APRENDIZADO DE MÁQUINA COM PYTHON

Navegação

Introdução a programação com Python

Conteúdos

- Porque o Python?
- Instalação (Windows)
- Entendendo melhor o print
- Variáveis
- Tipos de dados
- Operandos
- Métodos e funções de tipos
- Comentários

Porque o python?

Python é uma linguagem de programação rápida de se programar dado a sua simplicidade e versatilidade. Abstrações de detalhes técnicos do computador são usadas na linguagem para auxiliar o programador no desenvolvimento de seus programas, retirando recursos presentes em outras linguagens a fim de manter apenas o essencial ao programador.

Em outras linguagens o programador está sempre preocupado se está usando a tipagem correta, se está compilando os programas com as referências externas corretas, se os buffers de input estão limpos, se a memória alocada é suficiente, tudo isso para receber uma falha de segmentação depois de esquecer um ; em algum lugar.

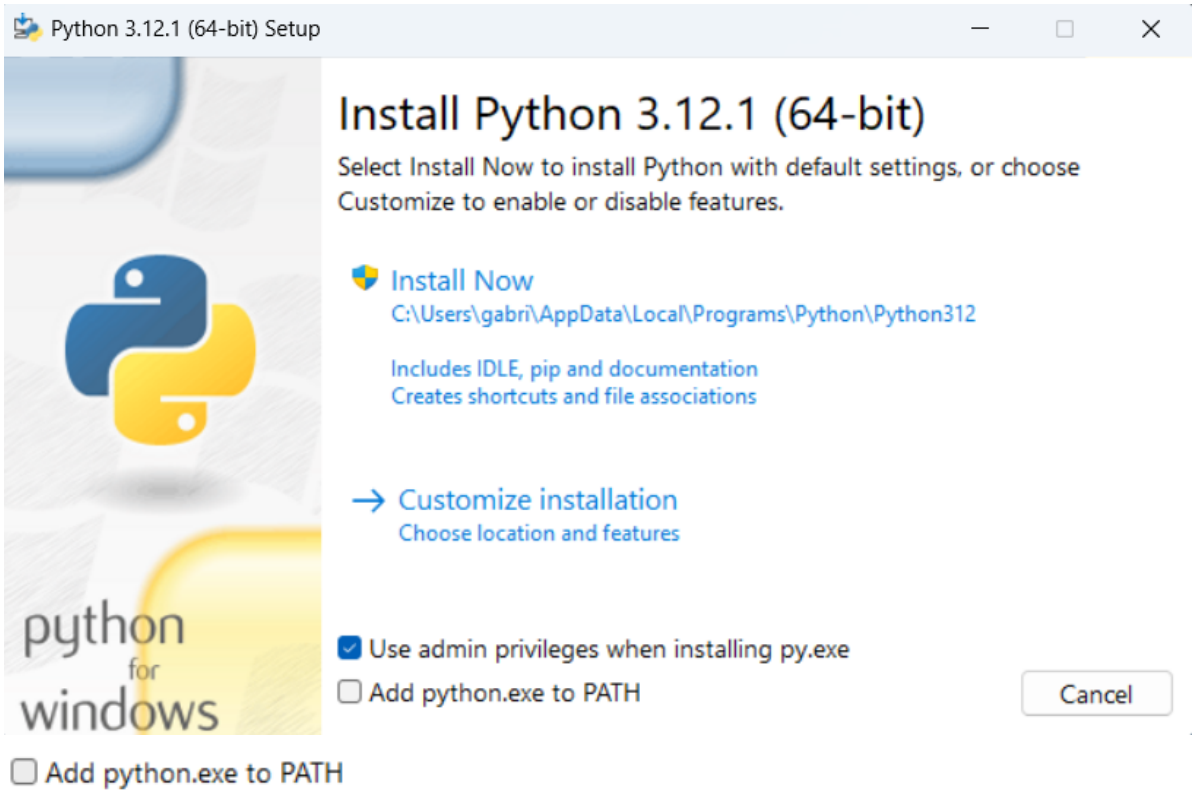
Mas o que é tudo isso? Muitos nomes difíceis de conceitos que parecem ainda mais difíceis. No Python nada disso é necessário. Python simplifica o processo de programar. Não que esses processos difíceis deixem de existir, mas eles são feitos de maneira automática pelo próprio Python. Contudo, isso tem um custo, Python é conhecido por ser uma linguagem de processamento lento e para certas aplicações que exigem boa performance, como um videogame, existem alternativas melhores. Essa simplicidade do Python faz com que os códigos possam ser desenvolvidos mais rapidamente. Aplicações que são focadas para resolver um problema específico, como a ciência de dados, usam a programação como o meio de resolver e não com o objetivo de executar o programa da maneira mais rápido possível, portanto aproveitam esse aspecto da linguagem. O programador pode focar no que importa de verdade que é a entrega final.

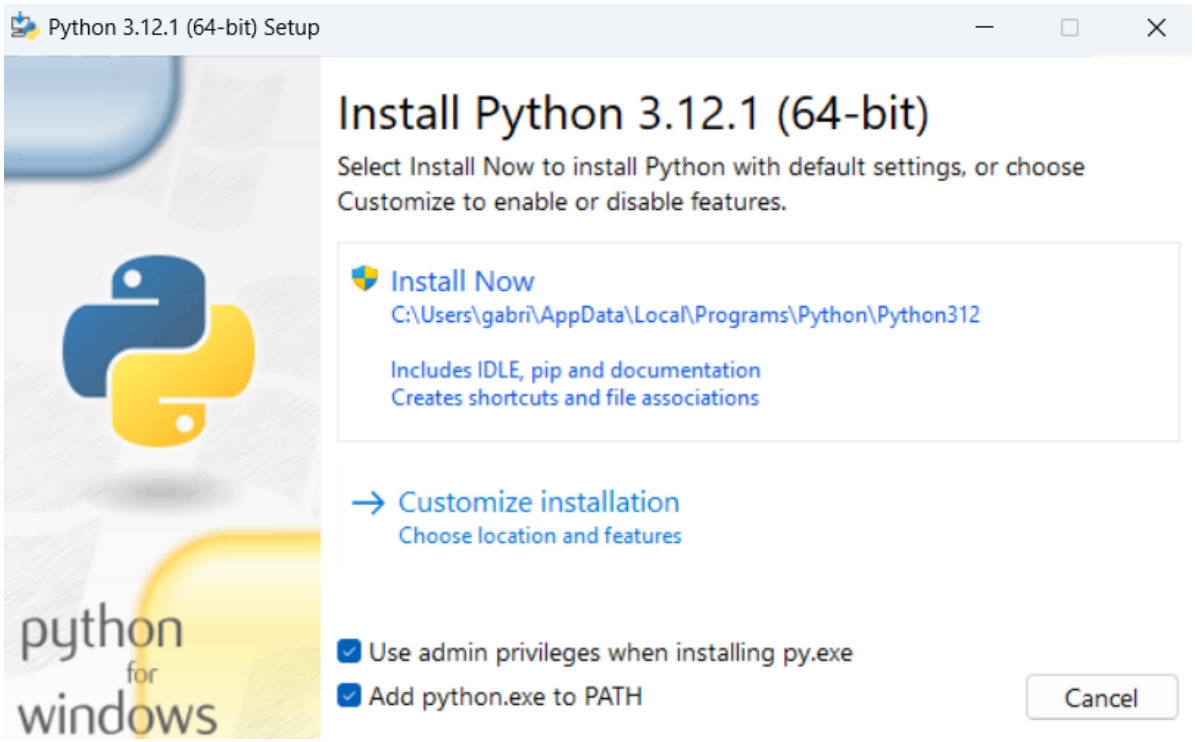
Outro ponto que explica o sucesso do Python é a sua comunidade. Milhares de programadores contribuem para o ecossistema de desenvolvimento com soluções que antes não eram contempladas originalmente. Desde ferramentas de criação de sites até inteligência artificial, o leque de opções do Python é vasto. Para ciência de dados as ferramentas pandas e numpy, desenvolvidas pela comunidade, são imprescindíveis.

Por essas e outras razões, Python é uma das principais linguagens de programação da atualidade.

Instalação (Windows)

Para instalar o Python precisa acessar o link https://www.Python.org/downloads/ e fazer o download da versão mais recente. Quando o download estiver concluído, execute o instalador baixado, marque a opção Add Python.exe to PATH e clique no install now, seguindo as opções padrões do Python.





Acabamos de instalar o interpretador do Python, que é o programa responsável por dar ao computador capacidade de executar os programas em Python. Todos os programas em Python são executados com o interpretador, o computador entende naturalmente apenas códigos binários, o interpretador é como se fosse um tradutor para o computador.

Primeira execução do Python

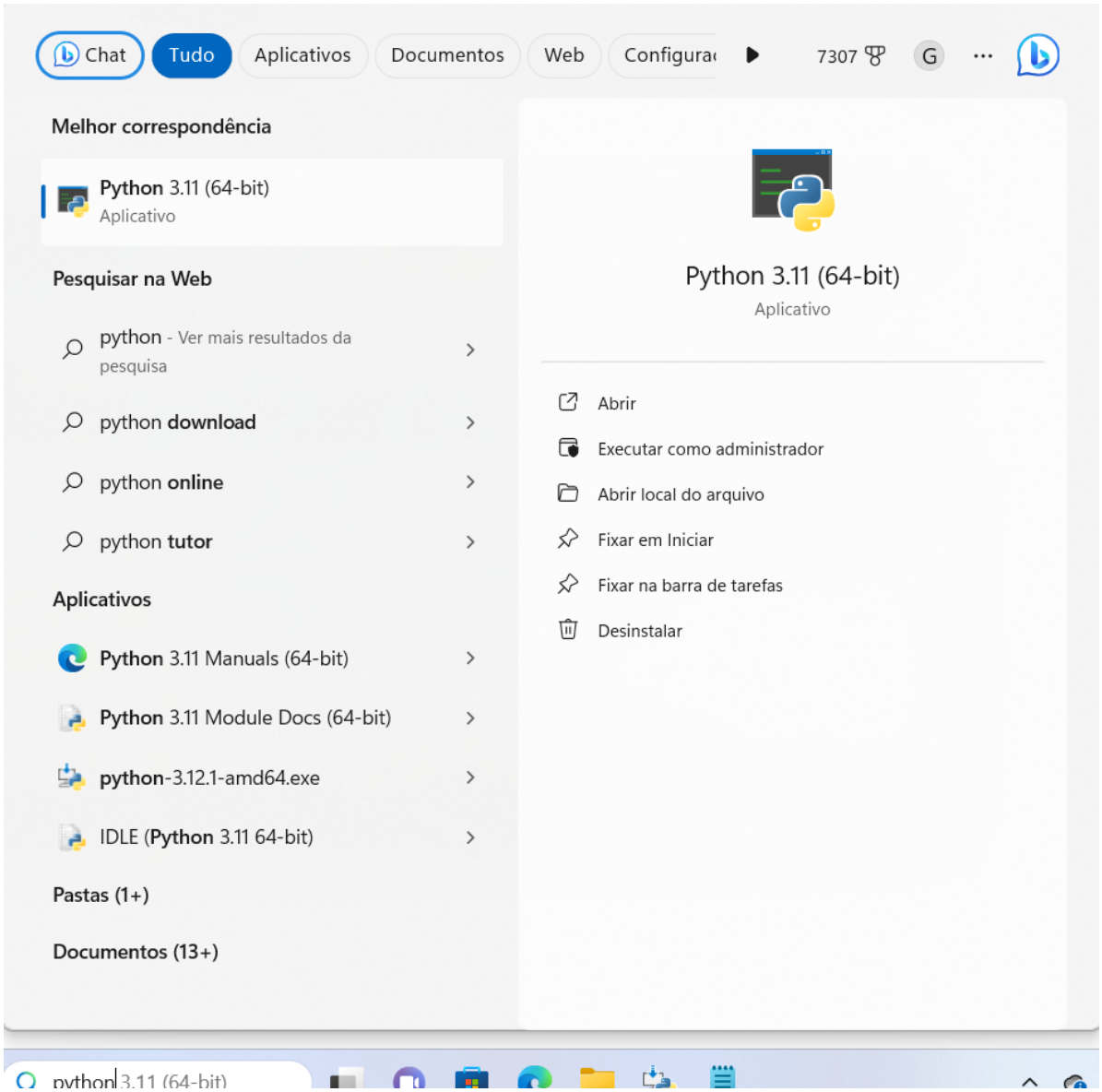
Tradicionalmente na programação o primeiro código executado quando se instala uma linguagem nova, tanto para aprender quanto em uma nova máquina, é o Hello World. Um programa simples que diz apenas Hello World. O código para isso em Python é:

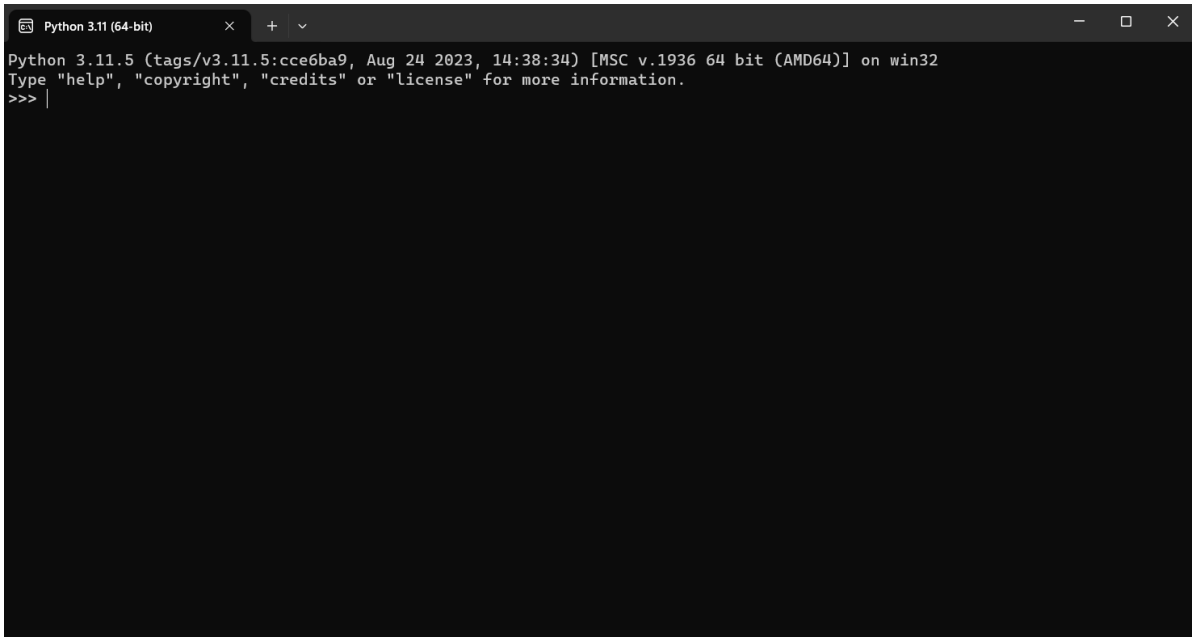
```
print("Hello World")
```

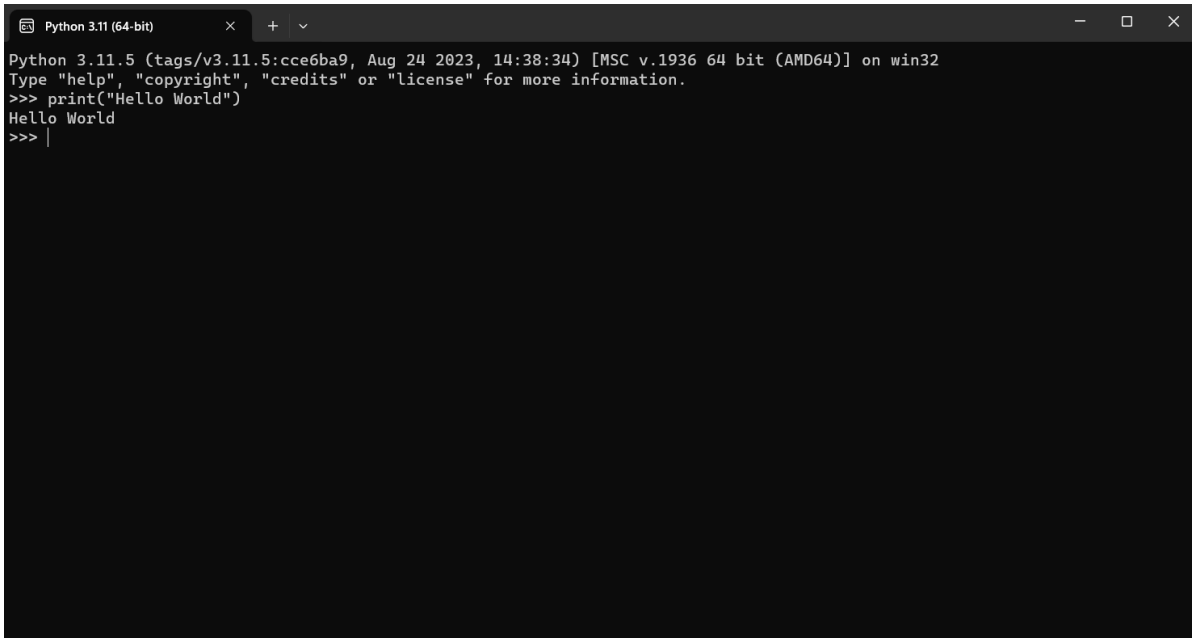
Porém como que diremos para o Python fazer isso? E aonde que ele dirá esse “Hello World”? Ele não tem uma interface visual que nem o Excel que você coloca numa célula uma fórmula que o excel executa.

Existem duas maneiras de executar um programa no Python: diretamente pelo interpretador e através de um arquivo.

Para executar diretamente pelo interpretador, pesquise no menu iniciar por Python e abra o aplicativo do Python. Digite o código em Python que quiser depois do »> e aperte enter.







Execução no interpretador não é muito importante porque não podemos reproduzir o que fizemos de maneira automática. Precisamos reescrever de cabeça cada linha de código que escrevemos anteriormente. A outra alternativa de execução de um código em Python é via um arquivo, onde escrevemos os códigos sequencialmente e eles serão todos executados pelo interpretador em ordem, assim manteremos registro do que foi executado no arquivo que criamos, facilitando ainda sua edição.

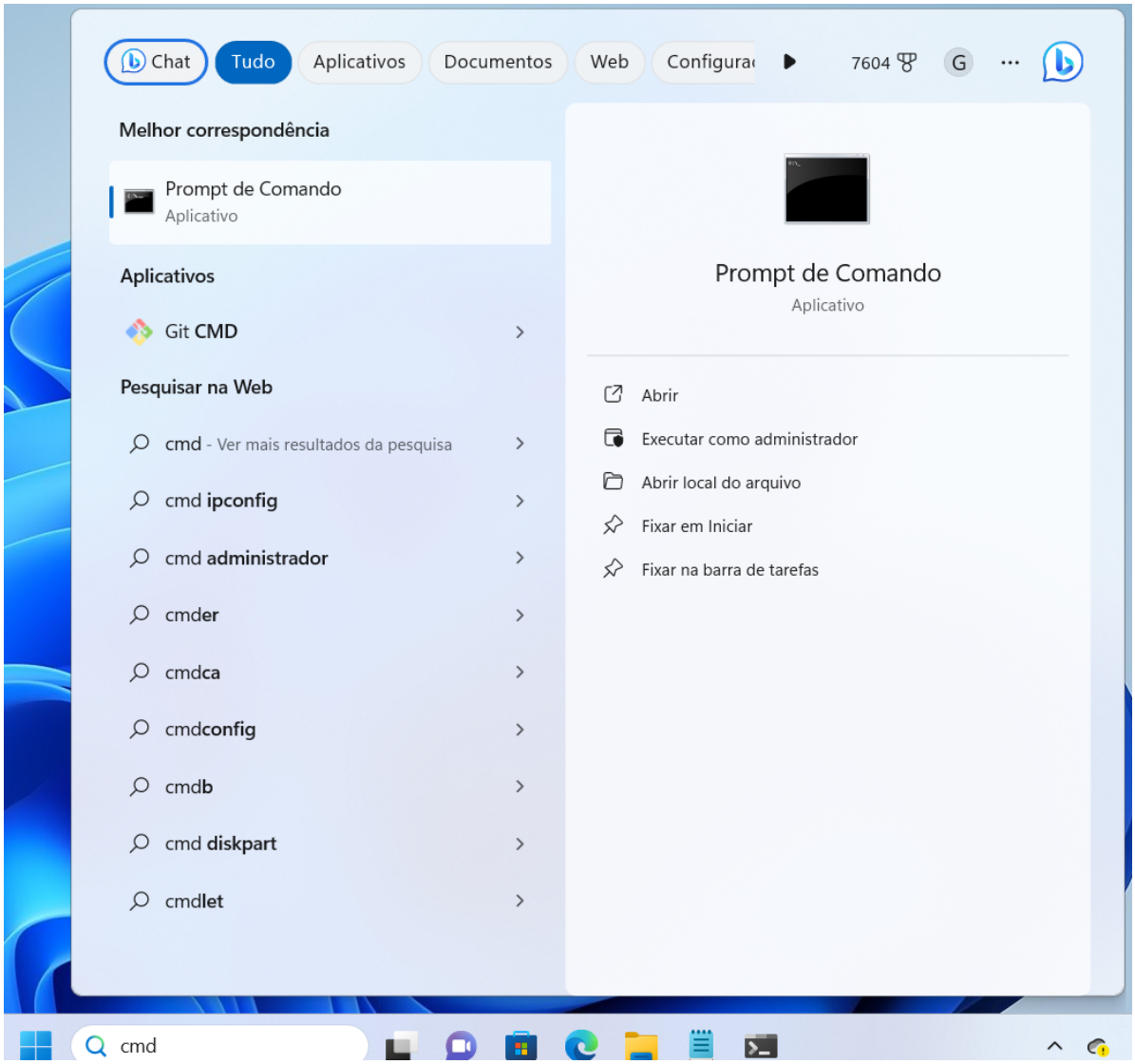
Para executar um código em Python presente em um arquivo precisamos cria-lo com extensão .py e chamar o interpretador para ler o arquivo. Isso pode ser feito rusticamente pelo terminal de comando do seu sistema operacional. Alguns programadores executam códigos apenas dessa maneira. Para quem está iniciando na programação, esse processo é uma dificuldade que pode atrapalhar no processo de aprendizagem.

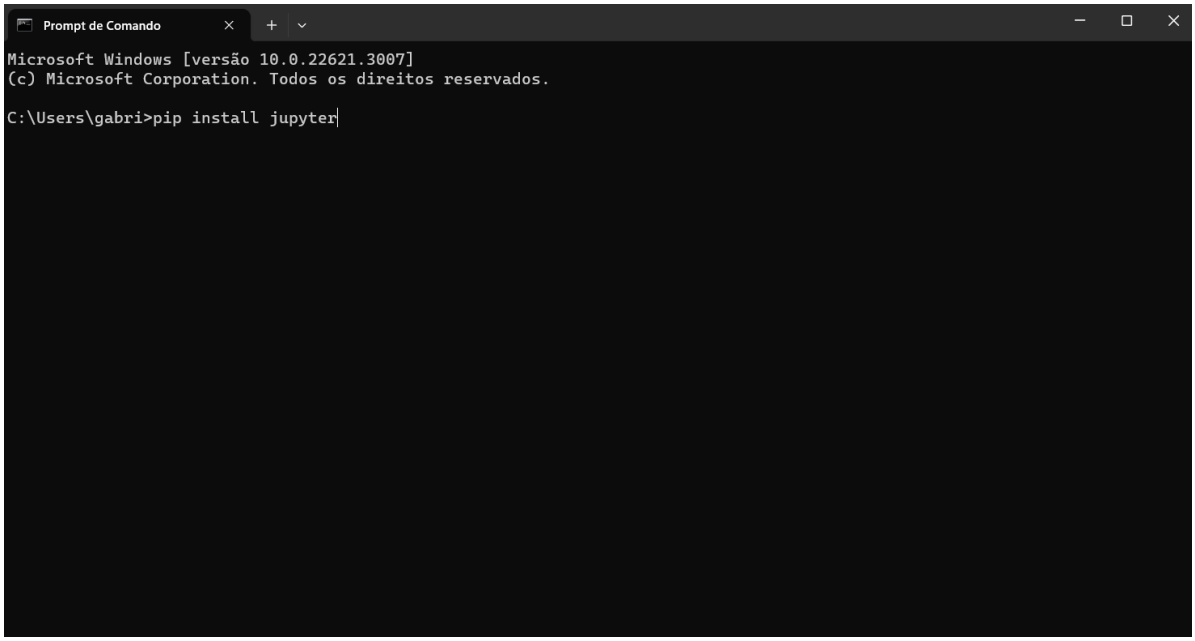
IDEs

Existem ferramentas desenvolvidas pela comunidade, algumas até por grandes empresas, com a finalidade de simplificar o processo de se programar em Python. Todas as linguagens de programação possuem alguma ferramenta com essa finalidade. O nome desse tipo de ferramenta é IDE, uma sigla em inglês que significa Integrated Development environment que traduzindo ao português é ambiente de desenvolvimento integrado. As funcionalidades mais comuns de uma IDE é a marcação de texto, que ajuda a legibilidade do código destacando palavras chave da linguagem, explorador de arquivos, que auxilia na criação, remoção e alteração de arquivos, e terminal de comando integrado, um espaço para executar os programas com apenas um botão. A maioria dos programadores profissionais usam alguma IDE. Para a ciência de dados a mais utilizada por desenvolvedores em Python é o Jupyter.

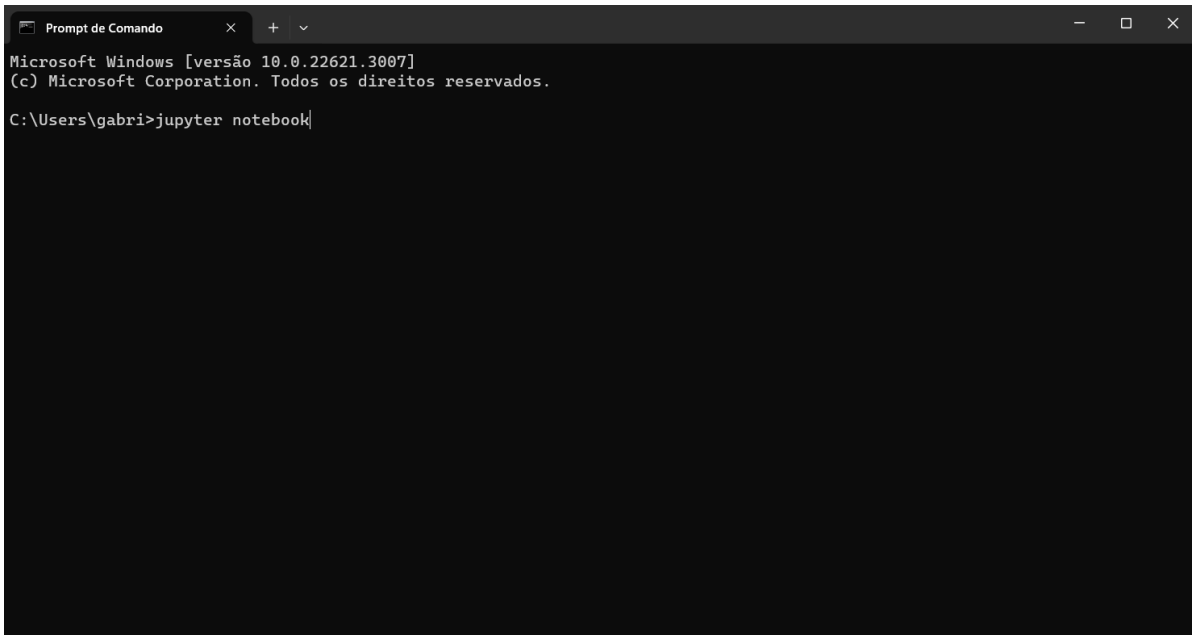
Instalação e execução do Jupyter

Para instalar o Jupyter precisaremos usar o pip, uma ferramenta do Python que permite a instalação de complementos a linguagem mais a frente exploraremos esse assunto melhor. No espaço de pesquisar do windows digite cmd e aperte enter. O cmd é o terminal de comandos do windows, a interface (ou a falta dela) pode assustar assustar um pouco, para o que queremos é só digitar pip install Jupyter e apertar enter, após o comando ser inserido, o terminal irá instalar a extensão do jupyter ao Python. Uma vez instalado o jupyter não precisa ser reinstalado novamente quando for usado.


Chat
Tudo
Aplicativos
Documentos
Web
Configura
7604
G
Melhor correspondência
Prompt de Comando
Aplicativo
Aplicativos
Git CMD
Pesquisar na Web
cmd - Ver mais resultados da pesquisa
cmd ipconfig
cmd administrador
cmder
cmdca
cmdconfig
cmdb
cmd diskpart
cmdlet
Prompt de Comando
Aplicativo
Abrir
Executar como administrador
Abrir local do arquivo
Fixar em Iniciar
Fixar na barra de tarefas
cmd

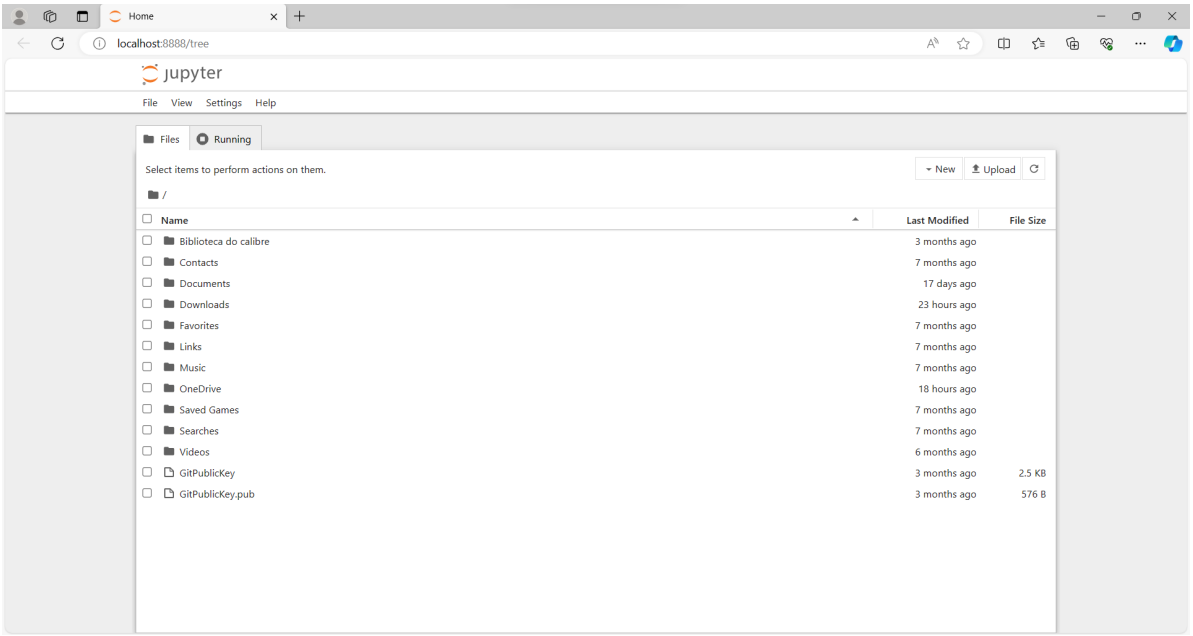


Para cada vez que formos abrir o Jupyter precisamos abrir o terminal de comando, pesquisando cmd na aba de pesquisa do windows e apertando enter, e digitar jupyter notebook e apertar enter. Ao fazer isso, o navegador padrão irá abrir uma aba do Jupyter.



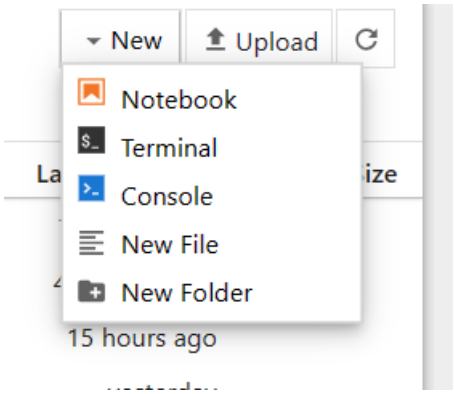
Importante não fechar o prompt de comando que executou o Jupyter, se não o Jupyter para de funcionar.

