgan Kaufmann, 2016.

**BOM CURSO!**