Hello World no Jupyter

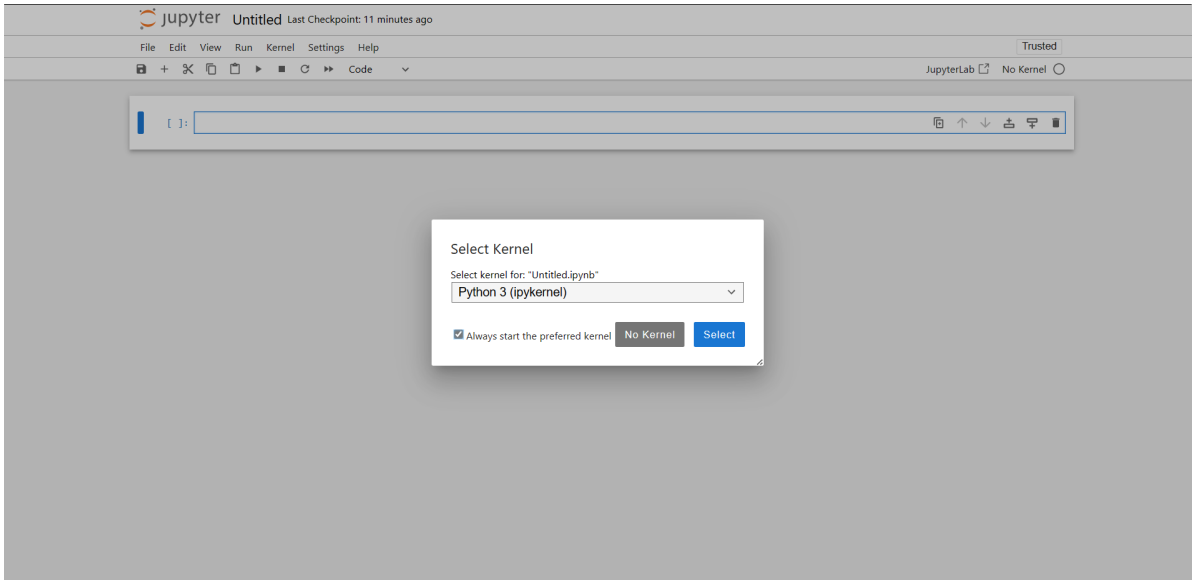


A primeira tela do Jupyter é o explorador de arquivos do seu sistema operacional (no meu caso Windows 11). Aqui você deve selecionar a qual pasta o projeto criado será salvo. Criei a pasta Aprendendo Python. O local ou o nome da pasta que salvará o projeto não é importante, pode ser salva inclusive no diretório inicial que aparece no Jupyter (embora seja recomendado criar uma pasta nova para cada projeto para uma melhor organização).

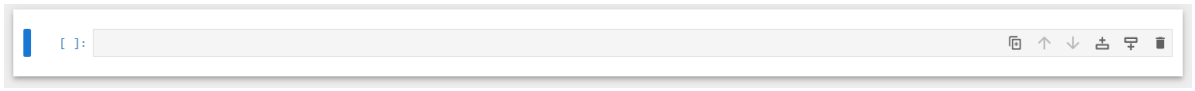
Para criar uma pasta ou um arquivo novo no Jupyter basta selecionar o local desejado no explorador de arquivos clicando duas vezes nos caminhos possíveis e depois apertando o botão New e a opção New Folder. Quero criar minha pasta do projeto dentro do meu OneDrive então cliquei duas vezes na pasta OneDrive depois apertei o botão New e selecionei a opção New Folder. Para renomear, deletar, copiar e outras funcionalidades padrões é só apertar com o botão direito em cima da pasta. Cliquei duas vezes na pasta Aprendendo Python que criei e usando uma outra opção do botão New chamada Notebook criei o arquivo que escreveremos nossos programas em Python a partir de agora.

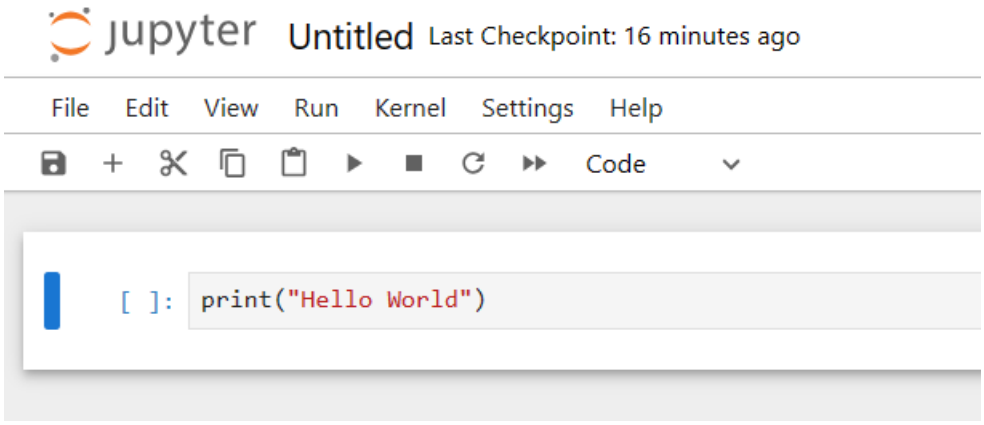


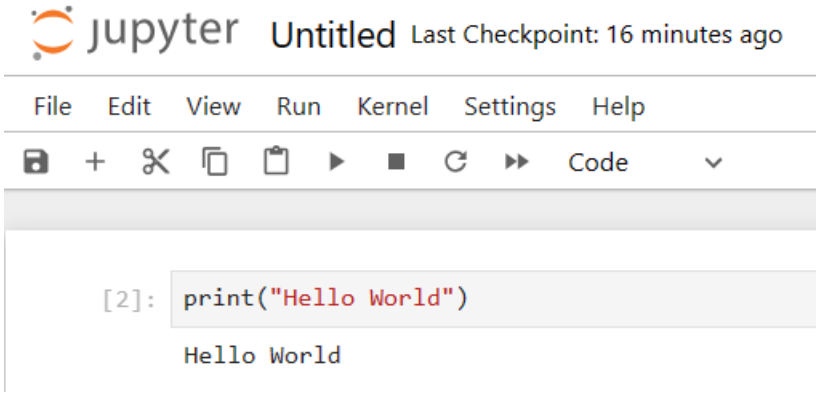
Automaticamente o Jupyter abrirá uma aba no navegador escrita Select Kernel, selecione a opção "Always start with the preferred kernel" e aperte a opção Select (o Jupyter está perguntando qual versão do Python usar, alguns programadores podem ter mais de uma versão do Python no computador).



Agora estamos finalmente no arquivo que executará programas em Python. Escrevendo `print("Hello World")` no arquivo e apertando o botão de start o código será executado.







Podemos ainda executar vários comandos de uma vez só. Veja esse exemplo:

```
[3]: print("Hello World")
     print("Esse")
     print("É meu segundo programa")
     print("Em python")
```

```
Hello World
Esse
É meu segundo programa
Em python
```

```
[ ]:
```

Entendendo melhor o print

O print é uma das funções mais importantes do Python por ser a principal ferramenta de comunicação entre o programa, o programador e o usuário. Linguagens de programação quase sempre tem os comandos no imperativo da língua inglesa, já que os ingleses e posteriormente os americanos foram pioneiros no desenvolvimento da ciência da computação. Print significa literalmente "imprimir".

O print imprime na tela o conteúdo entre parênteses na tela. Se o conteúdo for textual precisa estar entre aspas. Números podem ser imprimidos diretamente sem o uso de aspas. O print por configuração padrão imprime as mensagens pulando uma linha depois. Para mudar essa configuração em um print específico, no final do comando, antes de fechar os parênteses, precisamos por `, end=""`. Agora para esse print específico uma linha não será pulada no final da execução:

```
print("Hello World")
print("Esse", end="")
print("é meu terceiro programa", end="")
print("em Python")
```

```
Hello World
Esse
é meu terceiro programa
em Python
```

Variáveis

Na escola quebramos a cabeça em matemática para resolver uma equação. A professora Márcia que todos tivemos passa uma expressão: $4 + x = 7$. Mesmo como crianças sabíamos que para o 4 virar 7 o x precisaria ser um 3, o x armazena o valor 3. Na programação existem inúmeras vantagens de se usar variáveis.

Voltando para o exemplo do print, digamos que queremos imprimir uma mesma mensagem inúmeras vezes:

```
print("Ciência de dados!")
print("Ciência de dados!")
print("Ciência de dados!")
print("Ciência de dados!")
```

```
Ciência de dados!
Ciência de dados!
Ciência de dados!
Ciência de dados!
```

Agora quero invés de imprimir "Ciência de dados!" quero imprimir "O meu filme favorito é troll 2!". Para fazer isso teríamos que trocar cada uma das vezes o texto dentro do print pelo texto desejado. Existe alguma forma mais rápida de fazer isso: usando variáveis. Podemos criar uma variável x, que nem das aulas de matemática da professora Márcia que armazena o texto a ser impresso.

```
x = "Ciência de dados!"
print(x)
print(x)
print(x)
print(x)
```

```
Ciência de dados!
Ciência de dados!
Ciência de dados!
Ciência de dados!
```

Reparem que ambos os programas realizam a mesma tarefa, de maneiras diferentes. A vantagem do segundo programa é que para mudar o texto impresso para "O meu filme favorito é troll 2!" basta trocar o valor de x:

```
x = "O meu filme favorito é troll 2!"
print(x)
print(x)
print(x)
print(x)
```

```
O meu filme favorito é troll 2!
O meu filme favorito é troll 2!
O meu filme favorito é troll 2!
O meu filme favorito é troll 2!
```

Escolhi a variável x para realizar essa tarefa apenas para seguir a didática da professora Márcia. Porém, idealmente a nomeação de variáveis se faz baseado no uso dela, isso é, idealmente criamos variáveis com nomes elusivos. Nesse caso poderia invés de ser x ser texto. Isso facilita em projetos maiores quando várias pessoas trabalham juntas ou quando você precisa rever um código antigo seu.

Para criar uma variável basta colocar o nome que gostaria no início, seguido de = e depois o valor que gostaria de armazenar. O computador então passa a entender que sempre que você escreve texto no código é para substituir pelo valor "O meu filme favorito é troll 2!".

```
nomeDaVariavel = valorDaVariavel
```

O Python possui algumas regras para nomeação de variáveis. Assim como todas as linguagens de programação, existem alguns comandos que fazem uma tarefa, como o print. Esses comandos ou como chamamos na programação, funções, possuem o status de palavra reservada, que não podem ser usadas para outras coisas, entre outras palavras. Outras regras de nomeação de variáveis no python é que as variáveis não podem começar com números (podem ter números, só não como primeiro caracter), não podem ter . e nem espaços.

Quando criamos uma variável dizemos que x recebe o valor "O meu filme favorito é troll 2!" ou que atribuímos "O meu filme favorito é troll 2!" a x. Podemos também simplesmente dizer que x é igual a "O meu filme favorito é troll 2!".

Tipos de dados

No exemplo anterior, x recebeu um texto, “O meu filme favorito é troll 2!”. Poderíamos também simplesmente ter colocado um número:

```
x = 4
```

Note que para colocar um número não precisa colocar entre aspas. Essa diferença se dá no tipo do dado atribuído, um conceito importantíssimo em linguagens de programação. Enquanto “O meu filme favorito é troll 2!” é um texto, 4 é um número.

Mas poderíamos também ter feito x = “4”? Então o que seria? Nesse caso seria um texto.

O que define se um dado é do tipo texto é se ele está entre " " e um dado do tipo numérico é um número que não está entre aspas. Mas então qual a diferença?

```
x = 4
y = "4"
print(x)
print(y)
```

```
4
4
```

Pelo print parece a mesma coisa. A diferença seria no comportamento que essas variáveis vão ter no programa. Digamos que eu queira somar duas variáveis. Para isso utilizamos operadores matemáticos. O que se espera que será o resultado de x + x? 8, certo? E de y + y? Não será 8.

```
x = 4
y = "4"
print(x + x)
print(y + y)
```

```
8
44
```

Textos, ao invés de somarem, concatenam quando usamos o operador `+`. Falaremos mais de operadores em breve.

Quando criamos uma variável nova, estamos criando uma instância de um tipo, ou instanciando um tipo.

Cada tipo possui características próprias, que vão além de apenas o comportamento frente a operadores. Forma de instanciar, memória do computador usada, uso em funções, cada uma dessas características definem um `tipo`.

Função type()

Antes de entrar mais especificamente em cada tipo, precisamos falar de uma ferramenta importante. Existe uma função no python que retorna o tipo de uma variável, o `type()`, que quando usado com o print imprime o tipo na tela. Veja um exemplo:

```
x = 4
print(type(x))
```

```
class 'int'
```

Repare que uma função pode ser usada dentro de outra. Falaremos mais de funções em breve.

Sabemos que se trata de um int, afinal, nós criamos essa variável. Porém, como muitas coisas na programação, quando se aumenta a escala saber o tipo específico de uma variável em um programa maior ou de outra pessoa pode ser um pouco mais difícil.

Dito isso, podemos agora falar dos principais tipos.

Principais tipos de variáveis

int

Abreviação para `integer`, que traduzido do inglês significa inteiro. São equivalentes a números inteiros da matemática, podendo ser positivos ou negativos e se comportam igual na matemática.

Instanciação

Sempre que colocamos números sem estar entre aspas estamos instanciando um int:

```
x = 4
print(x)
print(type(x))
```

```
4
class 'int'
```

str

Abreviação de `string`, que traduzido do inglês significa fio, corda ou barbante, mas quando usado na programação significa cadeia, uma cadeia de caracteres. Linguagens de programação tradicionalmente possuem o tipo char, que é uma abreviação de characters que traduzido do inglês é caracteres. Esse tipo armazena apenas um caractere textual, o conjunto desses caracteres que é chamado de string. No python, para simplificar, todo dado textual é `str`.

Instanciação

Sempre que colocamos caracteres entre aspas simples (’ ’) ou duplas(“ “). Podemos também usar”“ ” “ “ “. Esse último é usado para quando queremos escrever um texto em mais de uma linha de código no python. Segue abaixo exemplos de instanciação:

```
texto1 = "Ciência de dados!"
texto2 = 'O meu filme favorito é "troll 2"!'
texto3 = """Minha terra tem palmeiras
Onde canta o Sabiá,
As aves, que aqui gorjeiam,
Não gorjeiam como lá."""

print(texto1)
print(type(texto1))
print(texto2)
print(type(texto2))
print(texto3)
print(type(texto3))
```

```
Ciência de dados!
class 'str'
O meu filme favorito é "troll 2"!
class 'str'
Minha terra tem palmeiras
Onde canta o Sabiá,
```

```
As aves, que aqui gorjeiam,
Não gorjeiam como lá.
class 'str'
```

Note que para usar aspas duplas dentro de uma string (str) precisamos escrever a string entre aspas simples para não terminar precocemente a string, o que resultaria em um erro. O contrário também serve para aspas simples como no apóstrofe da frase em inglês: `"That's life."`.

float

Abreviação de floating, que se refere a `floating point number` ou `número de ponto flutuante`. São os números racionais da matemática. O nome é uma tecnicalidade da computação referente ao padrão de representação de números racionais no sistema binário dos computadores que não pode ser feito de maneira simples como os números inteiros.

Instanciação

Sempre que colocamos números com a parte decimal explicita. Diferente do português que usa vírgula, o inglês representa números decimais com pontos:

```
numeroRacional1 = 1.0
numeroRacional2 = 56.133000
print(numeroRacional1)
print(type(numeroRacional1))
print(numeroRacional2)
print(type(numeroRacional2))
```

```
1.0
class 'float'
56.133
class 'float'
```

Note que o python por padrão corta os zeros à direita por padrão, ele só mantém o primeiro decimal obrigatóriamente para mostrar que se trata de um float.

bool

Esse aqui é um que é um pouco mais difícil de entender embora seja o mais fácil de se usar. bool é abreviação de `boolean` e não tem tradução para o português. É um tipo característico da programação e é fundamental devido ao caráter binário das representações em um computador. Sempre será False ou True, assim como no computador tudo é 0 ou 1.

Instanciação

Para instanciar, podemos escrever simplesmente True ou False, sem aspas. Outra opção é fazer um teste com operadores lógicos.

```
chovendo = True
calor = False
vascoEMaisCampeaoQueOFlamengo = 4 > 7
print(chovendo)
print(type(chovendo))
print(calor)
print(type(calor))
print(vascoEMaisCampeaoQueOFlamengo)
print(type(vascoEMaisCampeaoQueOFlamengo))
```

```
True
type 'bool'
False
type 'bool'
False
type 'bool'
```

Existem diversos operadores lógicos que analisam sentenças lógicas e retornam um booleano (uma tradução informal para o português). Falaremos deles assim que terminarmos os principais tipos de dados.

list

Antes de aprender o que é uma `list`, ou em português lista, precisamos aprender o que é um vetor na computação. Vetores são usados em diversas áreas do conhecimento: matemática, física e biologia. Na escola, nas aulas de matemática e física, aprendemos que vetor tem direção, sentido e módulo. Isso não é bem uma explicação do que é um vetor e sim mais como ele se manifesta. Entender o que é um vetor é um grande desafio para todo aluno do ensino médio. Existe uma definição algébrica complicada do que é um vetor usada na matemática e na física. Nas aulas de biologia existe uma definição mais geral e fácil de entender, que podemos inclusive levar para matemática, para física e para a computação (embora não perfeitamente para esses 3). De maneira geral, vetores são aquilo que carregam alguma coisa. O mosquito aedes aegypti é o vetor da dengue. F é o vetor da física que carrega a força. Na computação, vetores carregam informações e o principal tipo de vetor no python são as listas.

Instanciação

Para instanciar uma lista no python usamos colchetes. Separamos diferentes elementos com uso de vírgulas, tudo entre os colchetes:

```
vetorDeNumeros = [1, 2, 3, 4]
vetorDeFrutas = ["Morango", "Pera", "Banana"]
vetorVazio = [ ]
print(vetorDeNumeros)
print(type(vetorDeNumeros))
print(vetorDeFrutas)
print(type(vetorDeFrutas))
print(vetorVazio)
print(type(vetorVazio))
```

```
[1, 2, 3, 4]
class 'list'
["Morango", "Pera", "Banana"]
class 'list'
[]
class 'list'
```

Cada linguagem tem sua maneira de representar um vetor. Por padrão, vetores têm tamanho fixo e um único tipo. No python vetores são flexíveis, tanto quanto a tamanho quanto a tipo, uma das razões do porque o python é uma linguagem de programação tão usada nos dias de hoje em ciência de dados e outras áreas da computação.

Idealmente uma lista recebe apenas um tipo de dado: lista de números inteiros, lista de strings, etc. É mais uma convenção usada por programadores para facilitar o entendimento de um código naqueles projetos maiores que citei algumas vezes.

Existem outros tipos de vetores além das listas no próprio python: `tuple`, `set`, `dict`. Eles são usados em certas situações mais específicas. Para se aprofundar no python são indispensáveis. Quando chegarmos no uso de ferramentas mais diretas para ciência de dados falaremos um pouco mais.

Uso

Cada elemento de uma lista é acessível usando o nome da lista e colchetes com o índice do elemento na lista começando por 0 (índice zero aponta para o primeiro elemento da lista, uma herança do sistema binário dos computadores no python).

```
vetorDeFrutas = ["Morango", "Pera", "Banana"]
print(vetorDeFrutas[0])
print(vetorDeFrutas[1])
```

```
print(vetorDeFrutas[2])
```

```
Morango
Pera
Banana
```

Se tentarmos acessar um índice que não existe na lista, digamos, queremos acessar o quarto elemento do vetorDeFrutas acarretará em um erro:

```
vetorDeFrutas = ["Morango", "Pera", "Banana"]
print(vetorDeFrutas[3])
```

```
Traceback (most recent call last):
File "stdin", line 1, in module
IndexError: list index out of range
```

Esse erro é bem frequente quando trabalhamos com listas.

Entendi o que são vetores e sua principal representação no python, as listas e até como usá-las. Mas porque preciso usar? Digamos que você possui um conjunto de matrículas de alunos e um conjunto dos nomes dos alunos. Como você pode representar?

`Opção 1:`

nomeAluno1 = “Anderson” matriculaAluno1 = 123 nomeAluno2 = “Antonio” matriculaAluno2 = 456 nomeAluno3 = “Alexandre” matriculaAluno3 = 789

`Opção 2:`

listaDeNomes = [“Anderson”, “Antonio”, “Alexandre”] listaDeMatriculas = [123, 456, 789]

Não ficou mais fácil na segunda opção? Pode não parecer tão fácil assim agora, mas em breve você perceberá isso. Talvez só o uso exaustivo vai mostrar porque a segunda forma é melhor com todos é assim. É mais uma daquelas coisas na programação que quando aumenta a escala fica difícil de manter. No geral, se for usar apenas um nome no código inteiro, use uma variável simples. Se for usar mais de uma, use listas. Isso vale sempre que os dados forem informações relacionadas, nome com nome, fruta com fruta etc.

Matrizes

Existem ainda listas de lista, as chamadas matrizes. Matrizes na álgebra podem ser entendidas como conjunto de vetores, na computação é o mesmo entendimento. Elas são particularmente interessantes quando estamos usando um conjunto grande de dados. Imagina que além da matrícula e do nome temos a idade, o curso, o período que entrou na faculdade etc. Podemos ter apenas uma lista que possui todas essas listas.

```
cabecalhoInformacoesAlunos = ["matricula", "nome", "idade", "curso", "semestreIngresso"]
matrizInformacoesAlunos = [[123, 456, 789], ["Anderson", "Antonio", "Alexandre"], [20, 18,
print(matrizInformacoesAluno)
```

```
[[123, 456, 789], ["Anderson", "Antonio", "Alexandre"], [20, 18, 33], ["estatística", "dire
```

É importante lembrarmos a qual se refere cada lista da matriz, por isso a variável cabecalhoInformacoesAlunos. Em ferramentas específicas para ciência de dados no python isso é feito diretamente, mas no fundo é uma abstração do código a cima. Não preciso nem dizer que isso faz sentido se tivermos um projeto grande, né? A mesma lógica de sempre na programação se aplica aqui também. Assim como sobre quando usar uma variável simples ou uma lista, matrizes são interessantes quando for usar mais de uma lista de dados relacionados, nome de alunos, matrícula de alunos etc. Idealmente uma matriz possui listas de mesmo tamanho, mas isso é uma convenção, não uma regra.

Para acessar as informações de uma matriz é preciso dois índices entre colchetes.

```
cabecalhoInformacoesAlunos = ["matricula", "nome", "idade", "curso", "semestreIngresso"]
matrizInformacoesAlunos = [[123, 456, 789], ["Anderson", "Antonio", "Alexandre"], [20, 18,
print("Matrícula e idade do segundo aluno:")
print(matrizInformacoesAlunos[0][1])
print(matrizInformacoesAlunos[2][1])
```

```
Matrícula e idade do segundo aluno:
456
18
```

O primeiro índice é referente a posição da lista na matriz e o segundo índice é a posição do elemento específico na lista desejada. É equivalente ao conceito de colunas e linhas de uma matriz(ij) da matemática. Lembrando que o primeiro índice de uma lista é 0 e não 1.

Pode-se também fazer uma lista de matrizes, que se acessa usando 3 colchetes. Matrizes tem dimensões infinitas no python, mas para ciência de dados comumente usamos apenas 2 dimensões: colunas e linhas. Para computação gráfica, por exemplo, podemos usar mais dimensões e matrizes passam a ser chamadas de tensores. Isso não é importante para ciência de dados, pelo menos não agora (uma pequena antecipação de cenas dos próximos capítulos).

function

As functions, que do inglês significa `funções`, possuem um conceito semelhante ao da matemática, elas transformam uma informação.

O print() é um exemplo de função. Ele recebe como argumento um texto e imprime na tela. O type() recebe uma variável qualquer e retorna o seu tipo.

Existe uma nomenclatura própria para trabalhar com funções, o que a função recebe para trabalhar é o que chamamos de argumentos e o que ela cria a partir desse argumento é o retorno ou resultado. Uma função pode ter mais de um argumento, que são separados por vírgulas. Algumas funções não possuem resultado, o print() por mais que imprima na tela o texto, ele não gera retorno específico, ele é uma função bem especial, por isso falamos dela antes de qualquer outra.

```
nomeDaFunção(argumento1, argumento2, …, argumentoN)
```

Lembra que para imprimir duas informações diferentes em uma mesma linha precisávamos escrever um , end=“ ” depois do texto no print()? O print tem inúmeros argumentos. O end é um dos argumentos, ele por padrão é um `"\n"` que faz com que o computador pule uma linha. Argumentos que possuem padrão, se quisermos alterar, precisamos especificá-los, já que não é obrigatório para o funcionamento da função. Quando falarmos mais sobre como criar uma função falaremos mais sobre isso, não precisa entender isso agora.

Chamar uma função X Referenciar uma função

Venho falando de funções sempre com os parêntesis para diferenciar de outras variáveis normais. Na verdade, os parênteses servem para chamar a função para fazer alguma coisa. Nós podemos referenciar funções apenas pelo seu nome. Por exemplo:

```
print(type(print))
```

```
class 'builtin_function_or_method'
```

Esse código apenas exibe o tipo da variável print, que é uma função. Essa forma de representar funções é útil em códigos mais complexos que usam funções customizadas dentro de outras funções.

Na referenciação, funções funcionam como qualquer outra variável.

Instanciação

def soma(x, y): return x + y

Como pode ver é diferente dos demais tipos, usa uma palavra reservada para definir, o def. Para funções, invés de dizer que estamos instanciando, dizemos que estamos definindo-a. Em breve falaremos mais sobre como definir uma função, por enquanto é importante apenas entender que existem algumas funções padrões no python e que podemos criar mais se desejarmos.

Objetos

Os objetos são tipos especiais no python, especiais porque? Nós que o criamos!

Mas nós podemos criar uma função, ou uma String também não? O que tem de especial nisso?

Nós não criamos de fato uma String, nós instanciamos ela, ou seja, criamos exemplos delas. E funções nós definimos, o tipo continua sendo funções.

No dicionário li que essas palavras podem ser sinônimos, não estamos falando da mesma coisa?

Não. São palavras com significado parecido, mas na programação são diferentes.

Entendi, eu acho...

Enfim,

Nós definimos uma classe que o objeto pertence e instanciamos a partir dessa classe, essencialmente, essa classe que criamos é um tipo novo, definimos seu comportamento características etc. No próximo caderno falaremos mais sobre elas. Por enquanto vale apenas saber que eles existem.

Operandos

Os operandos são sinalizações para o python para realizar alguma operação com dois elementos que referenciamos. Muitos são idênticos ou semelhantes a matemática.

Os elementos referenciados podem ser variáveis ou valores. Podem ser usados dentro de funções como atribuição de variáveis.

Diferentes tipos possuem diferentes comportamentos em relação aos operadores.

+ e -

Os primeiros operadores que aprendemos na escola. O operador + é o operador da adição e - da subtração.

int e float

Em relação a ints e floats funcionam igual à matemática:

```
x = 1 + 3
print(x - 2)
```

```
2
```

Se somar um inteiro com um float a resposta será sempre um float, nem que seja com .0

```
x = 1 + 3.0
print(x - 2)
```

```
2.0
```

str

Para strings também é possível usar o operador +. Porém diferente de números, como citado anteriormente quando falamos de strings, o + concatena duas strings. Não podemos somar strings a outros tipos.

```
texto = "ketchup"
print(texto + " e mostarda")
```

```
ketchup e mostarda
```

bool

Até dá para usar, mas não serve de muita coisa. O python transforma o valor de True para o inteiro 1 e False para o inteiro 0 e soma como se fosse inteiros

```
print(True + True + False)
```

```
2
```

* e /

Multiplicação e divisão respectivamente.

int e float

Funcionam igual na matemática também, só o símbolo do operador que é diferente.

```
x = 4 * 8
print(x / 3)
```

```
10.666666666666666
```

Quando dividimos dois números inteiros que não possuem valor exato, a resposta é automaticamente convertida para float.

Repare que a divisão retornou o valor sem aproximação. Por mais que se trate de uma dízima periódica, por conta do funcionamento dos floats nos computadores, o resultado pode vir um pouco diferente do esperado, como no caso, que deveria ter vindo com um 7 no final.

str

Podemos usar multiplicação em strings. É como se fosse várias operações de adição em sequência, e funcionam igual a adição de strings normal. Não há divisão de strings no python.

```
textoOriginal = "Ciência de dados!\n"
print(textoOriginal*5)
```

```
Ciência de dados!
Ciência de dados!
Ciência de dados!
Ciência de dados!
Ciência de dados!
```

// e %

// é o operador para divisão inteira e % é o operador de resto da divisão. Digamos que você não queira a parte decimal de uma divisão por alguma razão e quer o resto da divisão também. Funciona apenas para ints e floats.

25 amigos foram jogar futebol na praça, querem montar equipes de 11 e duas pessoas ficam "na de fora" e entram quando o jogo acabar substituindo alguém do time perdedor. Quantos ficaram na de fora e quantas equipes serão formadas?

```
nAmigos = 25
nEquipes = 25//11
amigosNaDeFora = 25%11
print("Amigos na de fora:")
print(3)
print("Equipes formadas:")
print(2)
```

```
Amigos na de fora:
3
Equipes formadas:
```

```
2
```

**

Exponenciação. Funciona igual a matemática, apenas o símbolo que é novo. Apenas para inteiros e floats. Assim como na matemática, exponenciação pode ser radiciação se usando números menores que 1 como expoentes.

```
x = 9
xAoQuadrado = 9 ** 2
raizQuadradaDeX = 9 ** 0.5
print(xAoQuadrado)
print(raizQuadradaDeX)
```

```
81
3.0
```

(e)

Colocando assim parece que são dois operadores diferentes, mas na verdade funcionam sempre juntos. São os parêntesis. Funcionam independente do tipo. Colocar uma expressão entre parêntesis significa que ela será executada antes. Seguindo as regras da matemática, exponenciação e radiciação serão sempre feitas antes, multiplicação e divisão em seguida e por último adição e subtração. Usando parêntesis, semelhante a como funciona na matemática, muda essa ordem de prioridade. Os [ ] (colchetes) e { } (chaves) não funcionam que nem na matemática, servem para outras coisas, mas com os parêntesis já podemos fazer qualquer operação na ordem que quisermos.

```
expressaoAlgebrica = (((2+3)/2)+4**2)**(1/2)
print("Resultado da expressão algébrica (((2+3)/2)+4**2)**(1/2):")
print(expressaoAlgebrica)
```

```
Resultado da expressão algébrica (((2+3)/2)+4**2)**(1/2):
4.301162633521313
```

funções

Para funções funcionam um pouco diferente. Parêntesis executam uma função.

```
    print("Olá Mundo!")
```

```
Olá Mundo!
```

Operadores condicionais

Os operadores condicionais são operadores que avaliam uma expressão e retornam um booleano. São eles:

- > (maior que)
- < (menor que)
- >= (maior ou igual a)
- <= (menor ou igual a)
- == (igual)
- != (diferente)
- not
- and (ou &)
- or (ou |)
- in

>, <, >=, <=

Funcionam quando comparando números (inteiros ou floats) ou quando comparando strings. No caso dos números funcionam igual a matemática. Para strings considera o número de caracteres de cada uma e faz a comparação de tamanho.

```
    print(4 > 7)
    print(5 <= 10.4)
    print("Mostarda" >= "Maionese")
```

```
False
True
True
```

== e !=

Podem ser usados entre diferentes tipos para avaliar se os elementos são iguais, note que "4" é diferente de 4 por conta do tipo.

```
print(4 == 7)
print(5 == 5)
print(4 != "4")
print(True != True)
```

```
False
True
True
False
```

not, and, or

São usados entre booleanos apenas. Servem para avaliar expressões lógicas mais complexas.

O `not`, "não" traduzido do inglês, inverte o booleano de uma expressão.

O `and`, "e" traduzido do inglês, precisa ser usado entre duas expressões e retorna True se ambas expressões são verdadeiras e False caso não.

O `or`, "e" traduzido do inglês, precisa ser usado entre duas expressões e retorna True se ambas expressões forem verdadeiras ou se uma delas for verdadeira. Se ambas expressões forem falsas retornará False.

Expressões lógicas precisam estar entre parêntesis quando avaliadas.

```
expressao1 = (1 < 4) and (1 == 0.0)
expressao2 = (expressao1) or ("Mostarda" == "Mostarda")
expressao3 = (not expressao1) and ("Mostarda" == "Mostarda")
print(expressao1)
print(expressao2)
print(expressao3)
```

```
False
True
True
```

in

Exclusivo para vetores e strings, muito útil para saber se um elemento está dentro de um vetor ou não, já para strings, se uma string menor está presente em uma string maior.

```
    listaDeCodimentos = ["Ketchup", "Mostarda", "Maionese"]
    print("Pimenta" in listaDeCodimentos)
    print("Mostarda" in listaDeCodimentos)
    print("ostar" in listaDeCodimentos[1])
```

```
False
True
True
```

Métodos e funções padrão

Existem diversas funções padrões no python. Já vimos o `print()` e o `type()`.

Funções podem agilizar o trabalho de um programador, fazendo automaticamente operações complexas que às vezes exigiriam centenas de linhas de código. São uma base importante para ciência de dados ou qualquer outra área da informática. Já vimos que elas nada mais são do que um tipo dentro do python e que podem ser chamadas para realizar uma tarefa quando colocamos () depois de referenciá-las.

Métodos são um tipo de função também. São funções exclusivas de certos objetos, aquele tipo especial que falaremos no próximo caderno (criando expectativa com eles?). A diferença é que precisamos chamá-los depois de um elemento usando . como em `"Mostarda, Ketchup, Maionese".split(", ")` invés de `split("Mostarda, Ketchup, Maionese", ", ")`, algumas funções seriam mais fáceis de ser entendidas quando usadas no formato de método como no caso do split (falaremos dele em breve).

Transformação de tipo (casting)

O cast, ou transformação de tipo (uma tradução aproximada do inglês), é uma função que existe de uma forma ou outra em qualquer linguagem de programação. Serve para transformar um tipo em outro, simples assim.

Lembra que quando comparamos 4 com "4" usando o operador == recebemos False? Retornará verdadeiro se o tipo fosse o mesmo:

```
print(int("4") == 4)
```

```
True
```

Para fazer uma transformação de tipo usamos funções com o nome do tipo que queremos receber e dentro do parêntesis o elemento que queremos converter. Como no exemplo, transformamos uma string em um inteiro usando `int()`. Com booleanos, se uma string não estiver vazia ou se um número inteiro ou float não for 0 (ou 0.0) `bool()` retornará True e False caso contrário.

```
print("4" == str(4))
numeroInteiro = 4
numeroRacional = 3.0
print(int(numeroRacional))
print(float(numeroInteiro))
print(bool(numeroInteiro))
```

```
True
3
4.0
True
```

O mesmo serve para converter em strings, floats e booleanos. Entre diferentes tipos de vetores, podemos fazer casts também. Entre listas e tuplas por exemplo, basta usar `list()` ou `tuple()`

```
listaDeCodimentos = ["Mostarda", "Ketchup", "Maionese"]
print(tuple(listaDeCodimentos))
```

```
('Mostarda', 'Ketchup', 'Maionese')
```

Não existe type casting (tipo em inglês + gerúndio de cast, usado as vezes para se referir a transformação de tipos) de funções.

Principais funções de inteiros (int)

range()

O `range` cria um uma sequência numérica com os parâmetros passados.

Se usarmos apenas um número inteiro na função, a sequência iniciará com 0, terá passo 1 e será até o número anterior ao escrito. Com dois inteiros entre os parêntesis, o primeiro será o início e o segundo o número seguinte ao último da sequência. O terceiro argumento inteiro, que também é opcional, modifica o passo. O resultado da função é um tipo único, o tipo range, que pode ser convertido em lista.

```
de0a4 = range(5)
numerosDeUmADez = range(1, 11)
numerosParesDeZeroAVinte = range(0, 21, 2)
print(list(de0a4))
```

```
print(list(numerosDeUmADez))
print(list(numerosParesDeZeroAVinte))
```

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

random.randint()

Retorna um número inteiro aleatório dentro de um intervalo, especificado por dois inteiros inseridos.

Utiliza uma extensão do python chamada random. Para poder usar essa função precisamos escrever import random no início do arquivo python.

```
import random
numeroAleatorio1 = random.randint(3, 9)
numeroAleatorio2 = random.randint(3, 9)
numeroAleatorio3 = random.randint(3, 9)
numeroAleatorio4 = random.randint(3, 9)
print(numeroAleatorio1)
print(numeroAleatorio2)
print(numeroAleatorio3)
print(numeroAleatorio4)
```

```
3
4
7
7
```

Principais funções de floats (float)

round()

Arredonda um número. .5 é arredondado para baixo. O resultado é um número inteiro.

```
print(round(3.3))
print(round(2.5))
```

```
print(round(4.7))
```

```
3
2
5
```

math.ceil()

Arredonda um número racional para cima, ceil é uma palavra do inglês que significa teto.

Utiliza uma extensão do python chamada math. Para poder usar essa função precisamos escrever import math no início do arquivo python.

```
import math
print(math.ceil(3.3))
print(math.ceil(2.5))
print(math.ceil(4.7))
```

```
4
3
5
```

math.floor()

Arredonda um número racional para baixo, floor é uma palavra do inglês que significa chão

Utiliza uma extensão do python chamada math. Para poder usar essa função precisamos escrever import math no início do arquivo python.

```
import math
print(math.floor(3.3))
print(math.floor(2.5))
print(math.floor(4.7))
```

```
3
2
4
```

math.trunc()

Sigla para truncate, truncar em português. Remove a parte decimal de um número. Equivalente ao math.floor() para números positivos e equivalente ao math.ceil() para números negativos.

Utiliza uma extensão do python chamada math. Para poder usar essa função precisamos escrever import math no início do arquivo python.

```
import math
print(math.trunc(3.3))
print(math.trunc(2.5))
print(math.trunc(-4.7))
```

```
3
2
-4
```

Principais métodos e funções de strings (str)

input()

O `input` recebe uma string que será impressa na tela. Não muito diferente de um print né? A diferença é que o input após imprimir essa string recebe o que digitarmos até apertamos enter.

Ela sempre receberá uma string, então se quisermos inserir números, precisamos fazer uma transformação de tipo depois.

```
nome = input("Digite seu nome: \n")
idade = int(input("Digite sua idade: \n"))
print("Você se chama " + nome + " e fará " + str((idade + 1)) + " anos")
```

```
Digite seu nome:
Gabriel
Digite sua idade:
23
Você se chama Gabriel e fará 24 anos
```

len()

Sigla para length, que significa comprimento. Recebe uma string ou vetor e retorna quantos caracteres ou elementos possui em int.

```
frase = "Meu filme favorito é troll 2!"
print(len(frase))
```

```
29
```

.strip()

Remove espaços no início e no final de uma frase.

O `.strip()` é um método, então precisamos colocar a string antes do ponto e, para esse método, não colocamos nada dentro dos parêntesis como parâmetro.

```
frase = "     Meu filme    favorito é troll 2!                    "
print(frase.strip())
```

```
Meu filme    favorito é troll 2!
```

.replace()

Substituir do inglês. Substitui todas as ocorrências de uma string dentro de uma outra string por outra. Muito usada quando queremos remover elementos textuais.

O `.replace()` é um método, então precisamos colocar a string antes do ponto e, para esse método, colocamos como parâmetro a string que queremos substituir e em seguida a string nova.

```
frase = "Meu filme favorito é troll 2 e o meu segundo filme favorito é troll 1!"
print(frase.replace("troll", "sharknado"))
```

```
Meu filme favorito é sharknado 2 e o meu segundo filme favorito é sharknado 1!
```

.split()

Do inglês separar. Separa uma string em várias dado um separador. O resultado é uma lista de strings.

O `.split()` é um método, então precisamos colocar a string antes do ponto e, para esse método, colocamos como parâmetro uma string que será usada como separadora. Se não colocarmos nada dentro do split ele considerá por padrão o separador como sendo um espaço.

```
frase = "Ketchup Maionese Mostarda"
listaDeCodimentosGastronomicos = frase.split(" ")
print(listaDeCodimentosGastronomicos)
```

```
['Ketchup', 'Maionese', 'Mostarda']
```

.format()

Do inglês formatar. Substitui da string alvo as `{}` por outras strings.

O `.format()` é um método, então precisamos colocar a string antes do ponto e, para esse método, colocamos como parâmetro n strings para n {} na string alvo.

```
listaDeCodimentosGastronomicos = ['Ketchup', 'Maionese', 'Mostarda']
frase = "Codimento Gastronomico 1: {}\nCodimento Gastronomico 2: {}\nCodimento Gastronomic
print(frase)
```

```
Codimento Gastronomico 1: Ketchup
Codimento Gastronomico 2: Maionese
Codimento Gastronomico 3: Mostarda
```

.isalnum(), .isalpha() e .isnumeric()

`.isalnum()` retorna True se a string alvo for composta exclusivamente de números e letras. `.isalpha()` retorna True se a string alvo for composta exclusivamente de letras. `.isnumeric()` retorna True se a string alvo for composta exclusivamente de números.

Os três são métodos, então precisamos colocar a string antes do ponto e, para esses métodos, não colocamos nada dentro dos parêntesis como parâmetro.

```
print("Troll2".isalnum())
print("Ketchup".isalpha())
print("Ciência de dados!".isalpha())
print("2024".isnumeric())
print("Sharknado 5 e troll 2!".isalnum())
print("ano de 2024".isnumeric())
```

```
True
True
False
True
False
False
```

.upper(), .lower() e .capitalize()

`.upper()` significa superior, transforma todos os caracteres de uma string em caracteres maiúsculos. `.lower()` significa inferior, transforma todos os caracteres de uma string em caracteres minúsculos. `.capitalize()` significa capitalizar, transforma o primeiro caracter de uma string em um caractere maiúsculo e os demais em minúsculos.

Os três são métodos, então precisamos colocar a string antes do ponto e, para esses métodos, não colocamos nada dentro dos parêntesis como parâmetro.

```
print("Hello World!".lower())
print("ketchup, maionese, mostarda".upper())
print("ketchup, maionese, mostarda".capitalize())
```

```
hello world!
KETCHUP, MAIONESE, MOSTARDA
Ketchup, maionese, mostarda
```

.join()

Significa juntar. Junta uma lista de strings usando uma string comum no meio e retorna como uma string apenas.

O `.join()` é um método, então precisamos colocar a string antes do ponto e, para esse método, a string antes do ponto será a string comum que unirá e dentro do parêntesis a lista de strings para unir.

```
listaDeCodimentos = ["Ketchup", "Maionese", "Mostarda"]
print(", ".join(listaDeCodimentos))
```

```
Ketchup, Maionese, Mostarda
```

.find()

Do inglês encontrar. Retorna o índice em que uma string se encontra dentro da outra. Apenas para a primeira ocorrência. Se não encontrar a string, retornará -1.

O `.find()` é um método, então precisamos colocar a string antes do ponto e, para esse método, colocamos dentro do parêntesis a string que queremos encontrar dentro da string que referenciamos antes do ponto.

```
print("Melhores filmes: Troll 2 e Troll 1".find("Troll"))
print("Melhores filmes: Troll 2 e Troll 1".find("Sharknado"))
```

```
17
-1
```

.count()

Do inglês contar. Conta ocorrências de uma string dentro de outra.

O `.count()` é um método, então precisamos colocar a string antes do ponto e, para esse método, colocamos dentro do parêntesis a string que queremos contar dentro da string que referenciamos antes do ponto.

```
print("Sharknado 1, Sharknado 2, Sharknado 3, Sharknado 4, Sharknado 5, Sharknado 6".count
```

```
6
```

Principais métodos e funções de lista (list)

max()

Do inglês máximo. Retorna o maior valor da lista inserida entre os parêntesis. Funciona tanto para lista de inteiros quanto lista de racionais

```
print(max([1, 2, 7, 3, 4]))
```

```
7
```

min()

Do inglês mínimo. Retorna o menor valor da lista inserida entre os parêntesis. Funciona para listas com qualquer valor numérico, isto é, tanto floats quanto ints.

```
print(min([1, 2, 7, 3, 4]))
```

```
1
```

sum()

Do inglês somar. Retorna o valor da soma dos elementos da lista. Funciona para listas com qualquer valor numérico, isto é, tanto floats quanto ints.

```
print(sum([1, 2, 7, 3, 4]))
```

```
17
```

len()

Semelhante ao len() das strings. Retorna quantos elementos uma lista inserida entre os parêntesis possui.

```
print(len([1, 2, 7, 3, 4]))
```

```
5
```

.append()

Do inglês acrescentar. Acrescenta um elemento no final de uma lista. Não tem retorno, apenas modifica uma lista já existente.

O `.append()` é um método, então precisamos colocar a lista antes do ponto e, para esse método, colocamos dentro do parêntesis o elemento que queremos adicionar dentro da lista que referenciamos antes do ponto.

```
codimentosGastronomicos = ["Ketchup", "Maionese"]
codimentosGastronomicos.append("Mostarda")
print(codimentosGastronomicos)
```

```
["Ketchup", "Maionese", "Mostarda"]
```

.extend()

Do inglês extender. Adiciona os elementos de uma lista a outra. Não tem retorno, apenas modifica uma lista já existente.

O `.extend()` é um método, então precisamos colocar a lista antes do ponto e, para esse método, colocamos dentro do parêntesis a lista que possui os elementos que queremos adicionar na que referenciamos antes do ponto.

```
codimentosGastronomicos = ["Ketchup", "Maionese"]
codimentosGastronomicos.extend(["Mostarda"])
print(codimentosGastronomicos)
```

```
["Ketchup", "Maionese", "Mostarda"]
```

.insert()

Do inglês inserir. Insere um elemento dentro de uma lista no índice fornecido. Se o índice for maior que o número de elementos da lista, será adicionado no final da lista. Não tem retorno, apenas modifica uma lista já existente.

O `.insert()` é um método, então precisamos colocar a lista antes do ponto e, para esse método, colocamos dentro do parêntesis o índice seguido do elemento que queremos adicionar.

```
codimentosGastronomicos = ["Ketchup"]
codimentosGastronomicos.insert(231, "Mostarda")
codimentosGastronomicos.insert(1, "Maionese")
print(codimentosGastronomicos)
```

```
["Ketchup", "Maionese", "Mostarda"]
```

.remove()

Do inglês remover. Remove a primeira ocorrência de um elemento de uma lista. Não tem retorno, apenas modifica uma lista já existente.

O `.remove()` é um método, então precisamos colocar a lista antes do ponto e, para esse método, colocamos dentro do parêntesis o elemento que queremos remover.

```
codimentosGastronomicos = ["Ketchup", "Maionese", "Mostarda"]
codimentosGastronomicos.remove("Ketchup")
print(codimentosGastronomicos)
```

```
["Maionese", "Mostarda"]
```

.pop()

Do inglês disparar. Retorna o elemento da lista apontado pelo índice fornecido e o remove da lista.

O `.pop()` é um método, então precisamos colocar a lista antes do ponto e, para esse método, colocamos dentro do parêntesis o índice elemento que queremos remover. Se não colocarmos nenhum índice, o último elemento será removido.

```
codimentosGastronomicos = ["Ketchup", "Maionese", "Mostarda"]
codimentosGastronomicos.pop(1)
print(codimentosGastronomicos)
print(codimentosGastronomicos.pop())
print(codimentosGastronomicos)
```

```
['Ketchup', 'Mostarda']
Mostarda
['Ketchup']
```

.index()

Do inglês índice. Retorna o índice da primeira ocorrência do elemento entre os parêntesis.

O `.index()` é um método, então precisamos colocar a lista antes do ponto e, para esse método, colocamos dentro do parêntesis o elemento que queremos o índice. Se o elemento não estiver na lista resultará em um erro.

```
codimentosGastronomicos = ["Ketchup", "Maionese", "Mostarda"]
print(codimentosGastronomicos.index("Ketchup"))
```

```
0
```

.count()

Semelhante ao .count das strings. Retorna o número de ocorrências do elemento em uma lista.

O `.count()` é um método, então precisamos colocar a lista antes do ponto e, para esse método, colocamos dentro do parêntesis o elemento que queremos contar.

```
lista = ["Ciência de dados!", "Ciência de dados!", "Ciência de dados!", "Ciência de dados!
print(lista.count("Ciência de dados!"))
```

```
4
```

.sort() ou sorted()

Do inglês organizar e organizado. Rearruma uma lista de números organizando ela por ordem crescente ou decrescente. Funciona para listas que contém apenas números inteiros e racionais e para listas que contém apenas strings (organiza por ordem alfabética nesse caso).

Por padrão será de ordem crescente, mas se colocarmos `reverse=True` como parâmetro, será em ordem decrescente.

.sort() é um método que altera uma lista já existente, sorted() é uma função que retorna uma lista nova organizada.

```
numeros1 = [5, 1, 77, 8, 2, 3, 3213, 23, -3, -7]
numeros1.sort(reverse=True)
numeros2 = [5, 1, 77, 8, 2, 3, 3213, 23, -3, -7]
numeros2 = sorted(numeros2)
listaDeCodimentos = ["Ketchup", "Mostarda", "Maionese"]
print(numeros1)
```

```
print(numeros2)
print(sorted(listaDeCodimentos))
```

```
[3213, 77, 23, 8, 5, 3, 2, 1, -3, -7]
[-7, -3, 1, 2, 3, 5, 8, 23, 77, 3213]
['Ketchup', 'Maionese', 'Mostarda']
```

.reverse()

Do inglês inverso. Inverte a ordem de uma lista. Não gera retorno, apenas altera uma lista já existente.

É um método.

```
listaDeCodimentos = ["Ketchup", "Maionese", "Mostarda"]
listaDeCodimentos.reverse()
print(listaDeCodimentos)
```

```
['Mostarda', 'Maionese', 'Ketchup']
```

Comentários

As linguagens de programação possuem um recurso chamado comentário. Eles servem para escrever algum comentário no código que não será lido pelo interpretador. Muito útil para explicar o funcionamento de alguma parte do código, ou para deixar salvo algum recurso que não foi utilizado mas que pode ser utilizado no futuro.

É essencial para um programador utilizar comentários em seus códigos, tanto para se entender depois quanto para trabalhar em equipe. Fica muito difícil entender um código e não está comentado. A forma de se expressar em um comentário é de escolha do programador, de preferência se expressar da maneira mais direta possível.

Não é preciso comentar cada linha. Se comenta para explicar o que está escrito no código, muitos comentários podem até atrapalhar a legibilidade do código. Com o tempo, a experiência dirá onde comentar e onde não comentar.

No python existem duas formas de comentar no códgio: com `#` e com `""" """`.

Comentários com hashtag

Principal forma de comentar um código no python. Deixa tudo que está depois do caractere # comentado. Pode ser usado no início de uma linha ou no meio, no segundo caso, apenas o que está depois do # será comentado.

Ideal para explicar o funcionamento de uma parte do código e para deixar salvo algo que não foi utilizado mas que pode ser utilizado no futuro.

```
#Lista de codimentos mais utilizados na cozinha brasileira
listaDeCodimentos = ["Maionese", "Mostarda", "Ketchup"]
#Organiza em ordem alfabética
listaDeCodimentos.sort()
#Inverte a ordem dos codimentos (não vou utilizar agora)
#listaDeCodimentos.reverse()
print(listaDeCodimentos)
```

```
['Ketchup', 'Maionese', 'Mostarda']
```

Comentários com “ “ ” “ “ ”

Menos usado. Na verdade, não é comentário de verdade, o interpretador lê as frases entre as 3 aspas como se fosse uma string normal, porém não faz nada. Pode ser usado para comentar de maneira rápida muitas linhas de código.

Outro uso é para definir funções e classes. As IDEs usam essa string para ajudar na reutilização da função ou classe. Usar `""" """` no início da definição de uma função ou classe permite que esse comentário seja visto pela dica flutuante ao escrever uma função em um código dentro de uma IDE.

Aprofundamento em programação com Python

Conteúdos

- Controle de Fluxo
- Estruturas de repetição
- Funções
- Orientação a Objetos
- Instalação de bibliotecas
- Importação de bibliotecas

Controle de fluxo

Nem sempre nosso código deve se comportar de maneira igual para todas as situações, às vezes precisamos tomar uma decisão em função de nossas operações em situações específicas e para isso podemos utilizar estruturas de condição.

if

Quando falamos em tomar decisões, a estrutura que trabalhamos é sempre o `if`, que é uma estrutura que testa condições impostas para decidir como prosseguir em certas situações, como por exemplo checar se um número é maior, menor ou igual a outro. Testando uma condição única:

```
if 10 < 30:
    print('10 é menor que 30')
```

```
10 é menor que 30
```

Podemos também testar múltiplas condições exigindo que todas sejam verdadeiras ou apenas uma seja, dependendo do operador usado: or ou and. Precisamos separar essas diferentes condições usando parentêsis.

O operador `or`, "ou" traduzindo do inglês, faz com que a condição seja satisfeita se a primeira **ou** a segunda expressão (ou as duas) forem verdadeiras. Pode ser também representado pelo caractere `|`

```
if (7 < 3) or (5 > 1):
    print('Pelo menos um é verdade')
```

```
Pelo menos um é verdade
```

Já o operador `and`, "e" traduzindo do inglês, faz com que a condição seja satisfeita se a primeira **e** a segunda expressão sejam verdadeiras Pode ser também representado pelo caractere `&`

```
if (7 == 7) and (5 > 1):
    print('Ambas expressões são verdadeiras')
```

```
Ambas expressões são verdadeiras
```

else

Para toda situação sempre podemos incluir uma saída padrão como resposta se a condição não for satisfeita, para isso temos o `else`.

```
n = 40
if n < 30:
    print('%i é menor que 30'%(n))
else:
    print('%i é maior ou igual a 30'%(n))
```

```
40 é menor maior ou igual a 30
```

elif

Podemos também incluir mais de uma condição a ser testada, adicionando mais instâncias do if ao mesmo if com a estrutura `elif`.

```
n = 40
if n < 30:
    print('%i é menor que 30'%(n))
elif n > 30:
    print('%i é maior que 30'%(n))
else:
    print('%i é igual a 30'%(n))
```

```
40 é maior que 30
```

Estruturas de repetição

No python, assim como qualquer linguagem de programação, existe o que chamamos de estruturas de repetição. Elas servem para executar um mesmo comando mais de uma vez, muito útil quando trabalhando com listas.

for

O `for` é uma estrutura de repetição que percorre um escopo de valores específicos, por exemplo se queremos printar os números de 0 até 10 podemos escrever desta forma:

```
for numero in range(0,10):
    print(numero)
```

```
0
1
2
3
4
5
6
7
8
9
```

Note que o primeiro número impresso é o 0 como especificado e o último número o 9. A variável numero armazena o valor de cada iteração do loop, mas o for sempre para antes do último número, que seria o 10, porque o Python é uma linguagem que começa com 0 invés de 1, e descartar esse último número auxilia na hora de iterar listas, a principal razão do uso do laço for. Observe como percorrer os elementos de uma lista:

```
meses = ['jan', 'fev', 'mar', 'abr', 'mai', 'jun']
for i in range(length(meses)):
    print(meses[i])
```

```
jan
fev
mar
abr
mai
jun
```

Por convenção, no python usamos como variável do loop o `i` se quisermos percorrer uma lista. Usar um for dentro de outro serve para percorrer uma matriz. Nesse caso usaremos `j` no segundo. Se mais uma dimensão for adicionada será `k`. Para ordens ainda maiores não existe uma convenção, porém, eu utilizo o alfabeto invés de i, j, k.

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for i in range(length(matriz)):
    for j in range(length(matriz[i])):
        print(matriz[i][j])
```

```
1
2
3
4
5
6
7
8
9
```

Existe uma alternativa ao range de como percorrer uma lista com o for. O range cria esses índices para serem referenciados na lista. Podemos salvar os elementos direto na variável do loop como feito a seguir:

```
meses = ['jan', 'fev', 'mar', 'abr', 'mai', 'jun']
for mes in meses:
    print(mes)
```

```
jan
fev
mar
abr
mai
jun
```

O problema de usar o loop dessa forma é que não podemos alterar a lista original já que perdemos a referência numérica dela. Existem situações que o primeiro método é preferível e situações que o segundo método é preferível.

while

O `while` é um loop que enquanto uma condição não for satisfeita ou enquanto ela for satisfeita esse loop permanecerá ocorrendo, usado principalmente quando o número de iterações do loop não é definido por um número e sim por uma condição.

```
animais = ['cachorro', 'gato', 'rato', 'macaco', 'cachorro', 'cachorro', 'elefante', 'leao
while 'cachorro' in animais:
    animais.remove('cachorro')
print(animais)
```

```
['gato', 'rato', 'macaco', 'elefante', 'leao']
```

Repare que `'cachorro'` foi removido mais de uma vez da lista de animais. Uma vez que não haviam mais cachorros na lista de animais, o loop do while foi finalizado.

break e flags

Podemos interromper o while utilizando o `break` ou flags, como nos exemplos abaixo:

```
# Saudando pessoas até que elas queiram sair do programa com o break
while True:
    resposta = input('Digite seu nome para que eu possa te dar um oi!:\n'+'(Entre com "qui

    if resposta.lower() == 'quit':
        break
    # SE CAIR NA CONDIÇÃO DO `BREAK` AS LINHAS ABAIXO DESTE COMENTÁRIO NÃO SERÃO EXECUTADA
print('Olá, ' + resposta)
```

```
Olá quit
```

Um exemplo com flags que são variáveis booleanas que controlam o fluxo.

```
continuar_execucao = True
while continuar_execucao:
```

```
    resposta = input('Digite seu nome para que eu possa te dar um oi!:\n'+'(Entre com "qui
    if resposta.lower() != 'quit':
        print('Olá, ' + resposta)
    else:
        continuar_execucao = False
```

```
Olá quit
```

continue

Também podemos dar continuidade a um ciclo com a utilização do `continue`.

```
while True:
    resposta = input('Digite seu nome para que eu possa te dar um oi!:\n' + '(Entre com "q
    if resposta == '':
        print('Digite um nome válido!')
        continue
    # SE O USUÁRIO APENAS PRECIONAR ENTER SEM ADICIONAR SEU NOME O CICLO REINICIARÁ E PEDI
    # DIGITE SEU NOME.
    if resposta.lower() == 'quit':
        break
    # SE CAIR NA CONDIÇÃO DO `BREAK` AS LINHAS ABAIXO DESTE COMENTÁRIO NÃO SERÃO EXECUTADA
    print('Olá, ' + resposta)
```

```
Digite um nome válido!
```

Repare que o `print('Olá, ' + resposta)` não foi feito porque na primeira iteração o loop caiu no continue e no segundo no break porque o quit foi digitado.

Funções

Várias vezes na programação repetimos a mesma tarefa várias vezes ou precisamos criar uma tarefa que possa ser usada mais tarde em outro programa, para evitar repetição de código e seguir um dos princípios da programação que é a modulação utilizamos da criação de funções.

Em resumo, funções são um escopo de código que pode ser chamado para realizar uma tarefa específica, podem ter ou não um tipo de retorno, possuem um título e podem ter ou não parâmetros passados para a função.

Em python utilizamos a sintaxe: def nome_da_funcao(parametros): #escopo da função return objeto_a_ser_retornado

Os exemplos mais simples de funções podem ser vistos abaixo:

```
def soma(a, b): #definição/criação da função
    return a + b #escopo da função
soma(3, 4) #chamada da função
```

```
def ola_mundo():
    print('Olá, Mundo!')
ola_mundo()
```

```
Olá, Mundo!
```

Exemplo de uma função com retorno de um objeto do tipo lista:

```
def formata_nomes(nomes):
    lista_formatada = []
    for nome in nomes:
        lista_formatada.append(nome.title())
    return lista_formatada
lista_nao_formatada = ['dAnIeL', 'iGoR', 'caRoLINA']
lista_formatada = formata_nomes(lista_formatada)
print(lista_formatada)
```

```
['Daniel', 'Igor', 'Carolina']
```

Orientação a objetos

No mundo da programação existem diversas linguagens. As linguagens são criadas seguindo paradigmas por vezes distintos, isso é, diferentes lingugagens seguem diferentes formas de se encarar o próprio ato de programar, podendo ainda seguir mais de um paradigma diferente. No python, um dos paradigmas explorados é o de orientação a objetos.

Uma linguagem orientada a objetos escreve seus programas baseados em classes e objetos. Para entender o que são classes e objetos precisamos fazer um paralelo com o mundo real, por exemplo um carro.

Um carro possui características específicas: modelo, cor, velocidade. O carro possui ainda comportamentos: acelerar e freiar, acessíveis pelo motorista para usar quando quiser. Na programação, o carro é o objeto, a ideia do que é um carro é a classe e o motorista é o programador que usa o carro em seu programa.

Traduzindo para python:

```
class Carro():

    def __init__(self, modelo, cor, velocidade):
        self.modelo = modelo
        self.cor = cor
        self.velocidade = velocidade

    def getModelo(self):
        return self.modelo

    def getCor(self):
        return self.cor

    def getVelocidade(self):
        return self.velocidade

    def acelerar(self, incrementoVelocidade):
        self.velocidade = self.velocidade + incrementoVelocidade

    def freiar(self, decrementoVelocidade):
```

```
        if self.velocidade < decrementoVelocidade:
            self.velocidade = 0
        else:
            self.velocidade = self.velocidade - decrementoVelocidade


meuCarro = Carro("Gol", "vermelho", 0)
```

O comando class do python funciona semelhante ao def, porém define uma classe. Dentro da definição da classe, diversas "funções" foram definidas: ___init___, getModelo, getCor, getVelocidade, acelerar, freiar. O nome de funções no contexto de um objeto é método. Métodos são semelhantes a funções, porém utilizam também como variáveis os atributos da classe, isso é, são os comportamentos do carro e usam as características do carro. Na definição, os métodos da classe têm os argumentos começando com o self se forem usar algum atributo do objeto, na hora de utilizar esses metodos, não se escreve o self, uma vez que o metodo que parte do objeto em questão já entende que o mesmo é o "self".

O método ___init___ é um método especial, ele pode ser chamado de construtor. Ele receberá como argumentos as características que julgamos necessárias para definir nosso objeto. Na linha meuCarro = Carro("Gol", "vermelho", 0), o método ___init___ é chamado e retorna um objeto da classe Carro de modelo "Gol", de cor "vermelha" e velocidade 0. Sempre com o nome da classe e os argumentos do método ___init___ da classe em questão (exceto o self), essa função constroi o objeto.

Para usar os métodos precisa digitar a variável em que o objeto está armazenado depois .nomeDoMetodo(argumentos) como segue no exemplo abaixo:

```
print("Velocidade antes de acelerar:", meuCarro.getVelocidade())
meuCarro.acelerar(10)
print("Velocidade depois de acelerar:", meuCarro.getVelocidade())
```

```
Velocidade antes de acelerar: 0
Velocidade depois de acelerar: 10
```

O código do exemplo mudou a velocidade do objeto meuCarro, que originalmente era 0 e passou para 10.

Para acessar os atributos (características) do objeto, normalmente se usa esses métodos do tipo getAtributo(). Pode-se, também, trazendo pro contexto do exemplo, acessar usando apenas meuCarro.velocidade invés de meuCarro.getVelocidade(). Semelhante a como é feito a chamada dos métodos, porém sem o () no final. Embora possa parecer mais simples acessar a velocidade diretamente, sem criar e chamar o método getVelocidade, em orientação a objetos existem algumas práticas que todo o programador deve seguir, essa é uma delas, o

encapsulamento, que consiste em esconder os detalhes da implementação da classe do objeto criado.

Essa regra do encapsulamento e outras regrinhas não são importantes de se saber para o contexto de ciência de dados. Orientação a objetos é toda uma forma de se programar, um assunto extenso, imprescindível no processo de criação de um software, não necessariamente para se usar um. É importante saber o básico para entender o que está acontecendo por baixo de bibliotecas, que comumente utilizam classes. O matplotlib, uma biblioteca de visualização de dados, por exemplo, tem um objeto de uma classe de visualizações como meio de se criar as imagens de visualização de dados, todos os comandos de como criar os gráficos são feitos por métodos desse objeto.

Instalação de bibliotecas

O `pip` é um repositório que guarda uma quantidade enorme de pacotes e módulos para utilizarmos em nossas aplicações. Se o Python estiver adicionado ao PATH digite no terminal do windows:
~~~ $ pip –version ~~~

Caso seu pip esteja desatualizado utilize:

```
$ python -m pip install --upgrade pip
```

Para instalar algum pacote execute:

```
$ pip install "nome do pacote"
```

ou ainda

```
$ python -m pip install "nome do pacote"
```
~~~

Importação

Agora veremos as diferentes formas de importar módulos (os pacotes do python) para seu ambiente.

```
import numpy                          # Importa o módulo numpy
import pandas as pd                   # Importa o módulo pandas com um alias pd
import matplotlib.pyplot as plt       # Importa a classe pyplot dentro do pacote matplotlib
from os import getcwd                 # Importa apenas a função getcwd do módulo os
from time import asctime as data      # Importa a penas a função asctime com o alias data
from statistics import *              # Importa tudo que pertence ao módulo statistics
```

Alias é o "apelido" que o módulo terá no código, ou seja, apenas precisamos chamar o alias para referenciar o módulo

Vejamos agora como cada forma de importação é utilizada!

Veja que o numpy foi importado utilizando `import numpy` dessa forma devemos sempre colocar o nome do módulo antes de utilizar os métodos e atributos que ele possui

```
numpy.random.seed(60)
numpy.random.normal(0, 1, 10)       # Cria um array aleatório de tamanho 10 com distriuiçã
```

```
array([-9.21770993e-01, -5.86317634e-01,  1.16399914e+00, -1.24172396e+00,
       -1.98523022e+00,  1.30670891e+00,  7.37807059e-01,  3.79111282e-01,
        9.89200864e-04, -1.10503482e+00])
```

Já em `from os import getcwd` apenas importamos a função `getcwd` do módulo os

```
getcwd()                            # Retorna o diretório atual do workspace.
```

```
'/home/daniel.dsantos/Documents/personal/git/python-basico/notebooks/01_python_basico'
```

Utilizando o comando `from time import asctime as data` importamos a função asctime do módulo time e adicionamos um *alias* data utilizando a *keyword* `as`.

```
data()                                  # Retorna a data de hoje
```

```
'Wed Nov 13 10:39:42 2019'
```

Ao utilizar `from statistics import *` estamos importando todas as funções e métodos pertencentes ao módulo statistics. Não é aconselhável utilizar já que o código perde semântica.

```
numpy.random.seed(35)
numbers = numpy.random.randint(1, 20, 30)

print(mean(numbers))
print(harmonic_mean(numbers))
print(median(numbers))
```

```
9
4.522366591633217
10.5
```

Referências

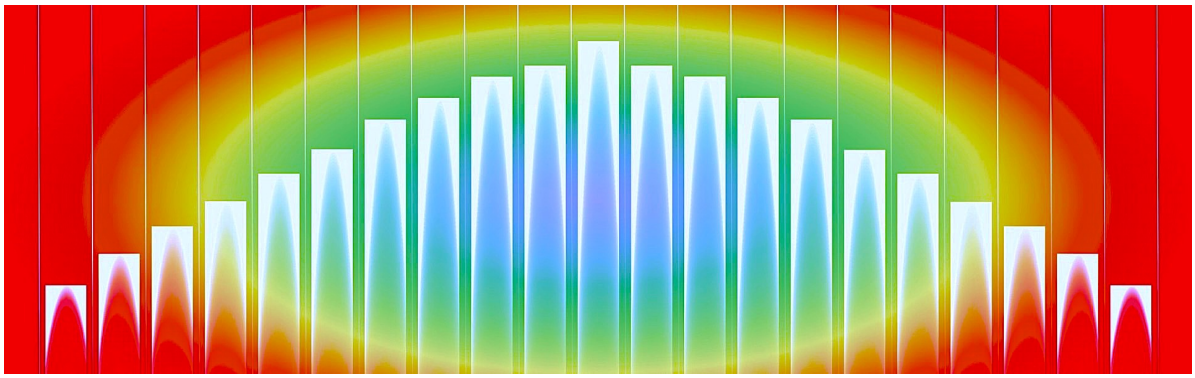
MATTHES, Eric. **Python Crash Course: A Hands-On, Project-Based Introduction to Programming**. San Francisco: No Starch Press, 2016.
LUTZ, Mark. **Learning Python**. Estados Unidos: O'Reilly Media, 2013.

Conceitos Gerais de Estatística

Conteúdos

Distribuições de Probabilidade

Neste capítulo vamos definir algumas distribuições de probabilidade que são consideradas como as mais conhecidas e as mais utilizadas nos processos de construção de Intervalos de Confiança e Testes de Hipóteses. Discutiremos sobre suas definições, seus parâmetros e seus usos no meio prático. Saber identificar se os dados seguem certa distribuição segue como um dos principais passos para se aplicar um teste de hipóteses.



SciPy

O **SciPy** é o pacote básico da linguagem Python que implementa diversas técnicas úteis na computação científica. Utilizaremos essa biblioteca para calcular as probabilidades, construir **Intervalos de Confiança** e aplicar **Testes de Hipóteses**.

Importação

Serão importados os seguintes pacotes para gerar variáveis aleatórias, calcular as probabilidades e plotar gráficos.

```
import numpy.random as nr #Gerador de amostras aleatórias
import numpy as np
import scipy.stats as ss #Principal ferramenta para os cálculos de probabilidades.
from scipy.special import gamma # função Gama
```

```
import matplotlib.pyplot as plt #Construção de Gráficos
```

Variáveis Aleatórias

Dado que o espaço amostral, representado pela letra grega , é definido como o conjunto formado por todos os possíveis resultados de um experimento aleatório, definimos como uma variável aleatória (v.a.) a função que associa cada elemento de a um número real.

Uma variável aleatória pode ser discreta quando o conjunto dos possíveis valores (imagem) for finito ou enumerável, ou contínua quando o conjunto dos possíveis valores for não enumerável, por exemplo intervalos na reta. O foco será nas variáveis aleatórias contínuas.

Distribuições Contínuas As distribuições contínuas descrevem o comportamento dos possíveis valores de uma variável aleatória contínua. Dentre as distribuições contínuas conhecidas, iremos apresentar:

- Distribuição Normal
- Distribuição Exponencial
- Distribuição T-Student
- Distribuição Qui-Quadrado
- Distribuição F

Defina-se a probabilidade de ocorrência sendo a área calculada sob a curva, ou seja, a probabilidade de uma observação assumir um valor entre dois pontos quaisquer equivale a área compreendida entre esses dois pontos. Através disso, dado que X é uma variável aleatória contínua e A um intervalo pertencente aos reais, definimos f sendo sua **função de densidade de probabilidade** (f.d.p) de forma que:

$P(X \in A) = \int_A f(x)dx$

Para que a função f seja uma legítima função de densidade de probabilidade ela deve satisfazer as seguintes propriedades:

- $f(x) \geq 0, \forall x \in \mathbb{R}$
- $\int_{-\infty}^{\infty} f_x(x)dx = 1$

Distribuição Normal

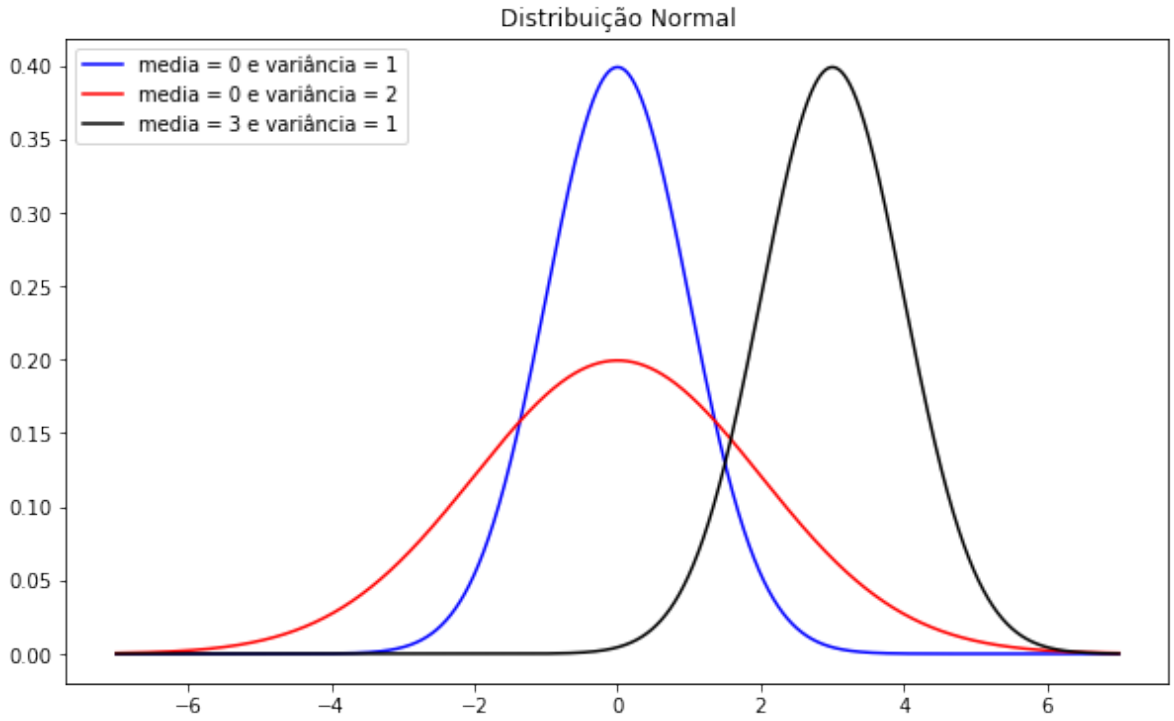
A Distribuição Normal é uma das mais importantes entre as distribuições contínuas. Sua importância se deve a vários fatores, dentre eles temos o Teorema Central do Limite (TCL), o qual é um resultado fundamental em aplicações práticas e teóricas, pois ele garante que, mesmo que os dados não sejam distribuídos segundo uma normal, a distribuição da média amostral dos dados se comporta como uma distribuição normal conforme o número de dados aumenta. Diversos estudos práticos tem como resultado uma distribuição normal, como por exemplo a altura de uma certa população em geral.

Diz-se que, uma variável aleatória contínua X, definida para todos os valores nos reais, tem Distribuição Normal com parâmetros μ e ^{2}, onde $-\infty < \mu < \infty$ e $0 < {}^2 < \infty$, se sua função densidade de probabilidade é dada por:

$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], onde - \infty < x < +\infty$

Os parâmetros e ^{2} são a média e a variância da distribuição Normal, respectivamente. Usaremos a seguinte notação para indicar que X tem distribuição normal com parâmetros μ e ^{2}: X N(μ, ^{2}).

Uma das características da distribuição Normal é que ela é simétrica em torno de sua média , ou seja, há 50% de probabilidade acumulada à esquerda e à direita da média. Outra característica é o seu formato de "sino", a seguir ilustramos esse comportamento através das densidades X ~ N(0,1), X ~ N(0,2) e X ~ N(3,1).



Nota-se que os dois parâmetros influenciam na caracterização das densidades. Assim temos:

- μ como parâmetro de locação, o qual designa o efeito de deslocar horizontalmente o gráfico de f(x).
- como parâmetro de escala, o qual designa o efeito de alongar/contrair o gráfico de f(x).

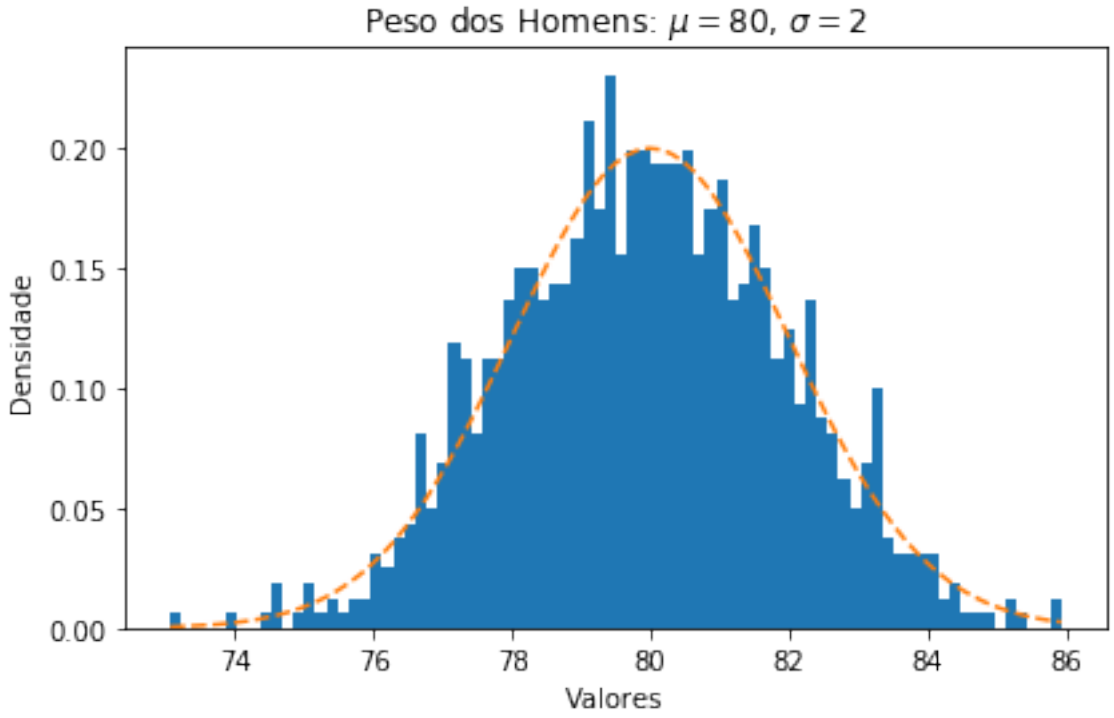
Como exemplo vamos plotar o histograma de uma amostra aleatória de distribuição normal de tamanho 1000 que representa os pesos dos homens de uma população de uma certa cidade, com = 80 e ^{2} = 4 junto a sua curva de densidade.

Para utilizar a função nr.normal() para gerar números aleatórios de uma distribuição normal, deve-se ter atenção aos seus principais argumentos de entrada:

- loc: parâmetro , ou seja, a média dos dados.
- scale: parâmetro , ou seja, o desvio padrão dos dados.
- size: tamanho da amostra.

```
nr.seed(15)
sample=nr.normal(80, 2, 1000)
fig, ax = plt.subplots()
n, x, patches = ax.hist(sample, bins=80, density=1)
```

```
y = ((1 / (np.sqrt(np.pi*8))) *
     np.exp((-(x - 80)**2)/8)) # Densidade de X ~ N(80,4)

ax.plot(x, y, '--')
ax.set_xlabel('Valores')
ax.set_ylabel('Densidade')
ax.set_title(r'Peso dos Homens: $\mu=80$, $\sigma=2$')

fig.tight_layout()
plt.show()
```



Utilizando essa mesma distribuição, onde X ~ N(80,4), vamos calcular algumas probabilidades:

Para calcular a densidade de probabilidade deve ser utilizada a função norm.pdf(). Para exemplificar, iremos calcular a densidade de probabilidade no ponto x = 80, ou seja, $f(x = 80)$

```
ss.norm.pdf(80,80,2)
```

```
0.19947114020071635
```

Para calcular a probabilidade acumulada até um certo ponto, utilizamos a função `norm.cdf()`. Para exemplificar, calcularemos a probabilidade acumulada $P(x < 80)$. .

```
ss.norm.cdf(80,80,2)
```

```
0.5
```

Para calcular um percentil dessa distribuição, utilizamos a função `norm.ppf()`. Para exemplificar, calcularemos o percentil 75, ou seja, achar o k, tal que $P(X \leq k) = 0.75$. .

```
ss.norm.ppf(0.75,80,2)
```

```
81.34897950039216
```

Distribuição Exponencial

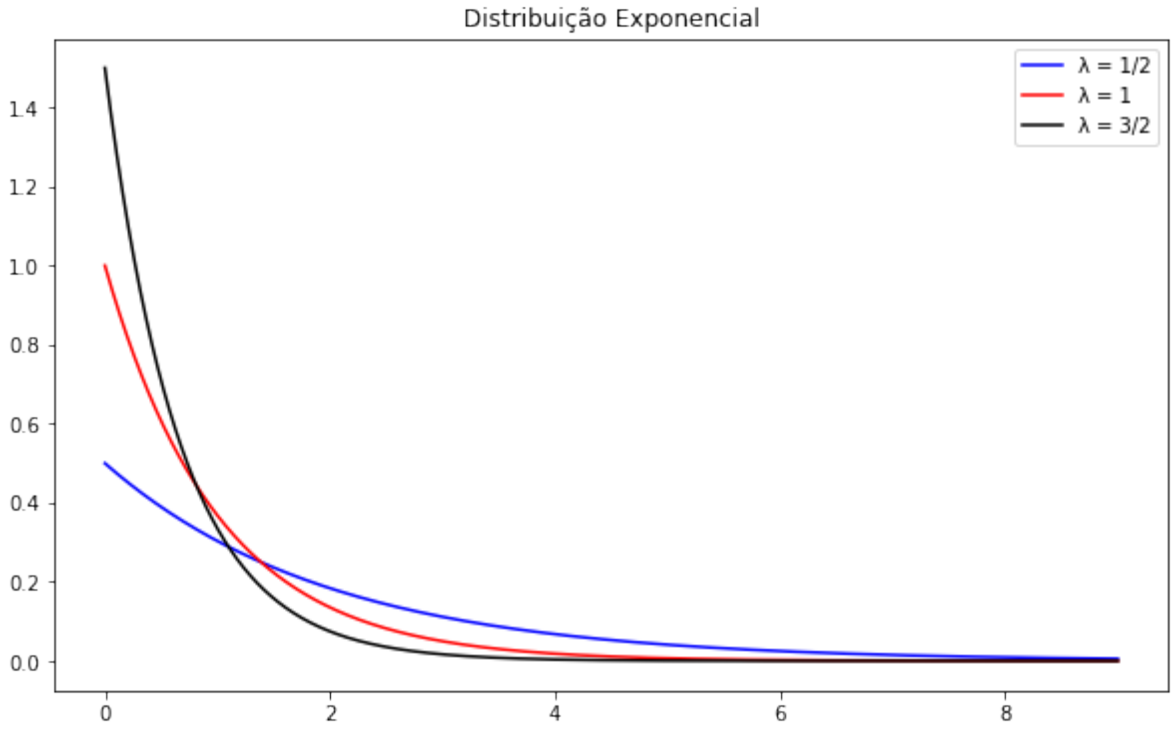
A distribuição exponencial é muito útil para descrever o tempo que se leva para completar uma tarefa ou tempo de duração de um equipamento. Como exemplos de aplicações temos:

- Tempo para realizar uma prova
- Tempo de chegadas de pacotes em um roteador.
- Tempo de vida de aparelhos.
- Tempo de espera em restaurantes, caixas de banco, postos de saúde.

Uma variável aleatória X contínua tem distribuição exponencial com parâmetro > 0 se sua função densidade de probabilidade é dada por:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Onde é o parâmetro da densidade exponencial, e denotamos por X ~ exp( ). Dentre suas propriedades temos que a média da distribuição é igual a 1/ . A seguir podemos visualizar A seguir podemos visualizar o comportamento da densidade para diferentes valores de .



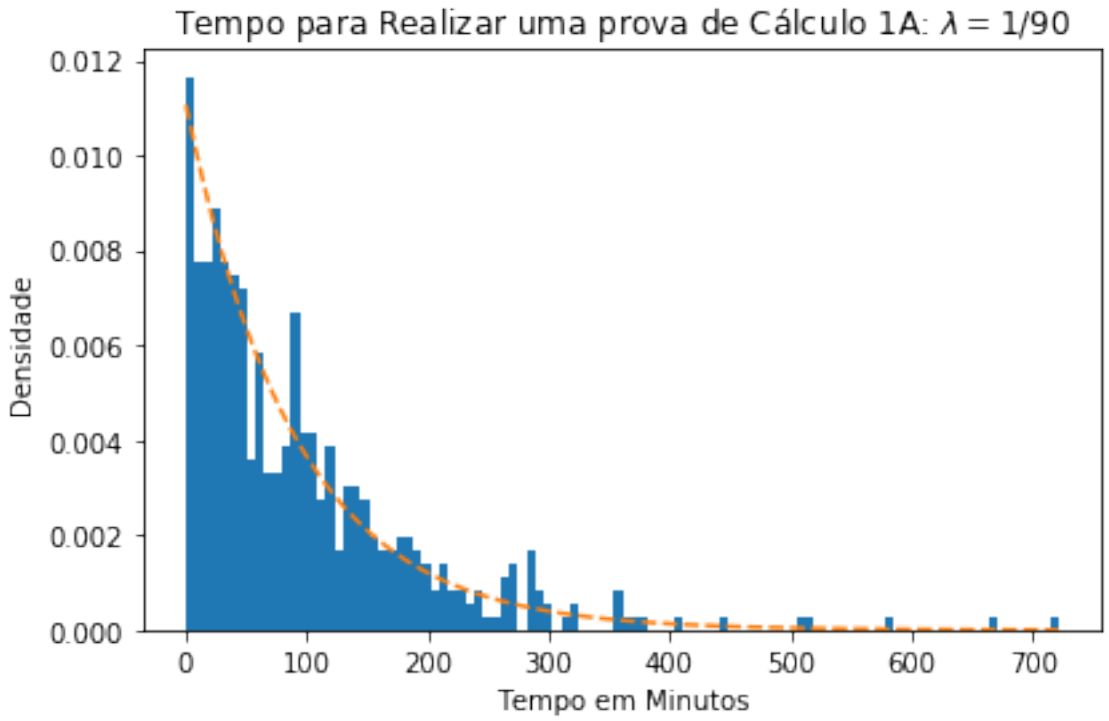
Como exemplo iremos plotar o histograma de uma amostra aleatória de distribuição exponencial de tamanho 500 onde X é o tempo(em minutos) para realizar uma prova de Cálculo 1A na UFF, com = 1/90 junto a sua curva de densidade.

Para gerar números provenientes de uma exponencial com =1/90, utiliza-se a função `nr.exponencial` sendo os principais argumentos de entrada:

- scale: 1/ , ou seja, a média dos dados.
- size: o tamanho da amostra.

```
nr.seed(20)
sample=nr.exponential(90, 500)
fig, ax = plt.subplots()
n, x, patches = ax.hist(sample, bins=100, density=1)

y = (1/90)*np.exp(-(1/90)*x)# Densidade de X ~ exp(1/90)

ax.plot(x, y, '--')
ax.set_xlabel('Tempo em Minutos')
ax.set_ylabel('Densidade')
ax.set_title(r'Tempo para Realizar uma prova de Cálculo 1A: $\lambda=1/90$')
```

```
fig.tight_layout()
plt.show()
```



Utilizando essa mesma distribuição, onde X ~ exp(1/90), vamos calcular algumas probabilidades:

Para calcular a densidade de probabilidade deve-se utilizar a função `expon.pdf()`. Para exemplificar, vamos calcular a densidade de probabilidade em x = 3, ou seja, $f(x = 3)$.

```
ss.expon.pdf(3,scale=90)
```

```
0.010746845560911177
```

Para calcular a probabilidade acumulada até um ponto, utiliza-se a função `expon.cdf()`. Para exemplificar, vamos calcular a probabilidade acumulada $P(x < 200)$:

```
ss.expon.cdf(200,scale=90)
```

```
0.8916319767781041
```

Caso queira calcular o percentil da distribuição utilize a função `expon.ppf()`. Para exemplificar, vamos calcular o percentil 50 dessa distribuição, ou seja, descobrir o k, tal que $P(X \leq k) = 0.5$.

```
ss.expon.ppf(0.5,scale=90)
```

```
62.383246250395075
```

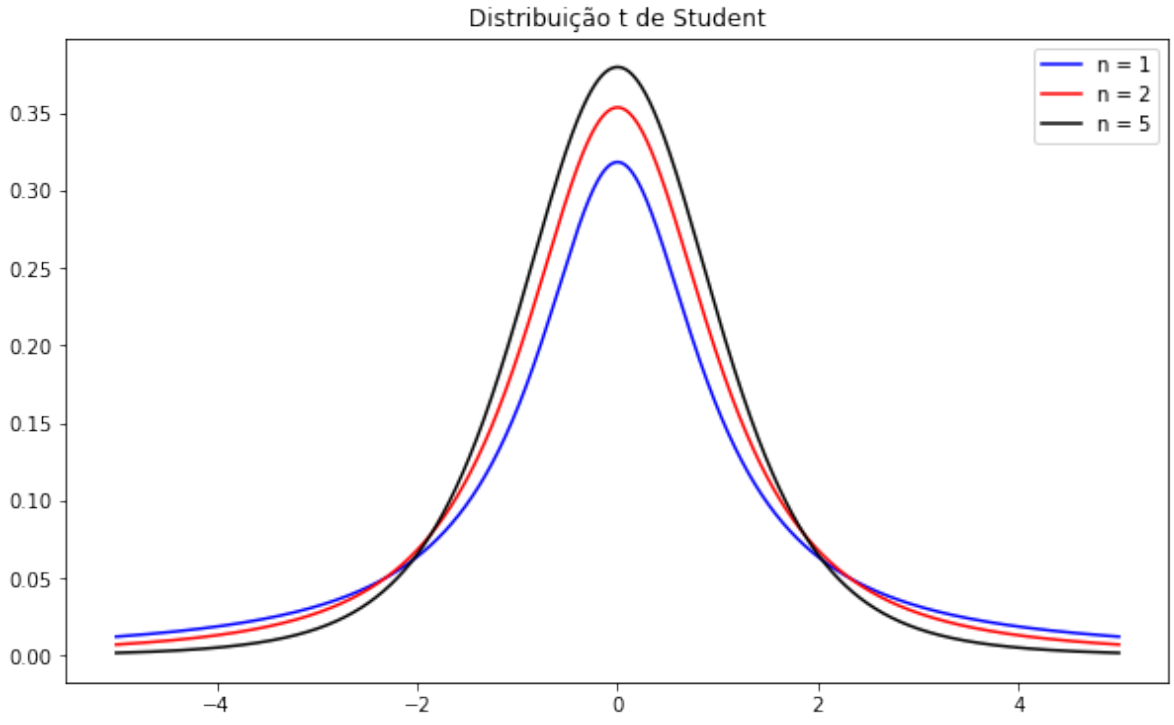
Distribuição T-Student

A distribuição t de Student é considerada uma das distribuições mais utilizadas na estatística com aplicações que vão desde a modelagem de dados até aplicações de Testes de Hipóteses.

Dizemos que X tem distribuição de t de Student com n graus de liberdade se sua função de densidade de probabilidade é dada por:

$f(x) = \frac{\Gamma(\frac{n+1}{2})}{\sqrt{n\pi}\Gamma(\frac{n}{2})}\left(1+\frac{x^2}{n}\right)^{-(\frac{n+1}{2})}, onde -\infty < x < +\infty$

Utilizamos a notação X ~ , onde . A seguir ilustramos a densidade para diferentes graus de liberdade:

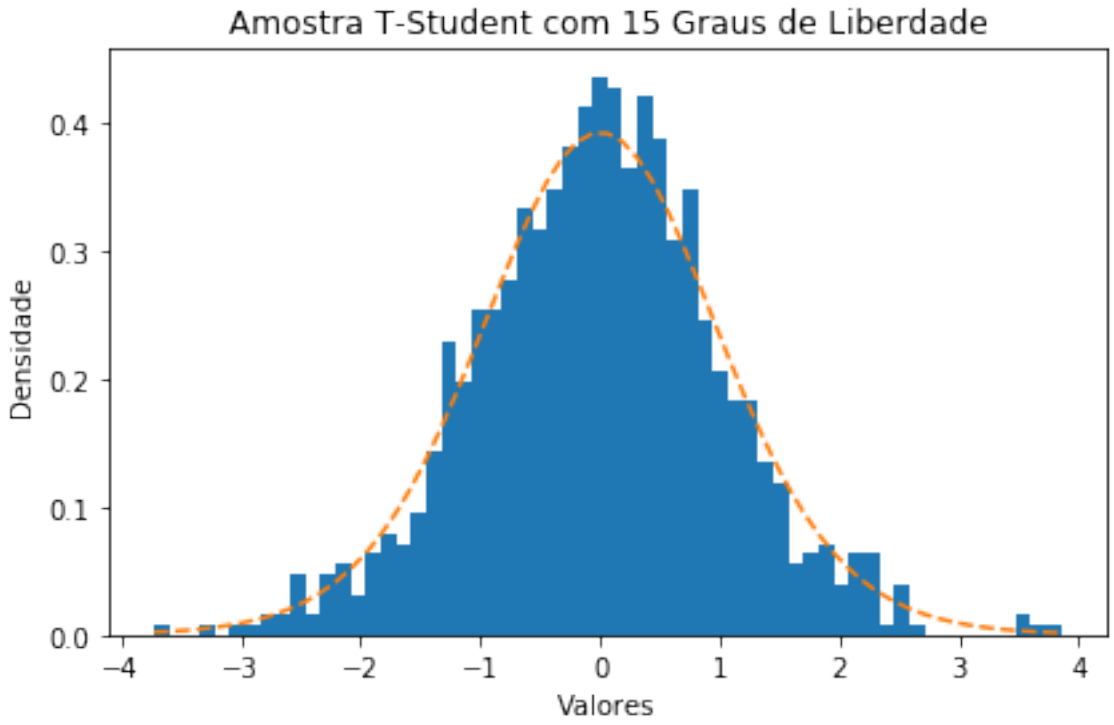


Propriedades:

- Quanto maior o grau de liberdade, mais a distribuição t de Student se aproxima da distribuição Normal.
- Média = 0, caso n>1

Para exemplificar, nós iremos plotar o histograma de uma amostra aleatória de distribuição T de Student de tamanho 1000 com 15 graus de liberdade, junto a sua curva de densidade.

```
nr.seed(10)
n=15
sample=nr.standard_t(n, 1000)
fig, ax = plt.subplots()
tam, x, patches = ax.hist(sample, bins=60, density=1)

y = ( (gamma( (n+1)/2 ))/ (np.sqrt(n*np.pi)* gamma(n/2)) ) * (1 +(( x**2)/n))**(-(n+1)/2)

ax.plot(x, y, '--')
ax.set_xlabel('Valores')
ax.set_ylabel('Densidade')
ax.set_title(r'Amostra T-Student com 15 Graus de Liberdade')

fig.tight_layout()
plt.show()
```



Utilizando essa mesma distribuição, onde X ~ , vamos calcular algumas probabilidades:

Para calcular a densidade de probabilidade deve ser utilizada a função t.pdf(). Para exemplificar, iremos calcular a densidade de probabilidade no ponto x = 0.5, ou seja, $f(x = 0.5)$.

```
ss.t.pdf(0.5,15)
```

```
0.34375457026313216
```

Para calcular a probabilidade acumulada até um certo ponto, utilizamos a função `t.cdf()`. Para exemplificar, calcularemos a probabilidade acumulada $P(x < 1)$. .

```
ss.t.cdf(1,15)
```

```
0.8334149320422619
```

Para calcular um percentil dessa distribuição, utilizamos a função `t.ppf()`. Para exemplificar, calcularemos o percentil 75, ou seja, achar o k, tal que $P(X \leq k) = 0.75$.

```
ss.t.ppf(0.75,15)
```

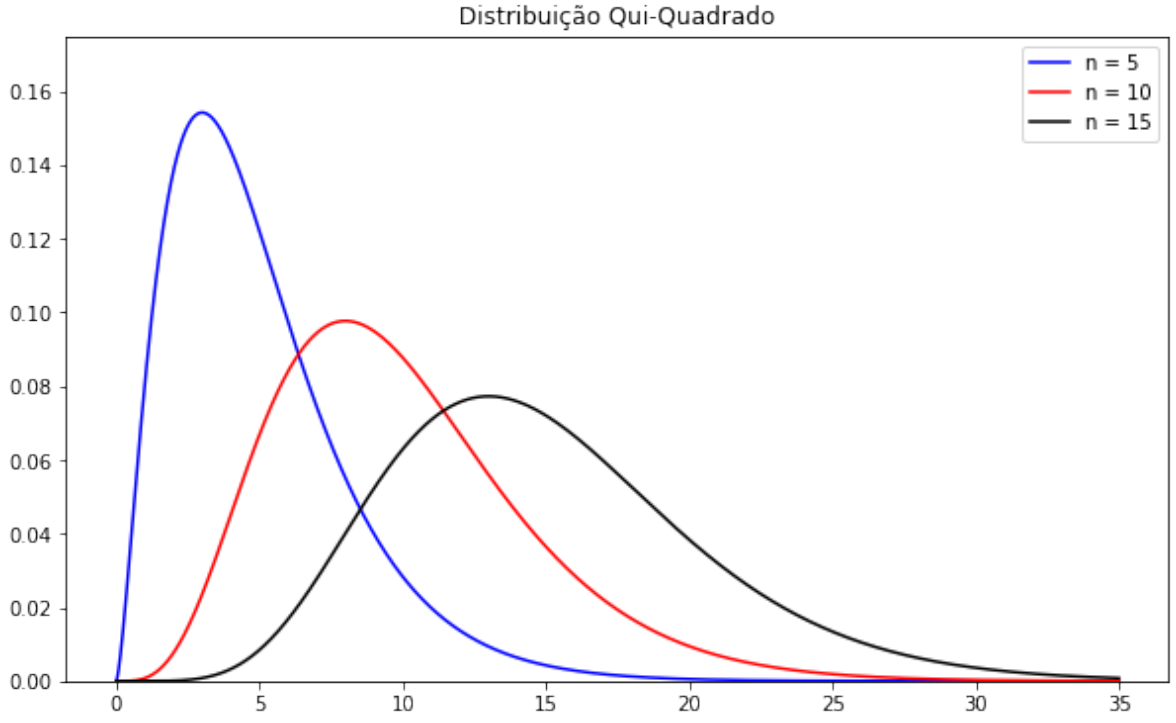
Distribuição Qui-Quadrado

A distribuição qui-quadrado será muito importante para quando tratarmos de Intervalo de Confiança e Teste de Hipótese para variância populacional.

Seja X uma variável aleatória contínua que assume somente valores positivos, temos que X segue uma distribuição Qui-Quadrado com n graus de liberdade se sua função de densidade é dada por:

$f(x) = \frac{1}{2^{\frac{n}{2}}\Gamma(\frac{n}{2})} x^{(\frac{n}{2})-1} exp\left(-\frac{x}{2}\right); n > 0, x > 0$

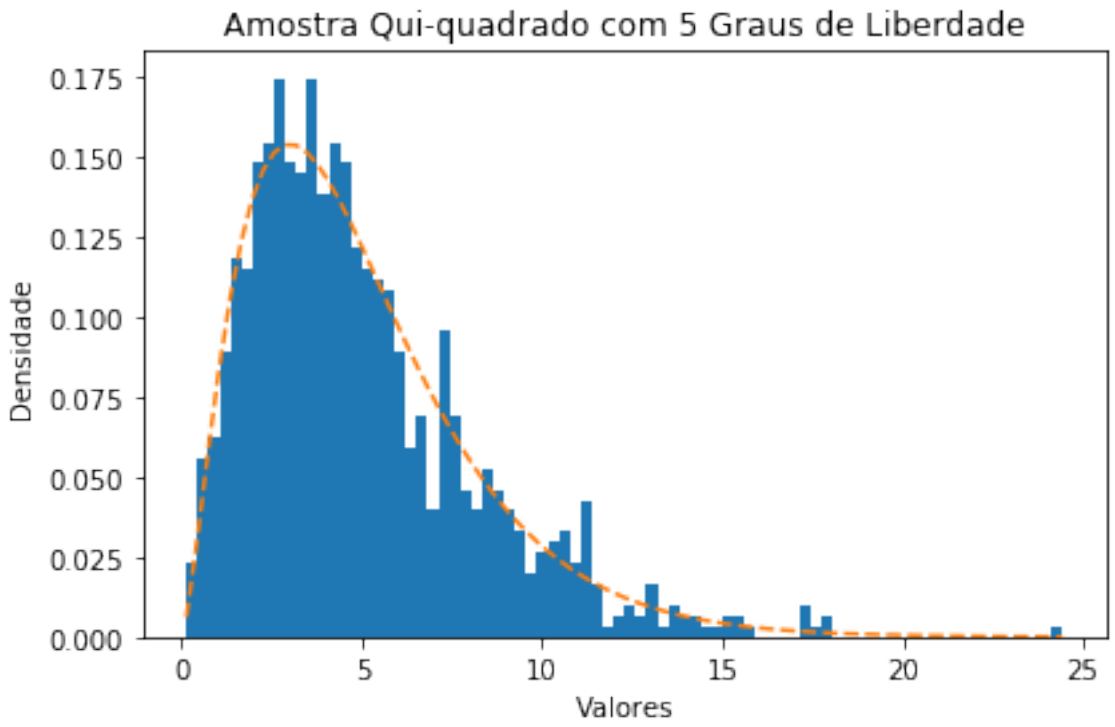
A notação utilizada é $X \sim ^2$. A seguir ilustramos como se caracteriza a densidade com diferentes graus de liberdade:



Após apresentados os gráficos e a fórmula de densidade de probabilidade, temos que a Qui-Quadrado é um caso especial da distribuição Gama com = n/2 e = 2. Para saber mais sobre a Distribuição Gama, clique aqui **(link quebrado)**

Como exemplo, vamos observar uma amostra aleatória de tamanho 1000 proveniente de um distribuição qui-quadrado com 5 graus de liberdade, junto a sua curva de densidade.

```
nr.seed(15)
n=5
sample = nr.chisquare(n, 1000)
fig, ax = plt.subplots()
tam, x, patches = ax.hist(sample, bins = 80, density = 1)

y = (1/(gamma(n/2)*(2**(n/2)))) * (x ** ((n/2) - 1)) * np.exp(-x/2)

ax.plot(x, y, '--')
ax.set_xlabel('Valores')
ax.set_ylabel('Densidade')
ax.set_title('Amostra Qui-quadrado com 5 Graus de Liberdade')

fig.tight_layout()
plt.show()
```



Utilizando essa mesma distribuição, onde X ~ X_5^2 , vamos calcular algumas probabilidades:

Para calcular a densidade de probabilidade deve ser utilizada a função `chi2.pdf()`. Para exemplificar, iremos calcular a densidade de probabilidade no ponto x = 5, ou seja, $f(x = 5)$.

```
ss.chi2.pdf(5, 5)
```

```
0.12204152134938738
```

Para calcular a probabilidade acumulada até um certo ponto, utilizamos a função `chi2.cdf()`. Para exemplificar, calcularemos a probabilidade acumulada $P(x < 4)$.

```
ss.chi2.cdf(4, 5)
```

```
0.4505840486472198
```

Para calcular um percentil dessa distribuição, utilizamos a função `chi2.ppf()`. Para exemplificar, calcularemos o percentil 75, ou seja, achar o k, tal que $P(X \leq k) = 0.75$.

```
ss.chi2.ppf(0.75, 5)
```

```
6.625679763829247
```

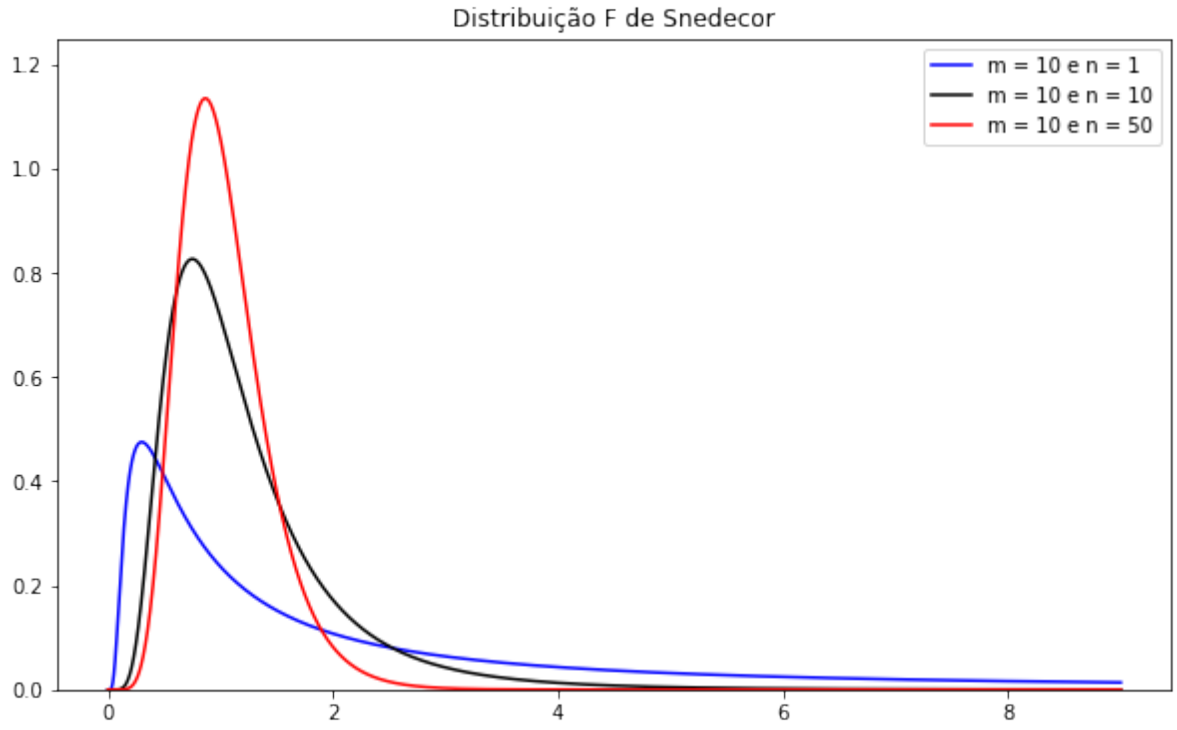
Distribuição F

A distribuição F será fundamental para análise de variâncias que será apresentado nos próximos capítulos.

Uma variável aleatória contínua X possui distribuição F de Snedecor com m e n graus de liberdade se sua função de densidade é definida por:

$$f(x) = \frac{\Gamma(\frac{\nu_1+\nu_2}{2})(\frac{\nu_1}{\nu_2})^{\frac{\nu_1}{2}} x^{\frac{\nu_1}{2}-1}}{\Gamma(\frac{\nu_1}{2})\Gamma(\frac{\nu_2}{2})(1+\frac{\nu_1 x}{\nu_2})^{\frac{\nu_1+\nu_2}{2}}}$$

Neste caso, utilizamos a notação , onde e . A seguir podemos visualizar como se caracteriza a densidade dessa distribuição quando o valor de = 20 e varia entre 1,10 e 50.



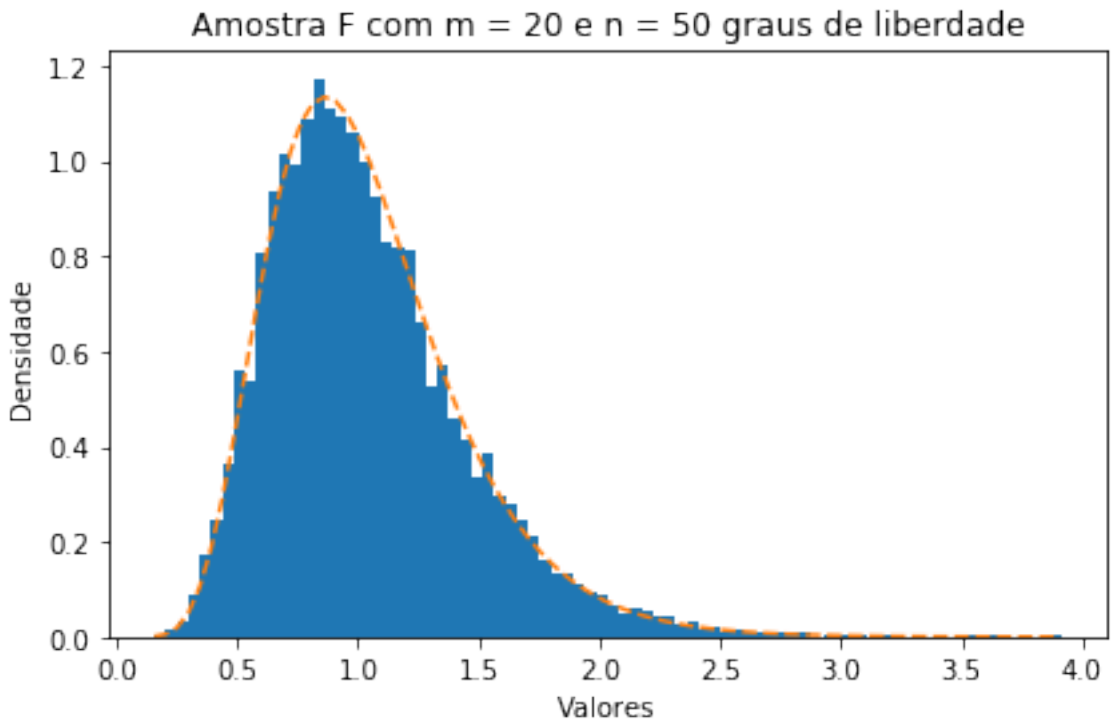
Propriedade: Sejam U e V duas variáveis aleatórias independentes tais que U ~ X^2m e X^2n . Seja:

$$W = \frac{\frac{U}{m}}{\frac{V}{n}} \sim F_{m,n}$$

Temos que W tem distribuição F de Snedecor com m e n graus de liberdade. Assim, tem-se que os graus de liberdade da distribuição F referem-se aos graus de liberdade das duas variáveis Qui-quadrado.

Para exemplificar, vamos observar uma amostra aleatória de tamanho 10000 proveniente de um distribuição F com `m = 20` e `n = 50`, junto a sua curva de densidade.

```
m = 20
n = 50
nr.seed(15)
sample = nr.f(m, n, 10000)
fig, ax = plt.subplots()
tam, x, patches = ax.hist(sample, bins = 80, density = 1)

y = (gamma((m + n)/2)/(gamma(m/2)*gamma(n/2))) * ((m/n) ** (m/2)) * ((x ** ((m/2)-1))/((1

ax.plot(x, y, '--')
ax.set_xlabel('Valores')
ax.set_ylabel('Densidade')
ax.set_title('Amostra F com m = 20 e n = 50 graus de liberdade')

fig.tight_layout()
plt.show()
```



Utilizando essa mesma distribuição, onde $X \sim F_{20,50}$, vamos calcular algumas probabilidades:

Para calcular a densidade de probabilidade deve ser utilizada a função `f.pdf()`. Para exemplificar, iremos calcular a densidade de probabilidade no ponto x = 1, ou seja, $f(x = 1)$.

```
ss.f.pdf(1, 20, 50)
```

```
1.0563663669556977
```

Para calcular a probabilidade acumulada até um certo ponto, utilizamos a função `f.cdf()`. Para exemplificar, calcularemos a probabilidade acumulada $P(x < 1.5)$

```
ss.f.cdf(1.5, 20, 50)
```

```
0.8764757016424363
```

Para calcular um percentil dessa distribuição, utilizamos a função `f.ppf()`. Para exemplificar, calcularemos o percentil 75, ou seja, achar o k, tal que $P(X \leq k) = 0.75$.

```
ss.f.ppf(0.75, 20, 50)
```

```
1.2592037942510066
```

Amostragem Aleatória Simples

Procedimento de seleção de amostras que considera que todas as amostras de tamanho n tem a mesma chance (global) de seleção, podendo ser realizado com reposição ou sem reposição. Uma amostra aleatória de tamanho n de uma variável aleatória X com distribuição de probabilidade f é o conjunto formado pelas variáveis $X_1, X_2, ..., X_n$ independentes e identicamente distribuídas com f.

Estatísticas

Obtida uma amostra aleatória, é possível definir funções reais da amostra aleatória. Como o seu valor depende da amostra sorteada, uma função da amostra aleatória também é uma variável aleatória. Por exemplo, a média amostral é a variável aleatória definida por:

$\bar{X} = \frac{X_1+X_2+X_3+...+X_n}{n}$

Outro exemplo é a variância amostral:

$S^2 = \frac{1}{n-1} \sum_{i=i}^{n} \left(X_i - \bar{X}\right)^2$

Dizemos que uma estatística ou estimador T é qualquer função da amostra $X_1, X_2, \dots, X_n$ que não depende dos parâmetros da distribuição f, isto é,

$T = g(X_1, X_2, \dots, X_n)$

Distribuições Amostrais

A distribuição amostral de uma estatística T é a distribuição de probabilidade associada aos valores que T pode assumir, considerando todas as possíveis amostras de tamanho n extraídas de uma população.

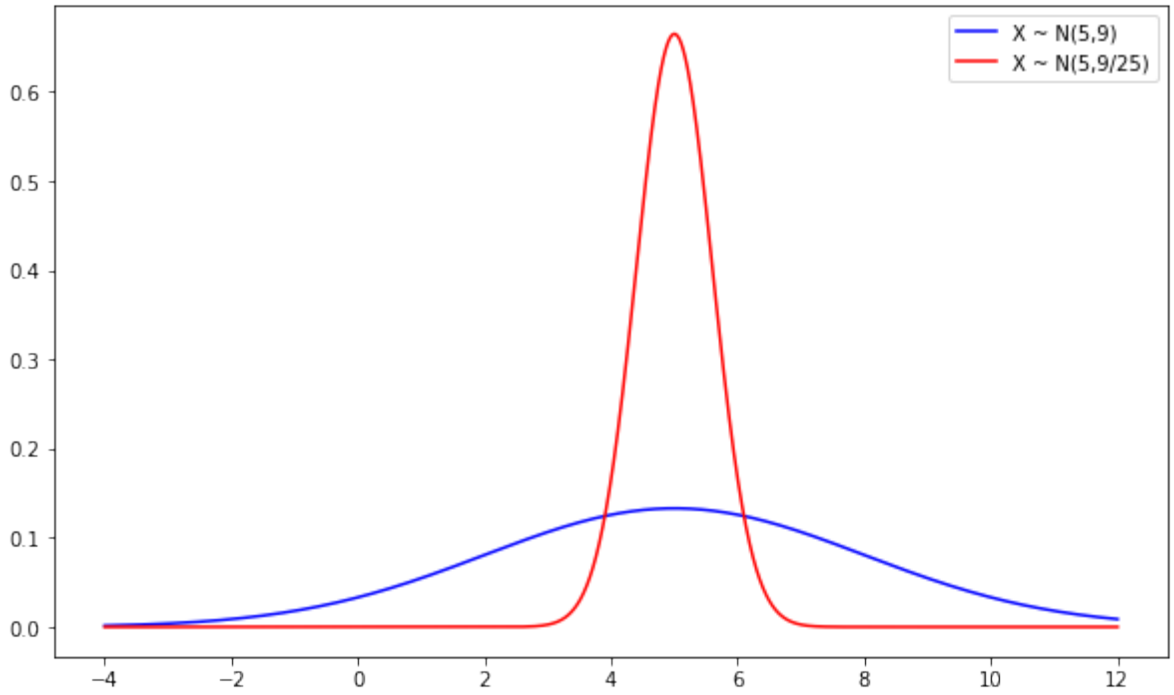
Nos problemas de inferência, estamos interessados em estimar um parâmetro da população (por exemplo, a média populacional) através de uma amostra aleatória simples $X_1, X_2, ..., X_n$. Para isso, usamos uma estatística T (por exemplo, a média amostral) e, com base no valor obtido para T a partir de uma particular amostra, iremos tomar as decisões que o problema exige.

Agora vamos descrever alguns exemplos de distribuições amostrais:

Distribuição da Média Amostral: Seja $X_1, X_2, ..., X_n$ uma amostra aleatória simples de tamanho n de uma população normal, isto é, uma população representada por uma variável aleatória normal $X \sim N(;\sigma^2)$. Então, a distribuição da média amostral $\bar{X}$ é normal com média μ e variância $\frac{\sigma^2}{n}$, ou seja,

$X \sim N(;\sigma^2) \Rightarrow \bar{X} \sim N(;\frac{\sigma^2}{n})$

A seguir ilustramos esse comportamento através de uma amostra aleatória simples de tamanho 25:



Em azul temos uma amostra de tamanho 1 e em vermelho, 25. Veja que os dados ficam mais "concentrados", ou seja, a variância decai conforme o tamanho da amostra aumenta, o que estabelece um formato de "sino" mais estreito ao gráfico. Isso ocorre através do efeito de escala determinado pela variância da distribuição.

Distribuição Amostral de $\sum_{i=1}^{n} \left(\frac{X_i - \mu_i}{\sigma_i}\right)^2$: Seja $X_1, X_2, ..., X_n$ uma amostra aleatória simples de tamanho n de uma população normal, isto é, uma população representada por uma variável aleatória normal $X \sim N(;\sigma^2)$ e S^2 a variância amostral. Então, a distribuição amostral da variância $\frac{(n-1)S^2}{\sigma^2}$ é Qui-Quadrado com $n-1$ graus de liberdade.

$\frac{(n-1)S^2}{\sigma^2} \sim X^2_{n-1}$

Esse resultado será muito importante para os próximos cadernos, quando trataremos do Intervalo de Confiança e Teste de Hipótese para variância.

Referências

FARIAS, A. M. L. KUBRUSLY, J. SOUZA, M.A.O. Probabilidade e Variáveis Aleatórias Unidimensionais. Departamento de Estatística. Universidade Federal Fluminense. Velarde, L. G. C. CAVALIERE, Y. F. Apostila Inferência Estatística. Departamento de Estatística. Universidade Federal Fluminense FARIAS, A. M. L. Apostila de Estatística II. Departamento de Estatística. 2017. Universidade Federal Fluminense.

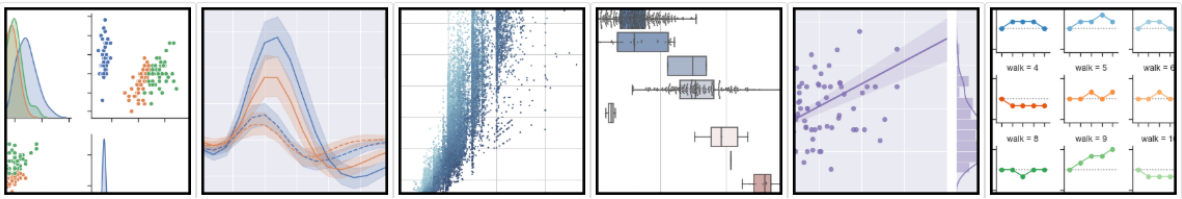
Visualização de Dados

Conteúdos

- Visualização de Dados
- Gráfico de linha
- Gráfico de barras
- Histogramas
- Gráfico de setores
- Boxplot
- Gráficos de dispersão
- Gráficos de dispersão com reta de regressão
- Criando painel

Visualização de Dados

Quando precisamos expôr resultados de forma objetiva e clara, devemos recorrer a recursos visuais. Uma boa visualização de dados se utiliza destes rercursos para chamar à atenção e a compreensão do público diante a análise que foi executada. Tabelas, gráficos, diagramas, etc, são exemplos de visulização. Desta forma, veremos que o Python nos oferece uma gama de opções para desenvolver tais recursos.



Matplotlib

O matplotlib é a principal biblioteca de visualização de dados no Python. Trata-se de uma biblioteca muito completa, sendo ideal para gerar qualquer tipo de gráfico. No entanto, as opções de personalização dos gráficos podem ser maçantes e complexas quando utilizamos apenas o matplotlib para gerar nossas visualizações, e por isso ferramentas como o seaborn facilitam a execução desta tarefa.

Seaborn

A biblioteca seaborn simplifica e abstrai algumas configurações de temas que seriam mais complexas do matplotlib, gerando gráficos mais atrativos com códigos mais limpos e mais simples. Neste notebook, apenas nos prenderemos a utilizar ao Seaborn por sua praticidade.

Importação

import matplotlib.pyplot as plt # É a principal ferramenta capaz de gerar gráficos no python. import seaborn as sns # Baseado no matplotlib o seaborn é uma biblioteca que facilita a criacao de gráficos no # matplotlib. import numpy as np # Estaremos importando o numpy apenas

para executar alguns exemplos. import pandas as pd # Neste notebook apenas utilizaremos para a leitura de csv.

Gráfico de linha

O gráfico de linha é um tipo de gráfico que exibe informações com uma série de pontos de dados chamados de marcadores, ligados por segmentos de linha reta.

Faremos um gráfico da quantidade de divórcios conscedidos em primeira instância o longo dos anos de 2009 a 2017 no Brasil. Para isto, criaremos um gráfico de linha com o intuito de demonstrar a confecção de uma série temporal.

```
# Dados retirados do site SIDRA do IBGE.
anos = [2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017]
divorcios = [171747, 239070, 347583, 341600, 324921, 341181, 328960, 344526, 373216]
sns.lineplot(anos,
         divorcios,
         linewidth=2)

# Configurando título e rótulos dos eixos.
plt.title('Divórcios consensuais concedidos em 1ª instância ou por escritura', fontsize=14
plt.xlabel('Anos', fontsize=12)
plt.ylabel('Quantidade de divórcios', fontsize=12)

# Mudando a grossura dos eixos.
plt.tick_params(axis='both', labelsize=10)

plt.show() # Para 'desenhar' o gráfico
```

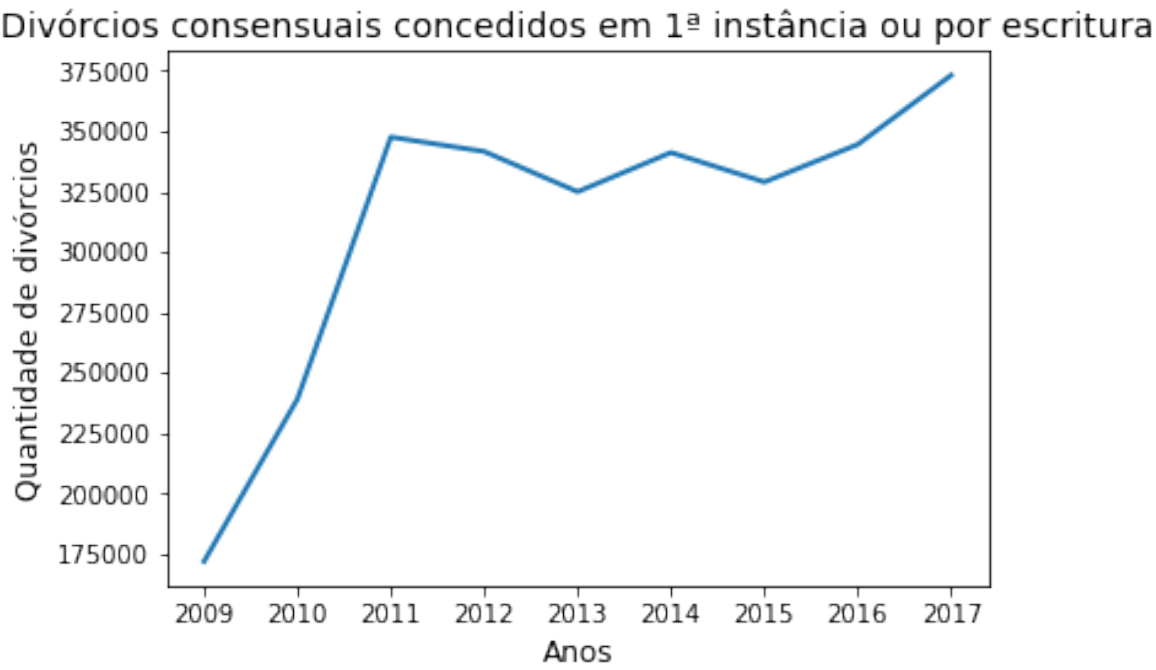

Divórcios consensuais concedidos em 1ª instância ou por escritura
Quantidade de divórcios
375000
350000
325000
300000
275000
250000
225000
200000
175000
2009
2010
2011
2012
2013
2014
2015
2016
2017
Anos

Gráfico de barras

O gráfico de barras é uma ferramenta que possibilita a análise de dados observando o tamanho de barras verticais ou horizontais. Estas barras podem reprensentar frequências e valores de variáveis.

Iremos criar uma classe chamada de "Dado", essa classe imita um dado. Para ser iniciada, devemos passar Dado (Número de faces no dado). Possui um método chamado de "rolar_dado" que simula o lançamento do dado.

```
class Dado:
    """Classe representando um dado de X lados."""

    def __init__(self, n_lados=6):
        """Inicializa um dado com o número de dados igual a n_lados"""
        self.n_lados = n_lados

    def rolar_dado(self, n_lancamentos=1):
        """Lança o dado sorteado uma das faces"""
        return np.random.randint(1, self.n_lados + 1, n_lancamentos)
```

Aqui iremos simular o lançamento de 2 dados com 5 faces e soma-las e também o lançamento de 1 dado com 10 faces. Iremos fazer 10000 lançamentos e comparar a frequência de resultados obtidos.

```
d10 = Dado(10)                                  # Instanciando um dado de 10 lados.
d5 = Dado(5)                                    # Instanciando um dado de 5 lados.

resultados_d10 = list(d10.rolar_dado(10000))


resultados_2d5 =[d5.rolar_dado() + d5.rolar_dado() for lancamento in range(10000)]


# Aqui estamos preenchendo uma lista com a frequência de cada valor ocorridos nos dados.
frequencias_d10 = []
for valor in range(1, 11):
```

```
    frequencia = resultados_d10.count(valor)
    frequencias_d10.append(frequencia)

frequencias_2d5 = []
for valor in range(2, 11):
    frequencia = resultados_2d5.count(valor)
    frequencias_2d5.append(frequencia)

frequencias_2d5.insert(0,0)
```

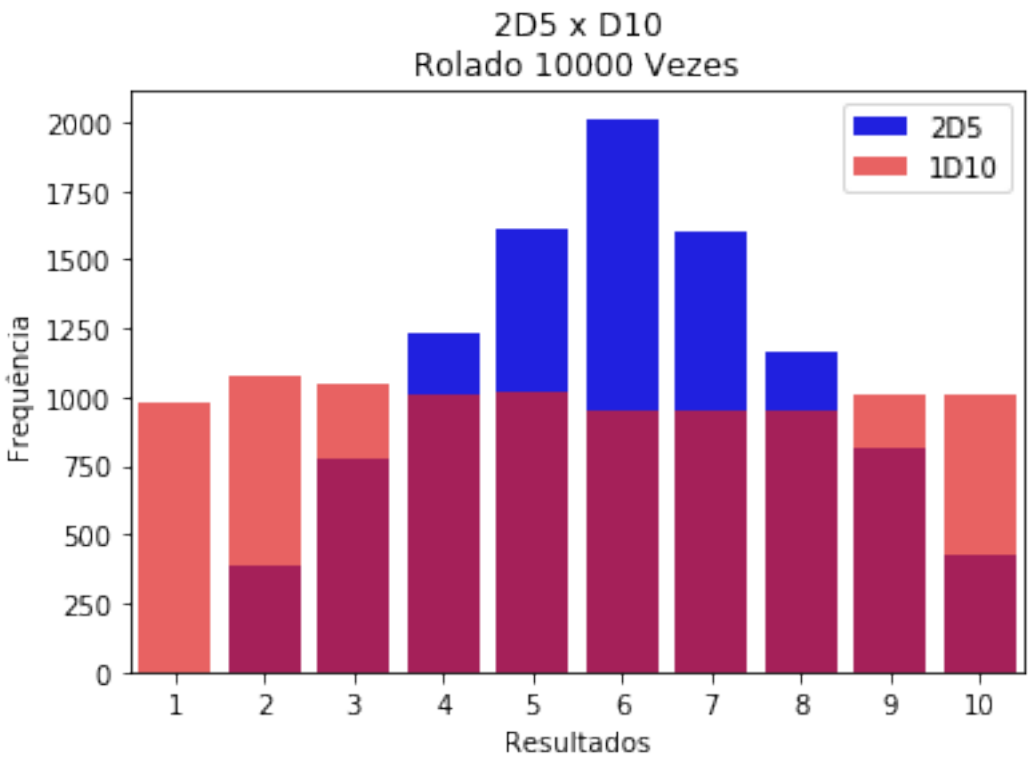
Iremos utilizar o método `sns.barplot()` para dizer ao python para guardar um gráfico de barras em memória. Se utilizarmos o comando duas vezes seguidas para diferentes conjuntos de dados os gráficos ficarão sobrepostos. Após isso, podemos utilizar plt.show() para mostrar os gráficos ao final da linha.

Gerando o gráfico

Para o lançamento de 2 dados de 5 faces.

```
sns.barplot(x=list(range(1,11)),              # `y` recebe o conjunto de valores que corres
            y=frequencias_2d5,                # `y` recebe o conjunto de valores que corres
            color="blue",                     # `color` recebe uma string com o nome de uma
            label='2D5')                      # `label` será o "nome" que este gráfico rece
                                              # gerar uma legenda para nosso gráfico.
plt.title('2D5\nRolado 10000 Vezes')          # Adiciona título ao gráfico
plt.ylabel('Frequência')                      # Adiciona um rótulo ao eixo y
plt.xlabel('Resultados')                      # Adiciona um rótulo ao eixo x
plt.legend()                                  # Adiciona legenda ao gráfico, ele utilizará
                                              # esta legenda.
plt.show()
![](Cadernos%20Grupo%20Python/Caderno%20Visualiza%C3%A7%C3%A3o%20de%20dados/visualizacao_0

sns.barplot(x=list(range(1,11)),
            y=frequencias_d10,
            color="red",
            label = '1D10')

plt.title('1D10\nRolado 10000 Vezes')
plt.ylabel('Frequência')
plt.xlabel('Resultados')
plt.legend()
```

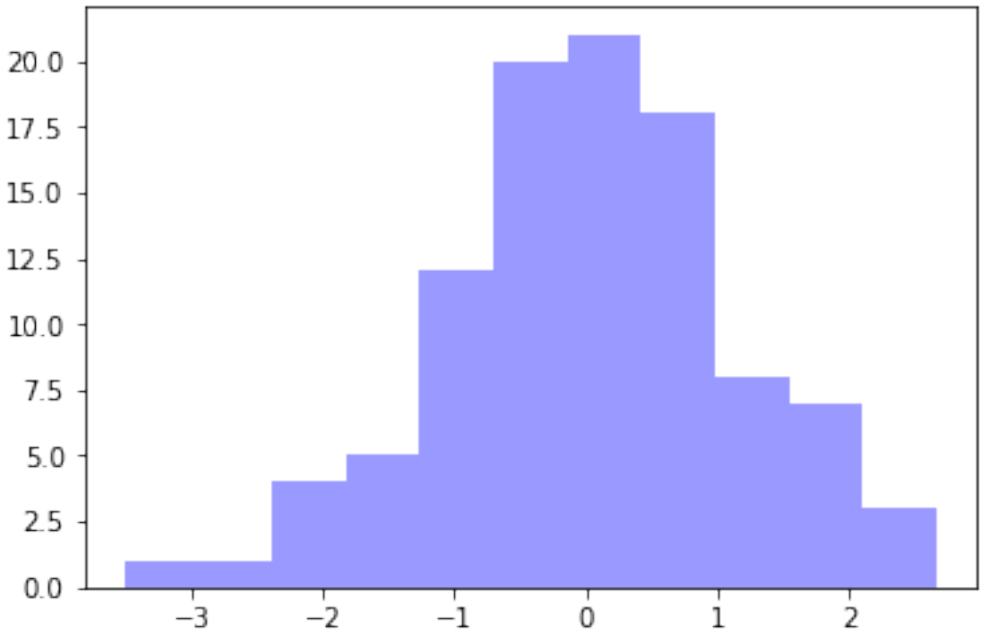
```
plt.show()
![](Cadernos%20Grupo%20Python/Caderno%20Visualiza%C3%A7%C3%A3o%20de%20dados/visualizacao_0

Para colocar os gráficos na mesma imagem, basta adicioná-los um após o outro. O mesmo serv

sns.barplot(x=list(range(1,11)),
            y=frequencias_2d5,
            color="blue",
            label='2D5')

sns.barplot(x=list(range(1,11)),
            y=frequencias_d10,
            color="red",
            alpha=0.7,                      # Adiciona transparência as barras.
            label = '1D10')

plt.title('2D5 x D10\nRolado 10000 Vezes')
plt.ylabel('Frequência')
plt.xlabel('Resultados')
plt.legend()

plt.show()
```


2D5 x D10
Rolado 10000 Vezes
2D5
1D10
Frequência
Resultados
0
250
500
750
1000
1250
1500
1750
2000
1
2
3
4
5
6
7
8
9
10

Histogramas

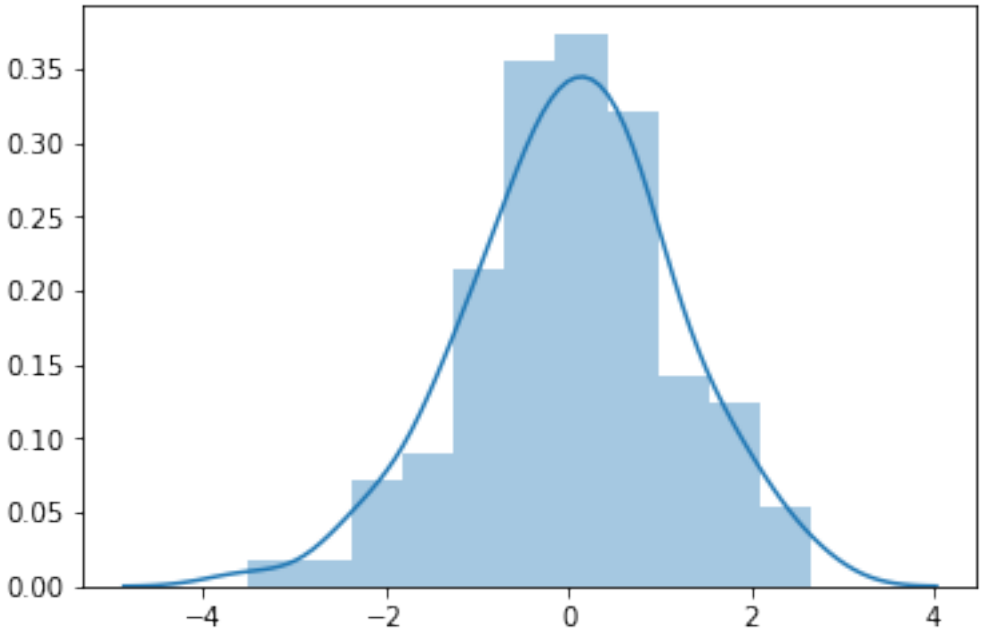
Geraremos uma amostra aleatória de tamanho 100 de uma população normal padrão utilizando a biblioteca numpy, com o intuito de gerar dados para os gráficos.

```
np.random.seed(29)                              # Configura uma "semente" para controlarmos a
amostra_normal = np.random.normal(size=100)     # Gera uma amostra de tamanho 100 de uma dist
                                                # variância 1.
amostra_normal[:10]                             # Visualizar os 10 primeiros valores sorteado
array([-0.41748213,  0.7060321 ,  1.9159847 , -2.1417555 ,  0.71905689,
        0.46707262,  0.76672253,  0.88200945,  0.80875066, -0.94716485])
O método sns.distplot irá gerar um gráfico da nossa distribuição. O argumento kde recebe u

sns.distplot(amostra_normal,
             kde=False,
             color="b")
plt.show()
```



Similar ao código anterior, o próximo gráfico recebe o argumento hist com o valor False isso fará com que o histograma seja omitido e apenas a cruva de densidade distribuição da nossa amostra apareça. O argumento rug também recebe um booleano, em caso de True adiciona rug plot ao nosso gráfico. Rug plot ou gráfico de tapete mostra a distribuição dos dados de forma unidimensional.

```
sns.distplot(amostra_normal,
             hist=False,
             rug=True,
             color="r")
plt.show()

Se desejamos plotar nosso gráfico com o histograma e a curva da densidade, podemos utiliza

sns.distplot(amostra_normal)
plt.show()
```



Se quisermos colocar várias distribuições no mesmo conjunto de eixos.

```
gamma_0 = np.random.gamma(5, 3, 100) # Gerando a amostra
gamma_1 = np.random.gamma(3, 2, 100)
```

```
sns.distplot(gamma_0)
sns.distplot(gamma_1)
plt.show()
```

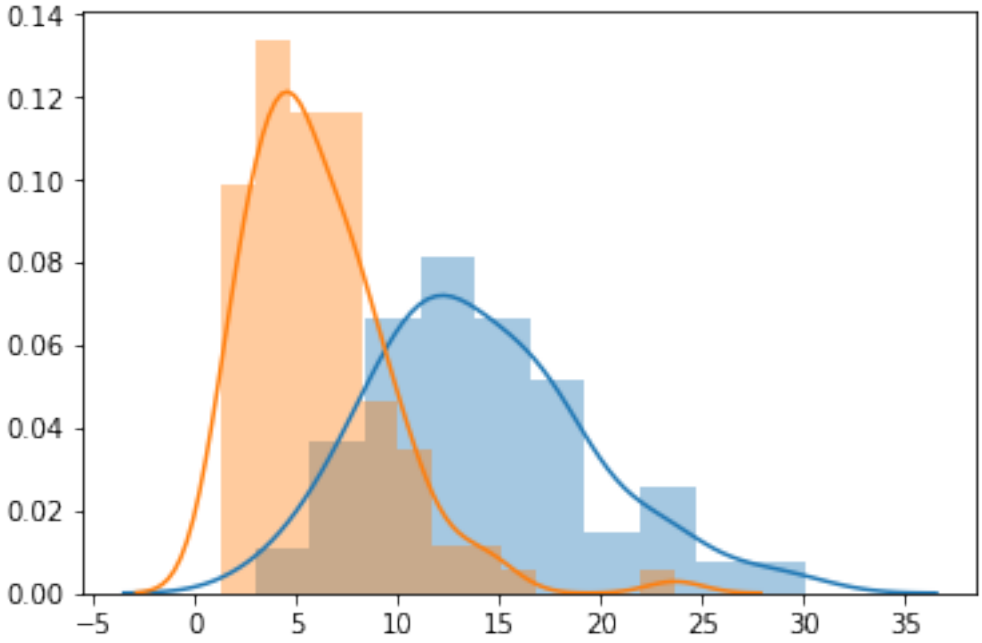
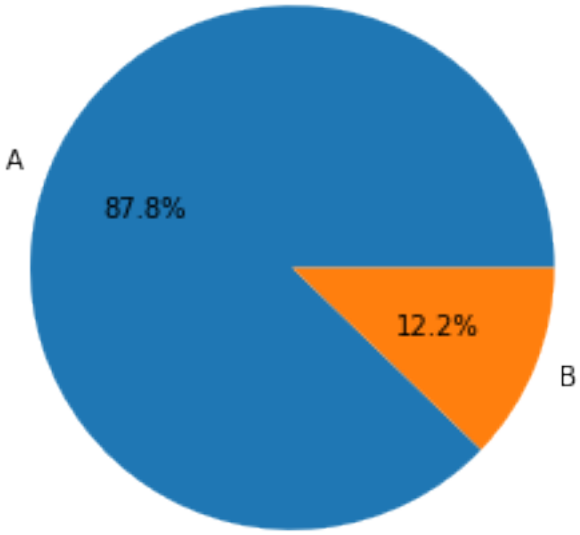


Gráfico de setores

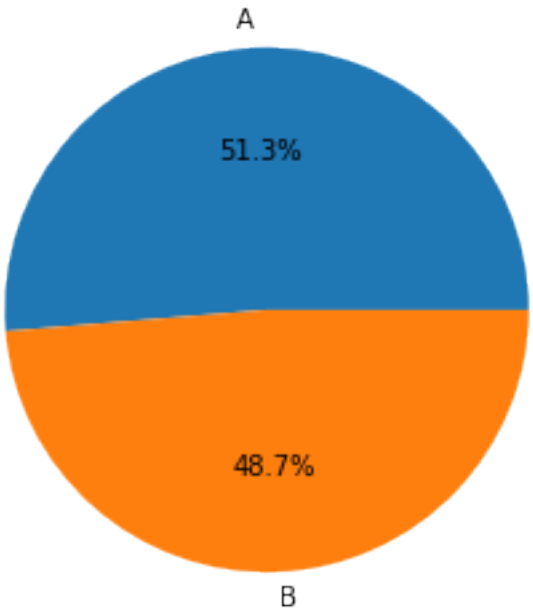
O gráfico de setores, também conhecido como gráfico de pizza, possui a mesma aplicação do gráfico de barras. Ele é geralmente utilizado para ilustrar quando há alta diferença entre frequências das variáveis estudadas. Quando não há essa diferença ou existem muitas divisões entre é preferível utilizar o gráfico de barras.

Para gerar um gráfico de setores devemos primeiro contruir uma lista com as contagens de cada elemento e a outra lista com os rótulos desejados para cada contagem.

```
labels = ['A', 'B']           # Criando rótulos de dois grupos.
contagem = [87.8, 12.2]       # Lista com as proporções.

fig1, ax1 = plt.subplots()    # Criando uma área "plotável"
ax1.pie(contagem,
        labels=labels,
        autopct='%1.1f%%')    # O argumento `autopct` mostra a formatação do texto com as

ax1.axis('equal')
plt.show()
```

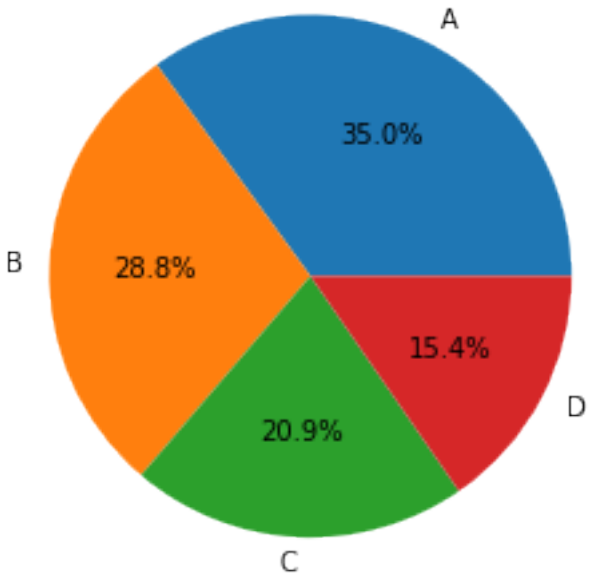



```
labels = ['A', 'B']
contagem = [51.3, 48.7]

fig1, ax1 = plt.subplots()
ax1.pie(contagem,
        labels=labels,
        autopct='%1.1f%%')

ax1.axis('equal')
plt.show()
```



Agora utilizaremos um banco de dados localizado na pasta datasets/ sobre o desempenho de alunos em testes de matemática e leitura e geraremos uma visualização para ele.

```
# Lendo o arquivo
data = pd.read_csv('datasets/StudentsPerformance.csv')
data.keys() # Verificando quais colunas nosso arquivo possui
Index(['gender', 'race/ethnicity', 'parental level of education', 'lunch',
       'test preparation course', 'math score', 'reading score',
       'writing score'],
      dtype='object')
labels = ['A', 'B', 'C', 'D'] # Criando rótulos dos grupos étnicos
# Colocando a contagem de cada elemento de uma coluna em uma lista
contagem = [data['race/ethnicity'].value_counts()[0],
        data['race/ethnicity'].value_counts()[1],
        data['race/ethnicity'].value_counts()[2],
        data['race/ethnicity'].value_counts()[3]]

fig1, ax1 = plt.subplots() # Criando uma área "plotável"
ax1.pie(contagem, labels=labels, autopct='%1.1f%%')
ax1.axis('equal')
plt.show()
```


A
35.0%
B
28.8%
15.4%
D
20.9%
C

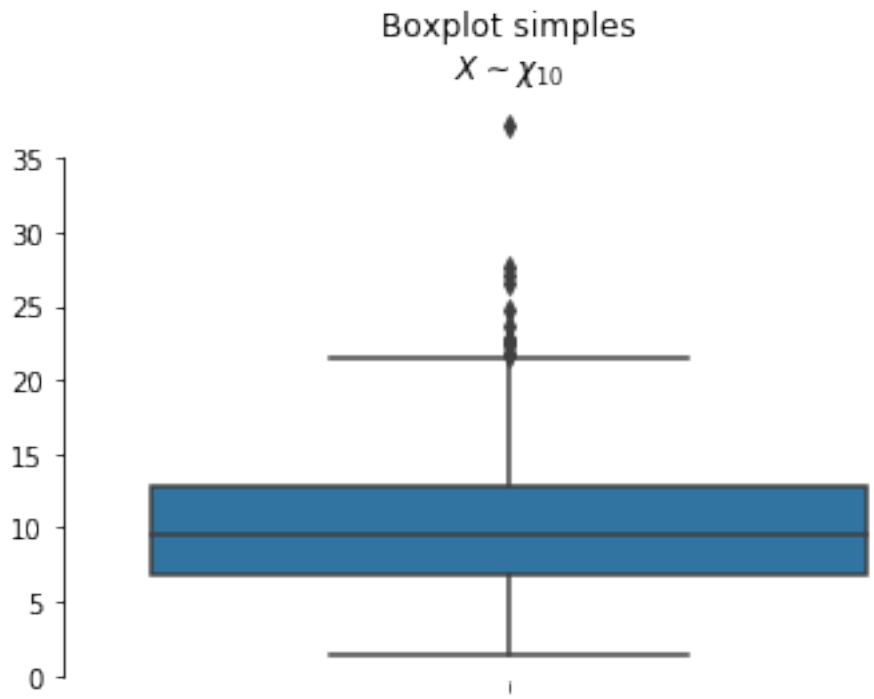
Boxplot

Boxplot é uma ferramenta gráfica para representar a variação de dados observados de uma variável numérica por meio de quartis, amplitude dos dados e possíveis pontos discrepantes. Os chamados 'bigodes', linhas horizontais localizadas nas pontas das caixas representam o máximo e o mínimo dos valores observados não levando em consideração os outliers. Outliers são valores discrepantes observados na nossa amostra. A linha horizontal central se trata da mediana, ou seja, até ela se encontram 50% dos dados observados. Ela é cercada por outras duas linhas o primeiro e terceiro quartis, que correspondem a 25% e 75% dos valores observados.

sns.boxplot é o metodo responsável por criar boxplots. Temos como argumentos os valores de x e y. Além disso, podemos colocar o argumento hue que recebe uma string com o nome de uma coluna com dados categórico, com o intuito de separar nossa observação nesses grupos. palette seleciona a paleta de cores, neste caso magenta e verde.

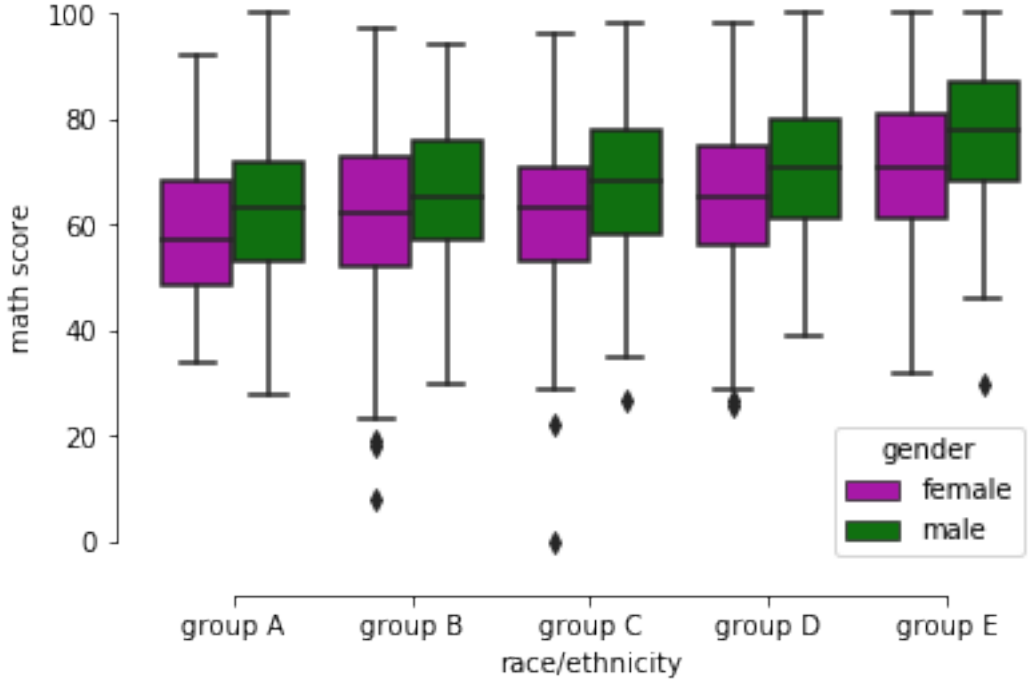
Boxplot simples

```
np.random.seed(1337)
amostra_qui_quadrado = np.random.chisquare(10, 1000)
sns.boxplot(y=amostra_qui_quadrado)
sns.despine(trim=True)                                  # `sns.despine` controla a estetéti
                                                        # remove o contorno da caixa de grá

plt.title('Boxplot simples\n$X\sim\chi_{10}$')          # Veja que podemos utilizar Latex n
plt.show()
```



Boxplot complexos

```
sns.boxplot(x="race/ethnicity",                                          # Eixo X rece
            y="math score",                                              # Eixo Y rece
            hue="gender",                                                # Hue colore
                                                                         # neste caso
            palette=["m", "g"],                                          # Paleta de c
            order=['group A', 'group B', 'group C', 'group D', 'group E'],  # Trocando a
                                                                         # no eixo X.
            data=data)                                                   # Dados

sns.despine(offset=10, trim=True)
```

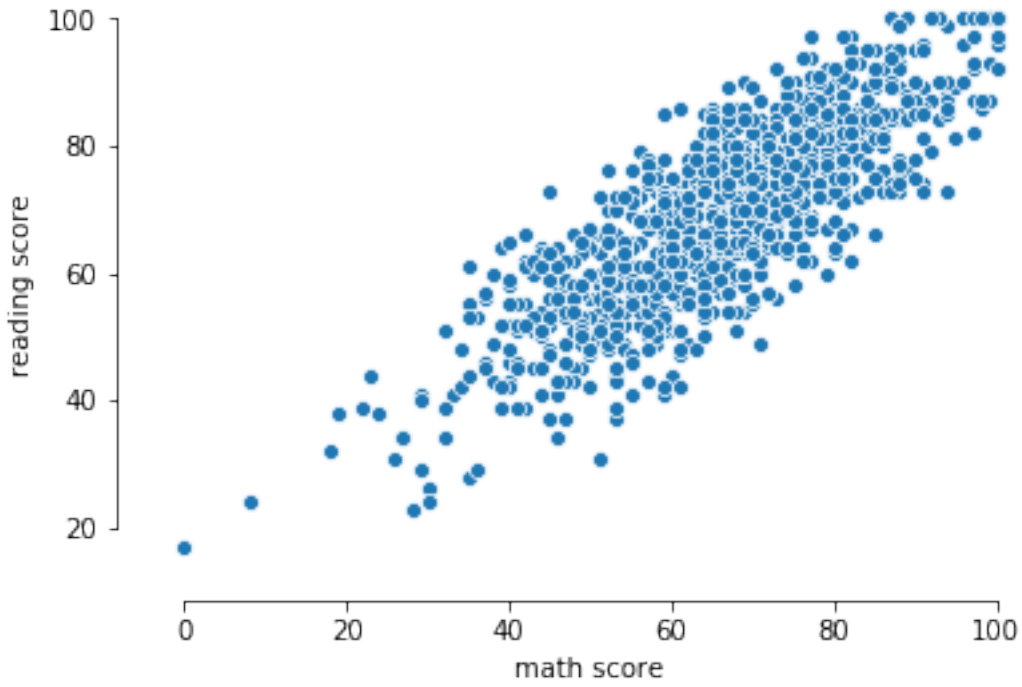

100
80
60
40
20
0
math score
gender
female
male
group A
group B
group C
group D
group E
race/ethnicity

Gráficos de dispersão

O gráfico de dispersão utiliza duas variáveis quantitativas nos eixos x e y para desenhar pontos para cada observação da amostra.

Para gerar um simples gráfico de dispersão podemos utilizar o método sns.scatterplot apenas passando as veriáveis de interesse que desejamos comparar.

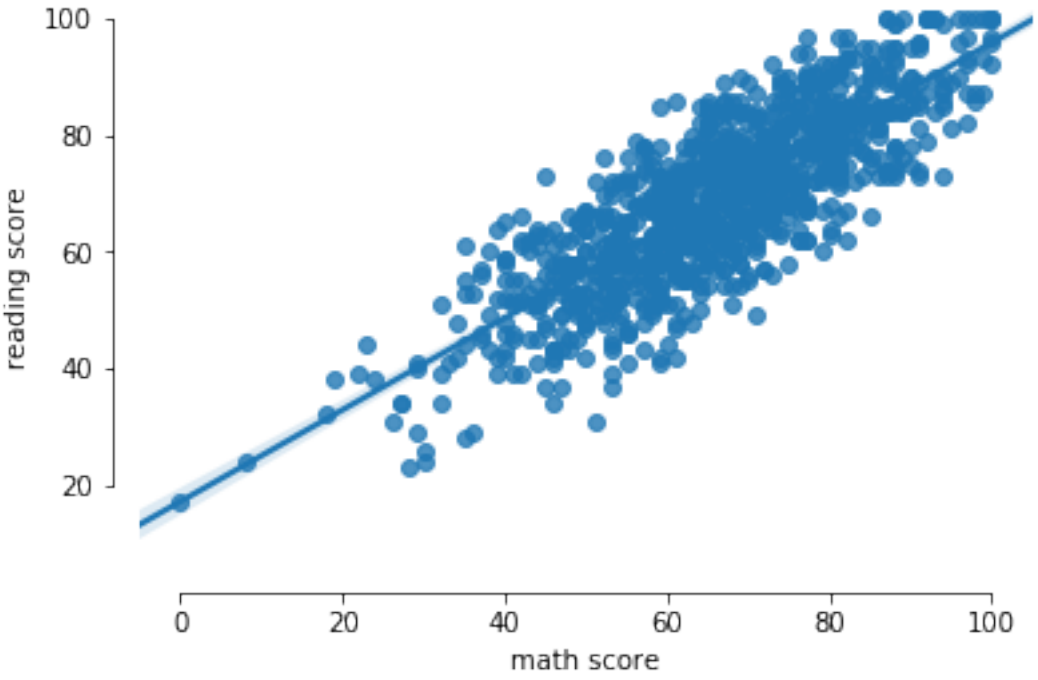
```
sns.scatterplot(x="math score",
                y="reading score",
                data=data);

sns.despine(offset=10, trim=True)
```



Gráficos de dispersão com reta de regressão

O método sns.regplot também recebe como argumentos as variáveis que desejamos estudar a correlação, sendo muito semelhante ao método que gera um gráfico de dispersão.

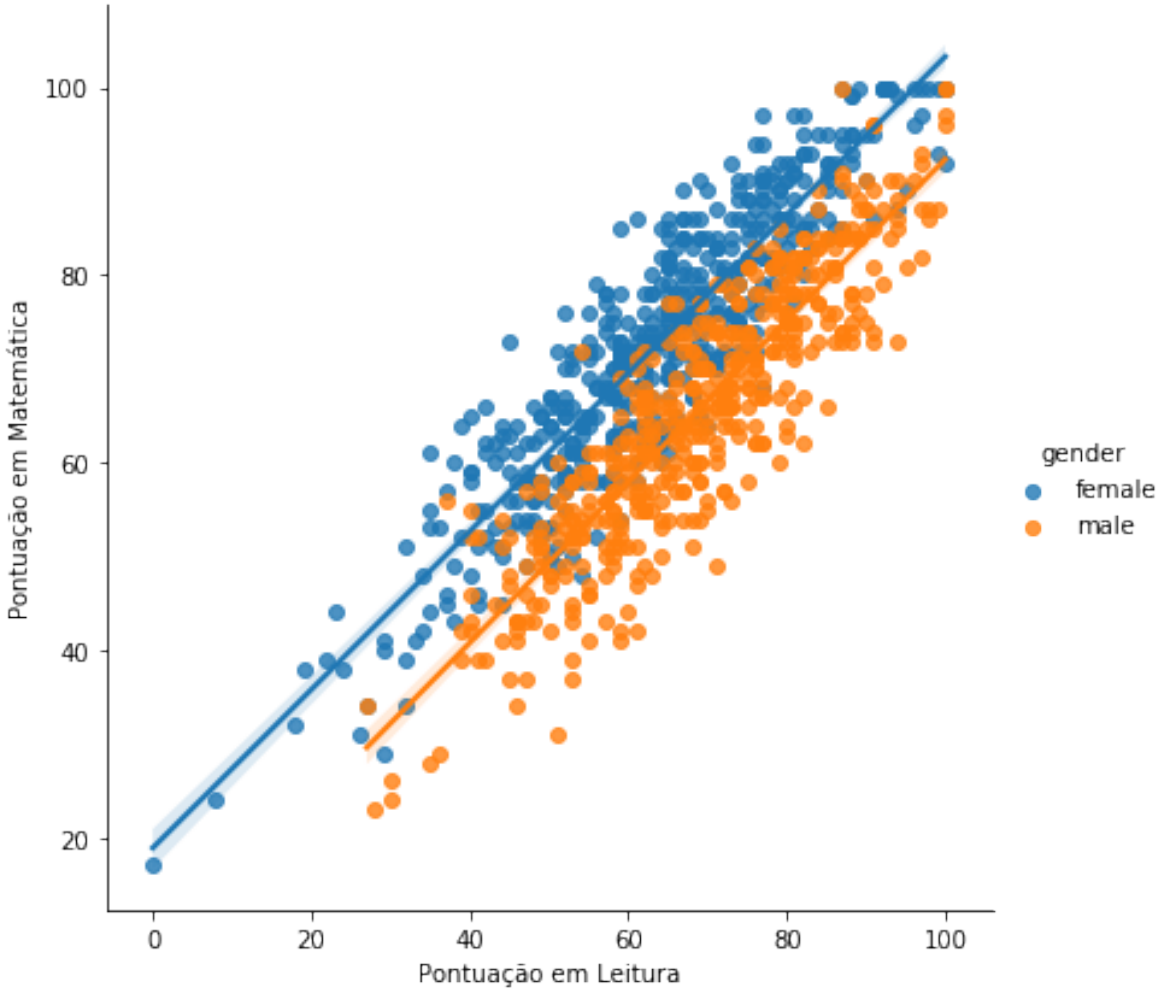
sns.regplot(x="math score", y="reading score", data=data); sns.despine(offset=10,



trim=True)

Agora vejamos como é simples criar em uma punica imagem um gráfico de regressão que separa em nossa amostra em dois grupos, neste caso o gênero.

```
g = sns.lmplot(x="math score", y="reading score", hue="gender", # Bastou adicionar o argum
               # categórias
               truncate=True, height=6, data=data)

g.set_axis_labels("Pontuação em Leitura", "Pontuação em Matemática")
plt.show()
```

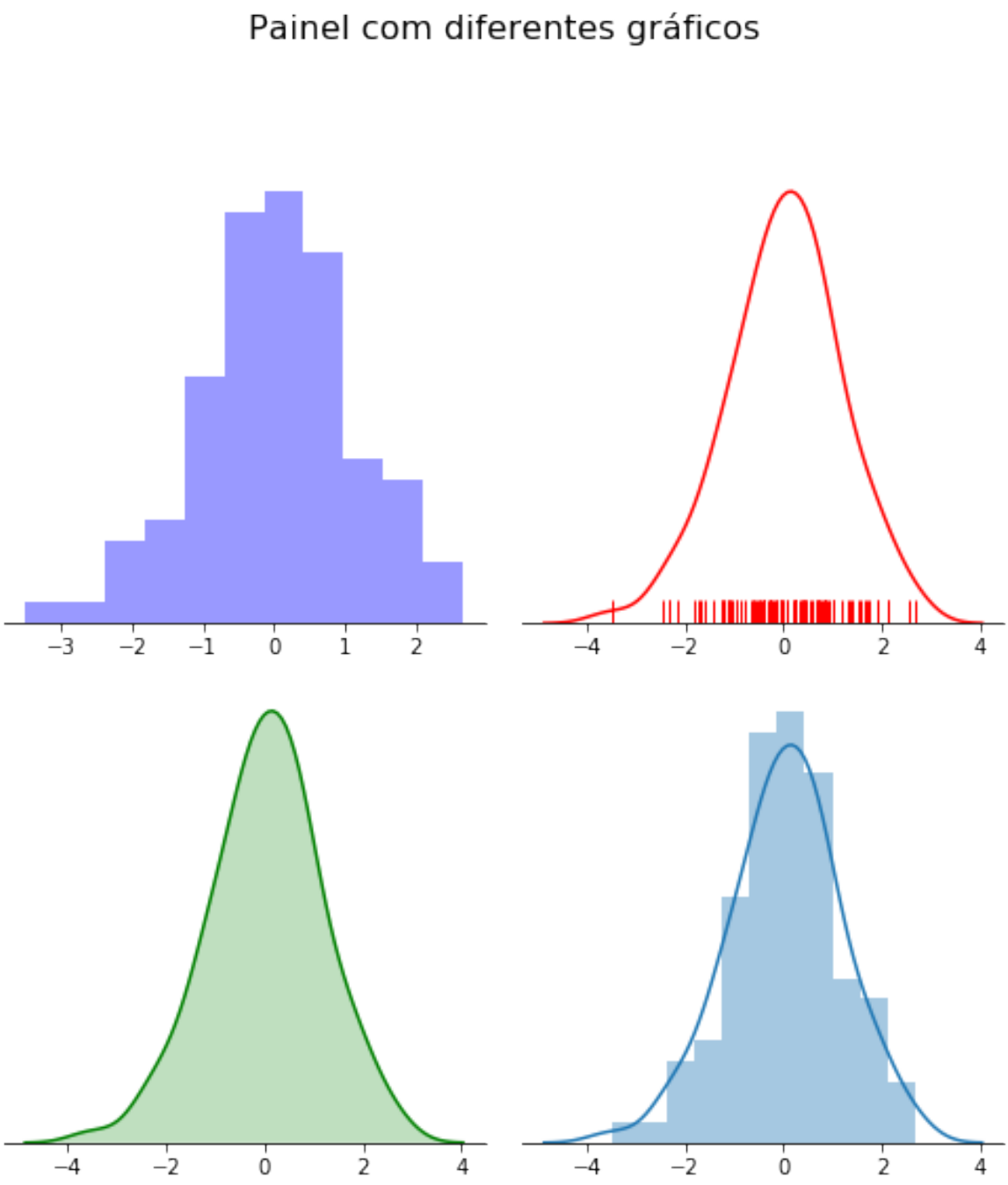

Pontuação em Matemática
Pontuação em Leitura
gender
female
male
100
80
60
40
20
0
20
40
60
80
100

Criando painel

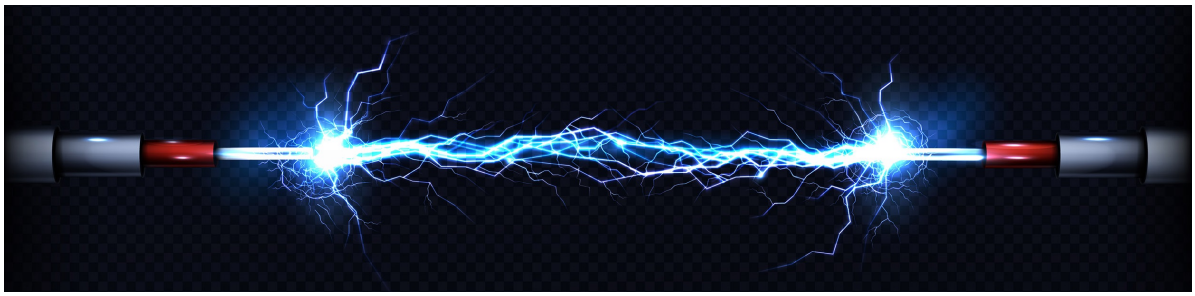
Utilizando o método plt.subplots() gera um "quadro" para que alocar os nossos gráficos. Os argumentos nrows e ncols determinam a número de linhas e colunas do nosso painel, respectivamente. O argumento figsize recebe um array com a altura e largura dos quadros.

```
f, axes = plt.subplots(2, 2, figsize=(7, 7))
sns.despine(left=True)                                  # Retira as bordas dos quadros.

np.random.seed(29)
amostra_normal = np.random.normal(size=100)             # Gerando uma amostra aleatória de ta
                                                        # normal padrão.

# A partir daqui podemos criar os gráficos normalmente alocando-os nos seus devidos "espaç
sns.distplot(amostra_normal,
             kde=False,
             color="b",
             ax=axes[0, 0])                             # Em `ax` devemos passar uma lista co
                                                        # que queremos para este gráfico. Nes
                                                        # primeiro espaço para gráfico, ou se
                                                        # esquerdo do quadro.

sns.distplot(amostra_normal,
             hist=False,
             rug=True,
             color="r",
             ax=axes[0, 1])

sns.distplot(amostra_normal,
             hist=False,
             color="g",
             kde_kws={"shade": True},
             ax=axes[1, 0])

sns.distplot(amostra_normal,
             ax=axes[1, 1])
```

```
plt.setp(axes,
         yticks=[])
plt.suptitle('Painel com diferentes gráficos',
             y = 1.12,
             fontsize = 16)
plt.tight_layout()
```


Painel com diferentes gráficos
−3
−2
−1
0
1
2
−4
−2
0
2
4
−4
−2
0
2
4
−4
−2
0
2
4

Estimativa por Intervalos



Biblioteca statsmodels

A biblioteca **statsmodels** é um módulo Python que fornece classes e funções para a estimativa de muitos modelos estatísticos diferentes, bem como para a realização de testes estatísticos e a exploração de dados.

Importação

```
import statsmodels.api as sm
import scipy.stats as ss
import pandas as pd
import numpy as np
```

Intervalos de Confiança

O objetivo central da Inferência Estatística é obter informações para uma população a partir do conhecimento de uma única amostra. Em geral, a população é representada por uma variável aleatória X, com função de distribuição ou densidade de probabilidade f_x que depende de um parâmetro desconhecido. Se $X_1, X_2, ..., X_n$ é uma amostra aleatória dessa população, então as estatísticas $\hat{\theta}_1$ e $\hat{\theta}_2$ formam um intervalo de $100 \times (1 - \alpha)$ confiança para , se:

$P(\hat{\theta}_1 \leq \theta \leq \hat{\theta}_2) = 1 - \alpha$

Na prática, temos apenas uma amostra e, assim, é importante que se dê alguma informação sobre essa possível variabilidade do estimador. Ou seja, é importante informar o valor do estimador $\hat{\theta}$ obtido com uma amostra específica, mas é importante informar também que o verdadeiro valor do parâmetro poderia estar em um determinado intervalo, digamos, no intervalo $[\hat{\theta} - \varepsilon, \hat{\theta} + \varepsilon]$. Dessa forma, informamos a nossa margem de erro no processo de estimação, com essa margem sendo a consequência do processo de seleção aleatória da amostra.

Nesse material será apresentado como obter esse intervalo, de modo a "acertar na maioria das vezes", isto é, mostraremos um procedimento que garanta que, na maioria das vezes (ou das amostras possíveis), o intervalo obtido conterá o verdadeiro valor do parâmetro. Em outras palavras, com alta probabilidade (em geral, indicada por 1−), o intervalo $[\hat{\theta} - \varepsilon, \hat{\theta} + \varepsilon]$ conterá o verdadeiro valor do parâmetro .

Com probabilidade alta (em geral, indicada por $1 - \alpha$), o intervalo $[\hat{\theta} - \varepsilon; \hat{\theta} + \varepsilon]$ conterá o verdadeiro valor do parâmetro , ou seja, o procedimento de construção garante uma alta probabilidade (1−) de se obter um intervalo que contenha o verdadeiro valor do parâmetro.

$1 - \alpha$ é chamado nível de confiança, enquanto o valor é conhecido como nível de significância. O intervalo $[\hat{\theta} - \varepsilon; \hat{\theta} + \varepsilon]$ é chamado de intervalo de confiança de nível $1 - \alpha$.

OBS: Mesmo representado através de uma probabilidade, não podemos interpretar o intervalo de confiança da forma: "A probabilidade de pertencer ao intervalo [$\hat{\theta}1$, $\hat{\theta}2$] é maior ou igual a 1- .", pois **não é variável aleatória**.

Exemplo: Em um estudo sobre o Índice de Massa Corporal (IMC), foi reportado o seguinte intervalo de confiança de 95% para o IMC médio µ da população da cidade de Niterói - RJ, com base em uma amostra de 650 pessoas: $[26,8 - 0,6; 26,8 + 0,6]$. Dessa forma podemos afirmar que com 95% de confiança esse intervalo contém o verdadeiro valor do IMC médio da população residente de Niterói - RJ.

Intervalos de Confiança para a Média

Seja $X_1, X_2, \ldots, X_n$ uma amostra aleatória proveniente de uma população $X \sim N(, \sigma^2)$ com variância desconhecida, temos que:

$X \sim N(; \sigma^2) \Rightarrow \bar{X} \sim N(; \frac{\sigma^2}{n}) \Rightarrow T = \frac{\bar{X}-\mu}{\frac{S}{\sqrt{n}}} \sim t_{n-1}$

Logo,

Ou seja, o intervalo de confiança para µ de nível de confiança 1 − é:

$\left[\bar{X} - t_{n-1;\alpha/2} \cdot \frac{S}{\sqrt{n}}; \bar{X} + t_{n-1;\alpha/2} \cdot \frac{S}{\sqrt{n}}\right]$

onde $t_{n-1,\alpha/2}$ é o valor crítico da distribuição t-Student com n−1 graus de liberdade que deixa área $\frac{\alpha}{2}$ acima dele.

Para realizar a construção de um intervalo de confiança para a média, devemos utilizar a função `t.interval()` da biblioteca **scipy.stats**, cujos argumentos de entrada são:

- `alpha`: nível de confiança, ou seja, 1- . (Sim, alpha = 1 - α)
- `df`: tamanho dos graus de liberdade, ou seja, n-1.
- `loc`: $\bar{x}$, ou seja, a média dos dados da amostra.
- `scale`: $\frac{S}{\sqrt{n}}$.

Para demonstrar aplicações dessa função utilizaremos o banco de dados a seguir, que representa uma amostra aleatória simples de clientes de um banco alemão.

```
df = pd.read_csv('german_credit_risk_target.csv')
df.head()
```

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Du
1	1	22	female	2	own	little	moderate	5951	48
10	10	25	female	2	rent	little	moderate	1295	12
11	11	24	female	2	rent	little	little	4308	48
12	12	22	female	2	own	little	moderate	1567	12
14	14	28	female	2	rent	little	little	1403	15

Exemplo 1: Supondo que a idade das mulheres segue uma distribuição normal, vamos estimar a idade média das mulheres dessa população através da amostra de 310 mulheres contida nesse banco de dados. Para isso calcularemos um intervalo de 95% de confiança.

```
df2 = df[df['Sex'] == "female"] # Filtrar apenas os dados das mulheres
df2.head()
```

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Du
1	1	22	female	2	own	little	moderate	5951	48
10	10	25	female	2	rent	little	moderate	1295	12
11	11	24	female	2	rent	little	little	4308	48
12	12	22	female	2	own	little	moderate	1567	12
14	14	28	female	2	rent	little	little	1403	15

```
ss.t.interval(alpha = 0.95, df = len(df2['Age'])-1, loc = np.mean(df2['Age']), scale = ss.
```

```
(31.48960295935143, 34.1168486535518)
```

Logo, podemos concluir que com 95% de confiança o intervalo $[31.4896; 34.1169]$ contém o verdadeiro valor da idade média das mulheres dessa população.

Exemplo 2: Supondo que a idade das pessoas pertencentes a empresa 3 segue uma distribuição normal, vamos estimar a idade média dessa população através da amostra de 148 pessoas presentes no banco de dados. Para isso calcularemos um intervalo de 99% de confiança.

```
df3 = df[df['Job'] == 3] # Filtrar apenas os dados das pessoas que trabalham na empresa 3
df3.head()
```

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Du
7	7	35	male	3	rent	little	moderate	6948	36
9	9	28	male	3	own	little	moderate	5234	30
18	18	44	female	3	free	little	moderate	12579	24
34	34	33	female	3	own	little	rich	1474	12
40	40	30	male	3	own	quite rich	NaN	2333	30

```
ss.t.interval(alpha = 0.99, df = len(df3['Age'])-1, loc = np.mean(df3['Age']), scale = ss.
```

```
(36.463454992835125, 41.590599061218924)
```

Portanto, podemos concluir que com 99% de confiança o intervalo $[36.4635; 41.5906]$ contém o verdadeiro valor da idade média dos trabalhadores da empresa 3.

Intervalos de Confiança para a proporção populacional

Seja $X_1, X_2, ..., X_n$ uma amostra aleatórial proveniente de uma população $X \sim Ber(p)$, logo $\hat{p} \sim N(p, p(1-p))$, $\hat{p}$ estimador de p, então:

$\frac{\hat{p}-p}{\sqrt{\frac{p(1-p)}{n}}} \sim N(0,1)$

Logo,

Como não sabemos o parâmetro populacional p, temos duas opções para substitui-lo no Intervalo de Confiança:

- $\hat{p}$ no lugar do p;
- maximizar o valor de p(1 - p) => p = 0,5 (conhecido como IC conservador)

Para realizar a construção de um intervalo de confiança para a proporção, utilizaremos a função proportion_confint() da biblioteca statsmodels.stats, cujos argumentos de entrada são:

- `count`: a quantidade de ocorrências.
- `nobs`: o tamanho da amostra.
- `alpha`: nível de significância.

Exemplo 3: Utilizando a mesma base do exemplo anterior, vamos construir um intervalo de confiança para a proporção de pessoas com `Risk` classificado como `good`.

```
df['Risk'] = df['Risk'].map({'good' : 1, 'bad' : 0}) # transformando a variável Risk para
```

```
sm.stats.proportion_confint(count = sum(df['Risk']), nobs = len(df['Risk']), alpha = 0.05)
```

```
(0.6715974234910674, 0.7284025765089325)
```

Desse modo, temos que com 95% de confiança o intervalo $[0,6716; 0,7284]$ contém o verdadeiro valor da proporção de clientes avaliados como `good`.

Intervalo de Confiança para a variância populacional

Seja $X_1, X_2, ..., X_n$ uma amostra aleatória proveniente de uma população $X \sim N(, \sigma^2)$, então $\frac{(n-1)S^2}{\sigma^2} \sim X^2_{n-1}$

Logo,

Nem o `statmodels` e nem o `scipy` possuem funções que geram intervalo de confiança para a variância. Então, vamos construir a nossa!

```
def variancia_int(amostra, alfa):

    n = len(amostra)
    s_2 = np.var(amostra, ddof=1)
    gl = n - 1

    limite_inferior = (n - 1) * s_2 / ss.chi2.ppf(1 - alfa / 2, gl)
    limite_superior = (n - 1) * s_2 / ss.chi2.ppf(alfa / 2, gl)

    print('(%.16f, %.16f)' % (limite_inferior, limite_superior))
```

Exemplo 4: Agora, vamos construir um intervalo de confiança para a variância da idade dos homens utilizando os dados da amostra.

```
df_homens = df[df['Sex'] == "male"]
```

```
variancia_int(df_homens['Age'], 0.05)
```

```
(108.9517847841306661, 134.5924067459885407)
```

Assim, podemos concluir que com 95% de confiança o intervalo $[108,9518; 134,5924]$ contém o verdadeiro valor da variância dos homens dessa população alvo.

Referências

FARIAS, A. M. L. Apostila de Estatística II. Departamento de Estatística. 2017. Universidade Federal Fluminense. Velarde, L. G. C. CAVALIERE, Y. F. Apostila Inferência Estatística. Departamento de Estatística. Universidade Federal Fluminense

Teste de Hipóteses - Parte 1

Na teoria de estimação, vimos que é possível, por meio de estatísticas amostrais adequadas, estimar parâmetros de uma população, dentro de um certo intervalo de confiança. Nos testes de hipóteses, em vez de se construir um intervalo de confiança no qual se espera que o parâmetro da população esteja contido, testa-se a validade de uma afirmação sobre um parâmetro da população. Então, em um teste de hipótese, procura-se tomar decisões a respeito de uma população com base em informações obtidas de uma amostra dessa população.

O contexto em que se baseia a teoria de teste de hipótese é basicamente o mesmo da teoria de estimação por intervalo de confiança. Temos uma população representada por uma variável aleatória X cuja distribuição de probabilidade depende de algum parâmetro . O interesse agora está em testar a veracidade de alguma afirmativa sobre .

Importação

```
import statsmodels.stats as sm
from statsmodels.stats.weightstats import ztest
import scipy.stats as ss
import pandas as pd
import numpy as np
import warnings
from statsmodels.stats.weightstats import ttest_ind
warnings.filterwarnings('ignore')
```

Hipóteses Nula e Alternativa

A hipótese nula, representada por H_0, é a hipótese básica que queremos testar. Consideraremos apenas hipóteses nulas simples, isto é, hipóteses que estabelecem que o parâmetro de interesse é igual a um determinado valor. A forma geral é:

$H_0 : \theta = \theta_0$ onde $\theta_0 \in \mathbb{R}$.

Alguns exemplos são:

$H_0 : \ = 6$ $H_0 : p = 0,5$ $H_0 : \sigma^2 = 25$

O procedimento de teste de hipótese resultará em uma regra de decisão que nos permitirá rejeitar ou não rejeitar H_0. A hipótese alternativa, representada por H_1, é a hipótese que devemos considerar no caso de rejeição da hipótese nula. A forma mais geral de H_1 é a hipótese bilateral.

$H_1 : \theta \neq \theta_0$

Em algumas situações, podemos ter informação que nos permita restringir o domínio da hipótese alternativa.Temos, então, hipóteses unilaterais à esquerda

$H_1 : \theta < \theta_0$

e hipóteses unilaterais à direita:

$H_1 : \theta > \theta_0$

A escolha entre essas formas de hipótese alternativa se faz com base no conhecimento sobre o problema sendo considerado e deve ser feita antes de se ter o resultado da amostra.

Exemplos

Exemplo 1: Uma revista X afirmou que o tempo de produção de parafusos de uma máquina da empresa Y é em `média de 2 horas`. Deseja-se saber se essa afirmação é verdadeira.

Hipótese Nula: $H_0 : \mu = 2$

Hipótese Alternativa: $H_1 : \mu \neq 2$

Exemplo 2: Uma empresa afirma que seus funcionários da área de TI possuem idade média `maior que 40 anos`. Desaja-se saber se essa afirmação é verdadeira.

Hipótese Nula: $H_0 : \mu = 40$

Hipótese Alternativa: $H_1 : \mu > 40$

OBS: Vale citar que, são mais utilizadas as hipóteses nulas apenas representadas por igualdades. Porém como forma de interpretação, essa hipótese é associada ao inverso da hipótese alternativa. Tendo como exemplo caso anterior, temos a hipótese nula $H_0 : \mu = 40$ que pode ser entendida também como $H_0 : \mu \leq 40$. Pois ao ser realizado um teste para verificar a veracidade da informação, busca-se evidências que a idade média é maior que 40. Caso não há evidências a favor, tem-se que a idade média pode ser menor ou igual a 40.

Estatística de Teste e Regra de Decisão

Assim como na construção dos intervalos de confiança, usaremos uma estatística amostral apropriada para construir o nosso teste de hipótese, e, nesse contexto, essa estatística é chamada estatística de teste. As estatísticas de teste naturalmente dependem do parâmetro envolvido no teste e nesse caso consideraremos os parâmetros média, variância e proporção. O procedimento de decisão será definido em termos da hipótese nula H_0, com duas decisões possíveis: (i) rejeitar H_0 ou (ii) não rejeitar H_0. No quadro a seguir, resumimos as situações possíveis.

		Decisão	
		Rejeitar H_0	Não rejeitar H_0
Possibilidades	H_0 verdadeira	Erro I	OK
	H_0 falsa	OK	Erro II

Vemos, aí, que existem duas possibilidades de erro:

- Erro tipo I: rejeitar H_0 quando H_0 é verdadeira.
- Erro tipo II: não rejeitar H_0 quando H_0 é falsa.

Nível de Significância

Definimos como nível de significância de um teste (geralmente denotado pela letra grega) a probabilidade do erro tipo 1, ou seja,

$\alpha = \text{P(erro I)} = \text{P(rejeitar } H_0 | H_0 \text{ é verdadeiro)}$.

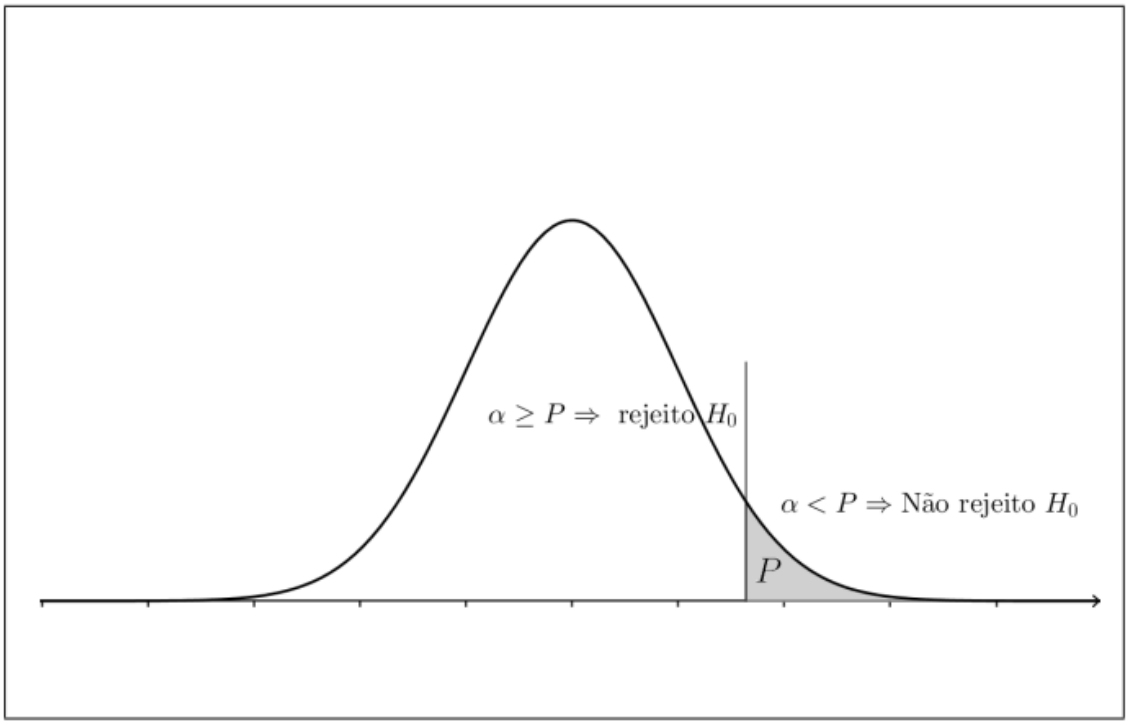
Em geral é escolhido um dos seguintes valores para : 1%, 5% e 10%.

P - Valor

O valor P (p-valor) é a probabilidade de se obter um valor da estatística de teste tão ou mais extremo que o valor observado, supondo-se H0 verdadeira.

A partir da definição, podemos definir a regra de decisão do teste através do resultado do p-valor.

- Se p-valor $> \alpha$, não rejeitamos H_0
- Se p-valor α, rejeitamos H_0, ou seja, há evidências a favor de H_1.



Graças a diversidade de testes disponíveis, esses testes estatísticos podem ser classificados em dois grupos distintos:

- **Testes Paramétricos**: testes realizados partindo do pressuposto que a população segue uma distribuição específica (Ex: Normal).

- **Testes Não Paramétricos**: testes que não precisam partir do pressuposto de uma distribuição específica (Ex: Normal).

A seguir serão apresentados alguns testes estatísticos **paramétricos** para 1 e 2 populações independentes.

Inferência sobre Uma População

Teste de Hipótese para a Média Populacional

Seja $X1, ..., Xn$ uma amostra aleatória de uma população normal com média e variância ^{2} desconhecidas. E suponha que queremos testar as seguintes hipóteses:

$H_0 : \mu = \mu_0$

$H_1 : \mu < ou > ou \neq \mu_0$,

onde μ_0 $\mathbb{R}$.

Seja T a estatística do teste definida por:

$\mathrm{T} = \frac{\bar{X}-\mu}{\frac{S}{\sqrt{n}}} \sim t_{n-1}$

onde $\bar{X}$ e S são a média amostral e a variância amostral, respectivamente.

Após a definição das hipóteses e da estatística de teste, temos que o p-valor é calculado por:

- Se $H_1 : \mu \neq \mu_0$: $p - valor = P(T > T_{obs} \mid H_0)$
- Se $H_1 : \mu > \mu_0$ $p - valor = P(T > T_{obs} \mid H_0)$
- $H_1 : \mu < \mu_0$ $p - valor = P(T < T_{obs} \mid H_0)$

onde

$T_{obs} = \frac{\bar{X}-\mu_0}{\frac{S}{\sqrt{n}}}$

Para se aplicar o teste de média para uma população com variância desconhecida, utilizamos a função `ttest_1samp` da biblioteca **scipy**, cujo argumentos de entrada obrigatórios são:

- `a` : a amostra observada.
- `popmean` : valor associado a μ_0 nas hipóteses.

Nota-se que essa função não possui nenhum argumento de entrada que indique o tipo da hipótese alternativa (unilateral ou bilateral) que será utilizada. Logo, tem-se que essa função retorna apenas o valor do T_{obs} e o p-valor do teste bilateral. Dessa forma, só podem ser aplicados testes cuja hipótese nula seja $\mu \neq \mu_0$.

Como na biblioteca **statsmodels** não há função para a utilização das hipóteses unilaterais, iremos também criar a função `ttest_uni` que retorna o valor de T_{obs} e o p-valor, e possui como argumentos de entrada:

- `amostra`: amostra observada.
- `popmean`: valor associado a μ_0 nas hipóteses.
- `alternative`: tipo da hipótese alternativa desejada: "larger" (unilateral à direita) ou "smaller" (unilateral à esquerda).

```
def ttest_uni(amostra, popmean, alternative):
    n= len(amostra)
    tobs = (np.mean(amostra)-popmean)/np.sqrt(np.var(amostra, ddof=1)/n)
    if(alternative == "smaller"):
        pvalor = ss.t.cdf(tobs,n-1)
    if(alternative == "larger"):
        pvalor = 1 - ss.t.cdf(tobs,n-1)

    print('(%.16f,%.16f)' % (tobs,pvalor))
```

Agora vamos aplicar alguns exemplos práticos de forma a demonstrar a facilidade da aplicabilidade das funções e dos conceitos estatísticos já apresentados.

Para isso será utilizado um banco de dados que representa uma amostra aleatória simples de 1470 funcionários da empresa IBM.

```
base=pd.read_csv('R-Employee-Attrition.csv')
base.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educa
0	41	Yes	Travel_Rarely	1102	Sales	1	2
1	49	No	Travel_Frequently	279	Research & Development	8	1
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2
3	33	No	Travel_Frequently	1392	Research & Development	3	4
4	27	No	Travel_Rarely	591	Research & Development	2	1

Exemplo 1:

Dado que a idade dos funcionários da empresa IBM que raramente viajam a trabalho segue uma distribuição normal com média e variância desconhecidas. Deseja-se testar se a média das idades dos funcionários que **raramente viajam a trabalho** pode ser considerada ou não `igual a 37 anos`, utilizando um nível de significância de `5%`.

Através desse problema temos as hipóteses:

$H_0 : \mu = 37$

$H_1 : \mu \neq 37$

```
base_rarely= base[base['BusinessTravel'] == "Travel_Rarely"] # Filtrar apenas os dados dos
base_rarely.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education
0	41	Yes	Travel_Rarely	1102	Sales	1	2
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2
4	27	No	Travel_Rarely	591	Research & Development	2	1
6	59	No	Travel_Rarely	1324	Research & Development	3	3
7	30	No	Travel_Rarely	1358	Research & Development	24	1

```
ss.ttest_1samp(base_rarely['Age'],popmean = 37)
Ttest_1sampResult(statistic=0.3183482529054914, pvalue=0.7502845793964478)
```

Como p-valor = 0.75 é maior que o nível de significância proposto, não rejeitamos a hipótese nula, ou seja, há evidências que a média das idades dos funcionários que raramente viajam a trabalho é igual a 37 anos.

Exemplo 2

Dado que a idade dos funcionários da empresa IBM que frequentemente viajam a trabalho segue uma distribuição normal com média e variância desconhecidas. Deseja-se testar se a média das idades dos funcionários que **frequentemente viajam a trabalho** pode ser considerada `menor que 39 anos`, utilizando um nível de significância de `1%`.

Através desse problema temos as hipóteses:

$H_0 : \mu = 39$

$H_1 : \mu < 39$

```
base_freq= base[base['BusinessTravel'] == "Travel_Frequently"] # Filtrar apenas os dados d
base_freq.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educ
1	49	No	Travel_Frequently	279	Research & Development	8	1
3	33	No	Travel_Frequently	1392	Research & Development	3	4
5	32	No	Travel_Frequently	1005	Research & Development	2	2
8	38	No	Travel_Frequently	216	Research & Development	23	3
26	32	Yes	Travel_Frequently	1125	Research & Development	16	1

```
ttest_uni(amostra = base_freq['Age'], popmean = 39, alternative = "smaller")
```

```
(-4.9557346279446639,0.0000006294162639)
```

Como p-valor = 0.0000006 é menor que o nível de significância proposto, rejeitamos a hipótese nula, ou seja, há evidências que a média das idades dos funcionários que frequentemente viajam a trabalho é menor do que a 39 anos.

Exemplo 3

Dado que a idade dos funcionários da empresa IBM do departamento de vendas segue uma distribuição normal com média e variância desconhecidas. Deseja-se testar se a média das idades dos funcionários do **departamento de vendas** pode ser considerada `maior do que 40 anos`, utilizando um nível de significância de `10%`.

Através desse problema temos as hipóteses:

$H_0 : \mu = 40$

$H_1 : \mu > 40$

```
base_sales= base[base['Department'] == "Sales"] # Filtrar apenas os dados dos funcinários
base_sales.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educatio
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Scier
18	53	No	Travel_Rarely	1219	Sales	2	4	Life Scier
21	36	Yes	Travel_Rarely	1218	Sales	9	4	Life Scier
27	42	No	Travel_Rarely	691	Sales	8	4	Marketin

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educatio
29	46	No	Travel_Rarely	705	Sales	2	4	Marketin

```
ttest_uni(amostra = base_sales['Age'],popmean = 40, alternative = "larger")
```

```
(-8.0835777128154724,0.9999999999999970)
```

Como o p-valor é maior que o nível de significância proposto, não rejeitamos a hipótese nula, ou seja, não há evidências que a idade média dos funcionários do departamento de vendas seja maior que 40 anos.

Teste de Hipóteses para a Proporção

De posse de uma grande amostra aleatória simples $X1, X2, ..., Xn$ extraída de uma população $X \sim Ber(p)$, nosso interesse está em testar a hipótese nula

$H_0 : p = p_0$

a um nível de significância , onde $p_0 \in (0, 1)$. E p se trata da proporção de elementos de uma população que possuem determinada característica de interesse para o experimento. Dependendo do conhecimento sobre o problema, a hipótese alternativa pode tomar uma das três formas:

$H_1 : p < ou > ou \neq p_0$

Em qualquer dos casos, a estatística de teste baseia-se na proporção amostral; para grandes amostras, sabemos que

$Z = \frac{\hat{p}-p}{\sqrt{\frac{p(1-p)}{n}}} \sim N(0, 1)$

Após as definições das hipóteses e da estatística de teste, temos que o p-valor é calculado por:

- Se $H_1 : p \neq p_0$

```
p-valor = P(Z \geq \left   Z_{obs} \right  \mid H_{0})
```

- Se $H_1 : p > p_0$ $p - valor = P(Z > Z_{obs} \mid H_0)$
- Se $H_1 : p < p_0$ $p - valor = P(Z < Z_{obs} \mid H_0)$

onde

$Z_{obs} = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}$

Para se aplicar o teste para proporção populacional, utilizamos a função `ztest` da biblioteca **statsmodels** que retorna o Z_{obs} e o p-valor, e possui como argumentos de entrada obrigatórios:

- `x1`: a amostra observada.
- `value`: valor associado a p_0
- `alternative`: tipo da hipótese alternativa desejada: "two-sided" (bilateral) , "larger" (unilateral à direita) ou "smaller" (unilateral à esquerda).

Agora vamos aplicar alguns exemplos práticos de forma a demonstrar a aplicabilidade das funções e dos conceitos estatísticos já apresentados.

Utilizaremos o banco de dados dos funcionários da empresa IBM já apresentado anteriormente.

Exemplo 1:

Deseja-se averiguar se é corretar afirmar que `25%` dos funcionários do **departamento de vendas** do IBM passaram por algum problema na empresa, utilizando um nível de significância de `5%`.

Através desse problema temos as hipóteses:

$H_0 : p = 0.25$

$H_1 : p \neq 0.25$

Tem-se essa informação através da variável `Attrition` do banco de dados.

```
base_sales['Attrition'] = base_sales['Attrition'].map({'Yes' : 1, 'No' : 0}) # transforman
base_sales.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educatio
0	41	1	Travel_Rarely	1102	Sales	1	2	Life Scier
18	53	0	Travel_Rarely	1219	Sales	2	4	Life Scier
21	36	1	Travel_Rarely	1218	Sales	9	4	Life Scier
27	42	0	Travel_Rarely	691	Sales	8	4	Marketin
29	46	0	Travel_Rarely	705	Sales	2	4	Marketin

```
ztest(x1 = base_sales['Attrition'], value = 0.25, alternative = "two-sided")
```

```
(-2.279392275404384, 0.0226437570988434)
```

Como p-valor = 0.02 é menor que o nível de significância proposto, rejeitamos a hipótese nula, ou seja, há evidências que a proporção de funcionários que passaram por problemas é diferente de 25%.

Exemplo 2:

Deseja-se averiguar se é corretar afirmar que menos de 30% dos funcionários do **departamento de pesquisa** do IBM passaram por algum problema na empresa, utilizando um nível de significância de 10%.

Através desse problema temos as hipóteses:

$H_0 : p = 0.3$

$H_1 : p < 0.3$

```
base_pesq= base[base['Department'] == "Research & Development"] # Filtrar apenas os dados
base_pesq['Attrition'] = base_pesq['Attrition'].map({'Yes' : 1, 'No' : 0})
base_pesq.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educa
1	49	0	Travel_Frequently	279	Research & Development	8	1
2	37	1	Travel_Rarely	1373	Research & Development	2	2
3	33	0	Travel_Frequently	1392	Research & Development	3	4
4	27	0	Travel_Rarely	591	Research & Development	2	1
5	32	0	Travel_Frequently	1005	Research & Development	2	2

```
ztest(x1 = base_pesq['Attrition'], value = 0.30, alternative = "smaller")
```

```
(-14.499935934146963, 6.063151045772378e-48)
```

Como p-valor é menor do que o nível de significância proposto, rejeitamos H_0, ou seja, há evidências que a proporção de funcionários do departamento de vendas que passaram por algum problema é menos de 30%.

Exemplo 3:

Deseja-se averiguar se é corretar afirmar que mais de 60% dos funcionários do IBM que **frequentemente viajam a trabalho** passaram por algum problema na empresa, utilizando um nível de significância de 1%.

$H_0 : p = 0.6$

$H_1 : p > 0.6$

```
base_freq['Attrition'] = base_freq['Attrition'].map({'Yes' : 1, 'No' : 0})
base_freq.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educ
1	49	0	Travel_Frequently	279	Research & Development	8	1
3	33	0	Travel_Frequently	1392	Research & Development	3	4
5	32	0	Travel_Frequently	1005	Research & Development	2	2
8	38	0	Travel_Frequently	216	Research & Development	23	3
26	32	1	Travel_Frequently	1125	Research & Development	16	1

```
ztest(x1 = base_freq['Attrition'], value = 0.6, alternative = "larger")
```

```
(-13.47921476827307, 1.0)
```

Como p-valor = 1 é maior que o nível de significância proposto, não rejeitamos a hipótese nula, ou seja, não há evidências de que mais de 60% dos funcionários que frequentemente viajam a trabalho tenham tido alguma complicação na empresa.

Teste de Hipóteses para a Variância

Seja $X1, X2, ..., Xn$ uma amostra aleatória de uma população com distribuição normal, com média e variância ^{2} desconhecidas.

E suponha que estamos interessados em testar as hipóteses

$H_o : \sigma^2 = \sigma_0^2$

$H_1 : \sigma^2 < ou > ou \neq \sigma_0^2$,

onde $\sigma_0^2 > 0$.

Em qualquer dos casos, a estatística de teste Q é dada por:

$Q = \frac{(n-1)S^2}{\sigma^2} \sim \chi^2_{(n-1)}$

Após as definições das hipóteses e da estatística de teste, temos que o p-valor é calculado por:

- Se $H_1 : \sigma^2 \neq \sigma_0^2$,

`Missing or unrecognized delimiter for \left`

- Se $H_1 : \sigma^2 > \sigma_0^2$, $p - valor = P(Q > Q_{obs} \mid H_0)$
- Se $H_1 : \sigma^2 < \sigma_0^2$ $p - valor = P(Q < Q_{obs} \mid H_0)$

onde

$Q_{obs} = \frac{(n-1)S^2}{\sigma_0^2}$

Como não há funções pertencentes às bibliotecas **statsmodels** e **scipy** que realizam o teste para a variância, criamos uma função própria chamada `test_var` que retorna o Q_{obs} e o p-valor, e recebe como argumentos de entrada:

- `amostra`: amostra observada
- `sigma2`: valor referente a σ_0^2
- `alternative`: tipo da hipótese alternativa desejada: "two-sided" (bilateral) , "larger" (unilateral à direita) ou "smaller" (unilateral à esquerda).

```
def test_var (amostra, sigma2, alternative= "two.sided"):
    n=len(amostra)
    qobs= (n-1)*np.var(amostra, ddof=1)/sigma2

    if(alternative == "smaller"):
        pvalor = ss.chi2.cdf(qobs,n-1)
    if(alternative == "larger"):
        pvalor = 1 - ss.chi2.cdf(qobs,n-1)
    if(alternative == "two.sided"):
        pvalor = 2*np.minimum(1 - ss.chi2.cdf(qobs,n-1),ss.chi2.cdf(qobs,n-1))

    print('(%.16f,%.16f)' % (qobs,pvalor))
```

Agora vamos aplicar alguns exemplos práticos de forma a demonstrar a facilidade da aplicabilidade das funções e dos conceitos estatísticos já apresentados. Utilizaremos o banco de dados dos funcionários da empresa IBM.

Exemplo 1:

Um dos diretores da empresa IBM afirmou que a variabilidade da idade dentre seus funcionários é de `76 anos`^{2}. Dado que a idade dos funcionários segue uma distribuição normal com média e variância desconhecidas, deseja-se averiguar a veracidade da afirmação, utilizando um nível de significância de `1%`.

Através desse problema temos as hipóteses:

$H_0 : \sigma^2 = 76$

$H_1 : \sigma^2 \neq 76$

Nesse caso não é preciso realizar nenhum processo de filtragem, ou seja, utilzaremos a base original.

```
test_var(amostra = base['Age'], sigma2 = 76, alternative = "two.sided")
```

```
(1613.0982456140357044,0.0096419873719233)
```

Como p-valor = 0.009 é menor que o nível de significância proposto, rejeitamos a hipótese nula, ou seja, temos evidências de que a variabilidade das idades dos funcionários é diferente de 76. Portanto a afirmação é considerada equivocada.

Exemplo 2:

Um jornal local noticiou que a variabilidade dos anos de trabalho dos funcionários da empresa IBM é `inferior a 60 anos`^{2}. Dado que os anos de trabalho dos funcionários da empresa IBM segue uma distribuição normal com média e variância desconhecidas, deseja-se averiguar a veracidade da afirmação, utilizando um nível de significância de `5%`.

Através desse problema temos as hipóteses:

$H_0 : \sigma^2 = 60$

$H_1 : \sigma^2 < 60$

Nesse caso, também não é preciso realizar nenhum processo de filtragem, ou seja, utilizaremos a base original.

```
test_var(base['TotalWorkingYears'],60,"smaller")
```

```
(1482.2347959183696275,0.6009205627661879)
```

Como p-valor = 0.6 é maior que o nível de significância proposto, não rejeitamos a hipótese nula, ou seja, não há evidências que a variabilidade dos anos de trabalho dos funcionários seja inferior a 60. Portanto, considera-se que a informação noticiada é incorreta.

Exemplo 3:

Um dos funcionários da empresa afirmou que a variabilidade dos anos de trabalho dos funcionários da empresa IBM **casados** é `superior a 30 anos`^{2}. Dado que os anos de trabalho dos funcionários casados da empresa IBM segue uma distribuição normal com média e variância desconhecidas, deseja-se averiguar a veracidade da afirmação, utilizando um nível de significância de `10%`.

Através desse problema temos as hipóteses:

$H_0 : \sigma^2 = 30$

$H_1 : \sigma^2 > 30$

Podemos achar essa informação através da variável `MaritalStatus` presente no banco de dados.

```
base_casados = base[base['MaritalStatus'] == "Married"]
base_casados.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educa
1	49	No	Travel_Frequently	279	Research & Development	8	1
3	33	No	Travel_Frequently	1392	Research & Development	3	4
4	27	No	Travel_Rarely	591	Research & Development	2	1
6	59	No	Travel_Rarely	1324	Research & Development	3	3
9	36	No	Travel_Rarely	1299	Research & Development	27	3

```
test_var(amostra = base_casados['TotalWorkingYears'], sigma2 = 50, alternative = "larger")
```

```
(834.6771471025249411,0.0000173254449680)
```

Como p-valor = 0.00001 é menor que o nível de significância proposto, rejeitamos a hipótese nula, ou seja há evidências que a variabilidade dos anos trabalhados entre os casados é superior a 30.Portanto a afirmação realizada é dada como verdadeira.

Inferência sobre Duas Populações

Teste de Hipóteses para Comparação de Duas Proporções

Sejam $X_1 \sim N(\mu_1, \sigma_1^2)$ e $X_2 \sim N(_2, \sigma_2^2)$ duas amostras independentes de tamanho n_1 e n_2, respectivamente, e $\hat{p_1}$ e $\hat{p_2}$ estimadores de p_1 e p_2, proporção de ocorrência de um certo evento. Então, aproximadamente, $\hat{p_1} \sim N\left(p_1, \frac{p_1(1-p_1)}{n_1}\right)$ e $\hat{p_2} \sim N\left(p_2, \frac{p_2(1-p_2)}{n_2}\right)$ implicando $\hat{p_1} - \hat{p_2} \sim N\left(p_1 - p_2, \frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}\right)$. Portanto:

$Z = \frac{(\hat{p_1}-\hat{p_2})-(p_1-p_2)}{\sqrt{\frac{\hat{p_1}(1-\hat{p_1})}{n_1}+\frac{\hat{p_2}(1-\hat{p_2})}{n_2}}} \sim N(0,1)$

Para realizar o teste da comparação de duas proporções, existem 3 formas de propor as hipóteses nula e alternativa:

$H_0 : p_1 - p_2 = p_0 \quad \times \quad H_1 : p_1 - p_2 \neq p_0$

ou

$H_0 : p_1 - p_2 = p_0 \quad \times \quad H_1 : p_1 - p_2 > p_0$

ou

$H_0 : p_1 - p_2 = p_0 \quad \times \quad H_1 : p_1 - p_2 < p_0$

onde $p_0 \in (0, 1)$.

Através da estatística de teste e a definição das hipóteses, para a tomada de decisão é preciso calcular o p-valor, cujas fórmulas são dadas por:

- Se $H_1 : p_X - p_Y \neq p_0$, $p - valor = P(|Z| \geq |Z_{obs}| \mid H_0)$
- Se $H_1 : p_X - p_Y > p_0$, $p - valor = P(Z > Z_{obs} \mid H_0)$
- Se $H_1 : p_X - p_Y < p_0$, $p - valor = P(Z < Z_{obs} \mid H_0)$

onde

$Z_{obs} = \frac{(\hat{p}1-\hat{p}2)-p_0}{\sqrt{\frac{\hat{p}1(1-\hat{p}1)}{n_1}+\frac{\hat{p}2(1-\hat{p}2)}{n_2}}}$

Para realizar esses cálculos no Python, utilizaremos a função `ztest()` da biblioteca **statsmodels**, já apresentado anteriormente, com os seguintes argumentos de entrada:

- `x1`: a amostra observada.
- `x2`: a segunda amostra observada.
- `value`: valor associado a p_0
- `alternative`: tipo da hipótese alternativa desejada: "two-sided" (bilateral) , "larger" (unilateral à direita) ou "smaller" (unilateral à esquerda).

Agora vamos aplicar alguns exemplos práticos de forma a demonstrar a aplicabilidade das funções e dos conceitos estatísticos já apresentados. Utilizaremos um banco de dados que representa uma amostra aleatória simples de 562 filmes lançados no cinema no período de 2007 a 2011, com informações como gênero, orçamento e críticas presentes no site Rotten Tomatoes.

```
df = pd.read_csv('Movie-Ratings.csv')
df.head()
```

	Film	Genre	Rotten Tomatoes Ratings %	Audience Ratings %	Budget (million
0	(500) Days of Summer	Comedy	87	81	8
1	10,000 B.C.	Adventure	9	44	105
2	12 Rounds	Action	30	52	20
3	127 Hours	Adventure	93	84	18
4	17 Again	Comedy	55	70	20

Exemplo 1

Deseja-se avaliar se no período de 2007 a 2011, a proporção de **filmes de Comédia** lançados no cinema foi `igual` a proporção de **filmes de Aventura**, utilizando um nível de significância de `5%`.

As hipóteses serão as seguintes:

$H_0 : p_X - p_Y = 0$

$H_1 : p_X - p_Y \neq 0$

Onde X ão filmes do gênero `Comédia` e Y, os filmes do gênero `Aventura`.

```
df['Genre_Comedy'] = np.where(df.Genre == "Comedy", 1, 0)
df['Genre_Adventure'] = np.where(df.Genre == "Adventure", 1, 0)
df.head()
```

	Film	Genre	Rotten Tomatoes Ratings %	Audience Ratings %	Budget (million
0	(500) Days of Summer	Comedy	87	81	8
1	10,000 B.C.	Adventure	9	44	105
2	12 Rounds	Action	30	52	20
3	127 Hours	Adventure	93	84	18
4	17 Again	Comedy	55	70	20

```
ztest(x1=df.Genre_Comedy, x2= df.Genre_Adventure, value = 0, alternative = 'two-sided')
```

```
(11.789438693301005, 4.42480180658489e-32)
```

Como o p-valor é menor que o nível de significância proposto, rejeitamos H_0, ou seja, há evidências que as proporções de filmes de Comédia e de Aventura lançados no período de 2007 a 2011 nos cinema sejam diferentes.

Exemplo 2

Baseado no resultado do exemplo anterior, sabe-se que as proporções de filmes de comédia e aventura em 2007-2011 são consideradas diferentes. Então, através disso, deseja-se avaliar se a proporção de **filmes de Comédia** é `maior` que a proporção de **filmes de Aventura** lançados nos cinemas em 2007-2011, utilizando um nível de significância de `1%`.

As hipóteses serão as seguintes:

$H_0 : p_X - p_Y = 0$

$H_1 : p_X - p_Y > 0 \Rightarrow H_1 : p_X > p_Y$

Onde X são filmes do gênero `Comédia` e Y, os filmes do gênero `Aventura`.

```
ztest(x1=df.Genre_Comedy, x2=df.Genre_Adventure, value = 0, alternative = 'larger')
```

```
(11.789438693301005, 2.212400903292445e-32)
```

Como p-valor é menor que o nível de significância proposto, rejeitamos H_0, ou seja, há evidências que foram lançados mais filmes do gênero Comédia do que filmes do gênero Aventura nos cinemas no período de 2007 a 2011.

Teste de Hipótese para Razão das Variâncias

Sejam $X_1 \sim N(_1, \sigma^2 1)$ e $X_2 \sim N(2, \sigma^2 2)$ duas amostras independentes de tamanho $n1 n_2$ e $S^2 1 S^2 2$ suas variâncias amostrais. Então,

$F = \frac{\frac{S^2 1}{S^2 2}}{\frac{\sigma^2 1}{\sigma^2 2}} \sim F_{n_1 - 1, n_2 - 1}$

Para realizar o teste para razão de duas variâncias, existem 3 formas de propor as hipóteses nula e alternativa:

- $H_0 : \frac{\sigma_1^2}{\sigma_2^2} = \sigma_0^2 \quad \times \quad H_1 : \frac{\sigma_1^2}{\sigma_2^2} \neq \sigma_0^2$
- $H_0 : \frac{\sigma_1^2}{\sigma_2^2} = \sigma_0^2 \quad \times \quad H_1 : \frac{\sigma_1^2}{\sigma_2^2} < \sigma_0^2$
- $H_0 : \frac{\sigma_1^2}{\sigma_2^2} = \sigma_0^2 \quad \times \quad H_1 : \frac{\sigma_1^2}{\sigma_2^2} > \sigma_0^2$

Através da estatística de teste e a definição das hipóteses, para a tomada de decisão é preciso calcular o p-valor, cujas fórmulas são dadas por:

- Se `$H_{1} : \frac{\sigma^2{X}}{\sigma^2{Y}}\neq \sigma^2{0}p-valor = 2\times min\left { P\left ( F \geq F{obs} \mid H_{0}\right ), P\left ( F \leq F_{obs} \mid H_{0}\right ) \right }$`
- Se $H_1 : \frac{\sigma^2 X}{\sigma^2 Y} < \sigma^2 0 p - valor = P\left(F < Fobs \mid H_0\right)$
- Se $H_1 : \frac{\sigma^2 X}{\sigma^2 Y} > \sigma^2 0 p - valor = P\left(F > Fobs \mid H_0\right)$

onde

$F_{obs} = \frac{\frac{S^2 1}{S^2 2}}{\frac{\sigma^2 1}{\sigma^2 2}}$

Infelizmente, assim como no caso de uma população, o Python não possui função para avaliar a razão das variâncias. Por isso, construímos a nossa própria função, cujos argumentos de entrada são:

- `x1`: primeira amostra;
- `x2`: segunda amostra;
- `value`: valor de σ_0^2;
- `alternative`: 'two-sided' (default): H1: razão das variâncias é diferente que `value`; 'larger': H1: razão das variâncias é maior que `value`; 'smaller': H1: razão das variâncias é menor que `value`.

Essa função retorna a estatística de teste F e o p-valor calculado.

```
# razão das variâncias

def rvar_test(x1, x2, value, alternative):
    stattest = (np.var(x1)/np.var(x2))/value

    if alternative == 'smaller':
        pvalor = ss.f.cdf(stattest, len(x1) - 1, len(x2) - 1)
    elif alternative == 'larger':
        pvalor = 1 - ss.f.cdf(stattest, len(x1) - 1, len(x2) - 1)
    elif alternative == 'two-sided':
        pvalor = 2*ss.f.cdf(stattest, len(x1) - 1, len(x2) - 1)
        if pvalor >= 1:
            pvalor = 1

    print('(%.16f,%.16f)' % (stattest,pvalor))
```

Agora vamos aplicar alguns exemplos práticos de forma a demonstrar a facilidade da aplicabilidade da função e dos conceitos estatísticos já apresentados. Utilizaremos o banco de dados de avaliações de filmes.

Exemplo

Dado que os orçamentos dos filmes de Ação e Aventura seguem distribuições normais com médias e variâncias desconhecidas. Deseja-se avaliar, ao nível de significância de `5%`, se a razão das variâncias dos orçamentos dos filmes de **Ação e Aventura** é igual a `1`. Ou seja, verificar se as variâncias são iguais.

As hipótetes serão as seguintes:

$H_0 : \frac{\sigma^2 X}{\sigma^2 Y} = 1$

$H_1 : \frac{\sigma^2 X}{\sigma^2 Y} \neq 1$

Onde X são os orçamentos dos filmes de `Ação` e Y, os orçamentos dos filmes de `Aventura`.

Teste de Hipótese para Comparação de Médias para Duas Populações Independentes

Sejam $X \sim N(_X, \sigma^2 X)$ e $Y \sim N(Y, \sigma^2_Y)$ duas amostras independentes com variâncias desconhecidas. Se, ao relizarmos o teste da razão das variâncias, constatarmos que as variâncias são iguais, teremos o seguinte caso:

$$T = \frac{(\bar{X}-\bar{Y})-\mu}{\sqrt{S^2comb\left(\frac{1}{nX}+\frac{1}{n_Y}\right)}} \sim t_{n_X+n_Y-2}$$

Onde,

$$S^2comb = \frac{(nX-1)S^2X+(nY-1)S^2Y}{nX+n_Y-2}$$

Se, ao relizarmos o teste da razão das variâncias, constatarmos que as variâncias são diferentes, teremos o seguinte caso:

$$T = \frac{(\bar{X}-\bar{Y})-\mu}{\sqrt{\frac{S^2X}{nX}+\frac{S^2Y}{nY}}} \sim t_v$$

Onde,

$$v = \frac{\left(\frac{s^2X}{nX}+\frac{s^2Y}{nY}\right)^2}{\frac{1}{n_X-1}\left(\frac{s^2X}{nX}\right)^2+\frac{1}{n_Y-1}\left(\frac{s^2Y}{nY}\right)^2}$$

Para realizar o teste da comparação de duas médias, existem 3 formas de propor as hipóteses nula e alternativa:

- $H_0 : \mu_X - \mu_Y = \mu_0 \quad \times \quad H_1 : \mu_X - \mu_Y \neq \mu_0$
- $H_0 : \mu_1 - \mu_2 = \mu_0 \quad \times \quad H_1 : \mu_1 - \mu_2 > \mu_0$
- $H_0 : \mu_1 - \mu_2 = \mu_0 \quad \times \quad H_1 : \mu_1 - \mu_2 < \mu_0$

Dada as estatísticas de teste e a definição das hipóteses, é preciso calcular o p-valor para a tomada de decisão, cujas fórmulas são dadas por:

- Se $H_1 : \mu_X - \mu_Y \neq \mu_0$, `$p-valor = P\left ( T > \left T_{obs} \right \mid H_{0} \right)$`
- Se $H_1 : \mu_X - \mu_Y > \mu_0$, $p - valor = P\left(T > T_{obs} \mid H_0\right)$
- Se $H_1 : \mu_X - \mu_Y < \mu_0$, $p - valor = P\left(T < T_{obs} \mid H_{obs}\right)$

onde

$$T_{obs} = \frac{(\bar{X}-\bar{Y})-\mu_0}{\sqrt{S^2comb\left(\frac{1}{nX}+\frac{1}{n_Y}\right)}}$$

quando as variâncias são iguais, ou

$$T_{obs} = \frac{(\bar{X}-\bar{Y})-\mu_0}{\sqrt{\frac{S^2X}{nX}+\frac{S^2Y}{nY}}}$$

quando as variâncias são diferentes.

Para realizar esses cálculos no Python, utilizaremos a função `ttest_ind()` da biblioteca **statsmodels**, cujos argumentos de entrada são:

- `x1`: amostra X.
- `x2`: amostra Y.

- `value`: valor de μ_0.
- `alternative`: 'two-sided' (default): H1: diferença entre as médias é diferente que `value`; 'larger': H1: diferença entre as médias é maior que `value`; 'smaller': H1: diferença entre as médias é menor que `value`.
- `usevar`: 'pooled' se as variâncias são iguais, 'unequal' se as variâncias são diferentes.

Essa função retornará a estatística de teste, o pvalor e os graus de liberdade, respectivamente.

Agora vamos aplicar alguns exemplos práticos de forma a demonstrar a facilidade da aplicabilidade da função e dos conceitos estatísticos já apresentados. Utilizaremos o banco de dados de avaliações de filmes.

Exemplo 1

Queremos avaliar se a média dos orçamentos dos filmes de **Ação e Aventura** são `iguais`, ao nível de significância de `5%`:

As hipóteses serão as seguintes:

$H_0 : \mu_X - \mu_Y = 0$

$H_1 : \mu_X - \mu_Y \neq 0$

Onde X são os orçamentos dos filmes de `Ação` e Y, os orçamentos dos filmes de `Aventura`.

Primeiro iremos aplicar o teste de razão de variâncias `rvar_test` para testar a igualdade entre elas. Utilize também o nível de significância de `5%`

```
rvar_test(x1 = df_acao["Budget (million $)"],x2 = df_adventure["Budget (million $)"],value
```

```
(1.1885458719831832,1.0000000000000000)
```

Como p-valor=1 sendo maior que o nível de significância proposto, há evidências de que as varianças são **iguais**. Agora aplicaremos o teste de comparação de médias considerando as variâncias iguais.

```
ttest_ind(x1 = df_acao["Budget (million $)"],x2 = df_adventure["Budget (million $)"], alte
```

```
(0.24627191134923368, 0.8057508151994627, 181.0)
```

Como p-valor= 0.80 é maior que no nível de significância proposto, não rejeitamos a hipótese nula, ou seja, há evidências de que os orçamentos dos filmes de Ação e Aventura são iguais.

Exemplo 2

A revista "The Hollywood Reporter" afirmou que a média dos orçamentos dos filmes de **Ação** é `superior` à média dos orçamentos dos filmes de **Comédia**. Deseja-se avaliar a veracidade dessa informação, ao nível de significância de `5%`.

As hipóteses serão as seguintes:

$H_0 : \mu_X - \mu_Y = 0$

$H_1 : \mu_X - \mu_Y \ > 0 \Rightarrow H_1 : \mu_X > \mu_Y$

Onde X são os orçamentos dos filmes de `Ação` e Y, os orçamentos dos filmes de `Comédia`.

```
df_comedia = df[df["Genre"] == 'Comedy']
df_comedia.head()
```

	Film	Genre	Rotten Tomatoes Ratings %	Audience Ratings %	Budget (million $
0	(500) Days of Summer	Comedy	87	81	8
4	17 Again	Comedy	55	70	20
6	27 Dresses	Comedy	40	71	30
8	30 Minutes or Less	Comedy	43	48	28
9	50/50	Comedy	93	93	8

Primeiro iremos aplicar o teste de razão de variâncias `rvar_test` para testar a igualdade entre elas. Utilize o nível de significância de `5%`.

```
rvar_test(x1 = df_acao["Budget (million $)"], x2= df_comedia["Budget (million $)"],value=1
```

```
(5.6148884065672879,0.0000000000000001)
```

Como o p-valor é menor que o nível de significância proposto, temos que as variâncias são diferentes. Agora realizaremos o teste de comparação de médias considerando variâncias diferentes.

```
ttest_ind(x1 = df_acao["Budget (million $)"],x2 = df_comedia["Budget (million $)"], altern
```

```
(8.765572657433554, 7.820570304870018e-16, 201.07797410706948)
```

Como p-valor é menor que o nível de significância proposto, rejeitamos a hipótese nula, ou seja, há evidências de que a média dos orçamentos dos filmes de Ação é superior à média dos orçamentos dos filmes de Comédia.

Teste de Hipótese para Comparação de Médias para Duas Populações Dependentes

Sejam $X \sim N(_X, \sigma^2 X)$ e $Y \sim N(Y, \sigma^2 Y)$ duas amostras pareadas tal que consideramos os pares $(X1, Y_1)$, ..., (X_n, Y_n). Definindo $D = X - Y$, temos que $D \sim N(_D, \sigma^2 D)$, com $d = {}_X - {}_Y$. Assim,

$T = \frac{\bar{D} - \mu}{\sqrt{\frac{S^2 D}{n}}} \sim tn - 1$

onde,

$D_i = X_i - Y_i$ e $\bar{D} = \sum_{i=1}^{n} \frac{D_i}{n}$

Assim como nos outros casos, existirão 3 hipóteses possíveis, e suas formas de calcular o pvalor:

- $H_0 : \mu_D = \mu_0 \quad \times \quad H_1 : \mu_D \neq \mu_0$
- $H_0 : \mu_D = \mu_0 \quad \times \quad H_1 : \mu_D > \mu_0$
- $H_0 : \mu_D = \mu_0 \quad \times \quad H_1 : \mu_D < \mu_0$

Se $H_1 : \mu_D \neq \mu_0$,

```
$p-valor = P (\left    T \right    >\left  T_{obs} \right  \mid H_{0})$
```

Se $H_1 : \mu_D > \mu_0$, $p - valor = P(T > T_{obs} \mid H_0)$

Se $H_1 : \mu_D < \mu_0$, $p - valor = P(T < T_{obs} \mid H_0)$

onde

$T_{obs} = \frac{\bar{D} - \mu_0}{\sqrt{\frac{S_D^2}{n}}}$

Definindo $D = X - Y$, temos o caso de Teste de Hipótese para Média de uma população. Portanto, após obtermos D, podemos aplicar as funções `ttest_1samp` e `ttest_uni` já apresentadas no material.

Para aplicações, utilizaremos o banco de dados que representa uma amostra aleatória simples de 198 pessoas. Essa base representa um experimento sobre os efeitos do medicamento ansiolíticos na utilização da memória.

```
data = pd.read_csv("Islander_data.csv")
data.head()
```

	first_name	last_name	age	Happy_Sad_group	Dosage	Drug	Mem_Score_Before	Mem_Score_
0	Bastian	Carrasco	25	H	1	A	63.5	61.2
1	Evan	Carrasco	52	S	1	A	41.6	40.7
2	Florencia	Carrasco	29	H	1	A	59.7	55.1
3	Holly	Carrasco	50	S	1	A	51.7	51.2
4	Justin	Carrasco	52	H	1	A	47.0	47.1

Exemplo 1

Um experimento foi realizado com moradores de diversas ilhas, onde foi possível testar os efeitos de remédios ansiolíticos no uso da memória. Sabendo que os tempos dos testes de memória, antes e depois, dos moradores expostos ao medicamento `A` seguem uma distribuição normal com média e variância desconhecidas, deseja-se avaliar se a droga `A` afeta o uso da memória, seja positivamente ou negativamente, observando a diferença do tempo do teste de memória antes e depois da exposição ao medicamento, ao nível de significância de `5%`.

As hipótetes serão as seguintes:

$H_0 : \mu_X - \mu_Y = 0 \Rightarrow H_0 : \mu_D = 0$

$H_1 : \mu_X - \mu_Y \neq 0 \Rightarrow H_1 : \mu_D \neq 0$

Onde D é a diferença entre o tempo de resposta do teste depois (X) e antes (Y) do uso da droga `A`.

```
data_A = data[data['Drug'] == 'A']
data_A.head()
```

	first_name	last_name	age	Happy_Sad_group	Dosage	Drug	Mem_Score_Before	Mem_Score_
0	Bastian	Carrasco	25	H	1	A	63.5	61.2
1	Evan	Carrasco	52	S	1	A	41.6	40.7
2	Florencia	Carrasco	29	H	1	A	59.7	55.1
3	Holly	Carrasco	50	S	1	A	51.7	51.2
4	Justin	Carrasco	52	H	1	A	47.0	47.1

Utilizaremos a variável `Diff` que já contém os valores das diferenças dos tempos de resposta dos testes de memórias, sendo calculado da forma: `Diff = Mem_Score_After - Mem_Score_Before`.

```
ss.ttest_1samp(a = data_A['Diff'], popmean =  0)
Ttest_1sampResult(statistic=5.848567939730841, pvalue=1.6898179935298675e-07)
```

Como p-valor é menor que no nível de significância proposto, rejeitamos a hipótese nula, ou seja, há evidências de que o medicamento afeta, seja positivamente ou negativamente, o uso da memória.

Exemplo 2

Dado que no exemplo anterior as médias dos tempos foram considerados diferentes, iremos avaliar se a droga **A** afeta o uso da memória `positivamente`, ao nível de significância de `1%`.

As hipótetes serão as seguintes:

$H_0 : \mu_X - \mu_Y = 0 \Rightarrow H_0 : \mu_D = 0$

$H_1 : \mu_X - \mu_Y > 0 \Rightarrow H_1 : \mu_D > 0$

Onde D é a diferença entre o tempo de resposta do teste depois (X) e antes (Y) do uso da droga `A`.

```
ttest_uni(amostra = data_A['Diff'],popmean = 0, alternative = "larger")
```

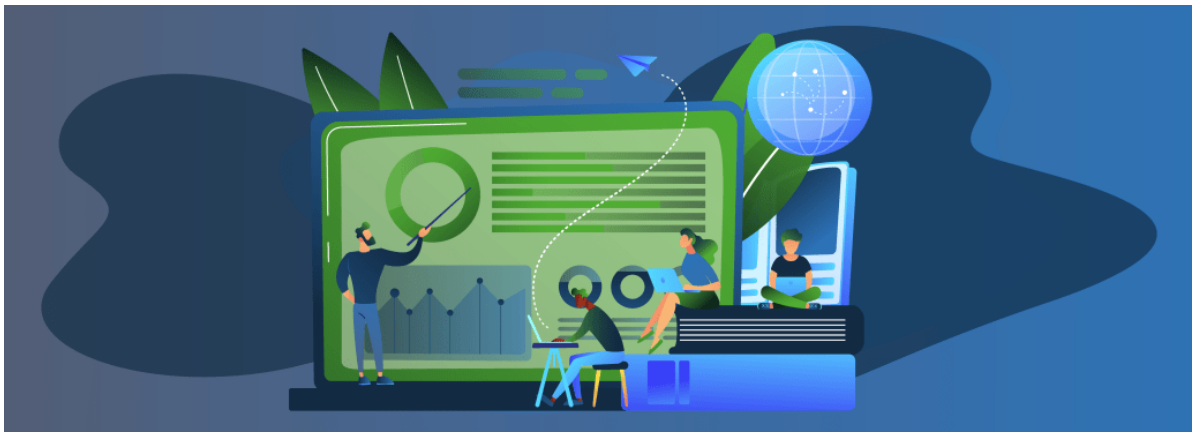
```
(5.8485679397308408, 0.0000000844908996)
```

Como p-valor é menor que o nível de significância proposto, rejeitamos a hipótese nula, ou seja, há evidências de que o tempo de resposta do teste depois do uso do medicamento foi maior do que antes. Portanto, tem-se que o medicamento A afeta positivamente o uso da memória.

Referências

FARIAS, A. M. L. **Apostila de Estatística II**. Departamento de Estatística. 2017. Universidade Federal Fluminense. Velarde, L. G. C. CAVALIERE, Y. F. **Apostila Inferência Estatística**. Departamento de Estatística. Universidade Federal Fluminense

Teste de Hipóteses - Parte 2



Importação

```
import statsmodels.stats as sm
from statsmodels.stats.weightstats import ztest
import scipy.stats as ss
import pandas as pd
import numpy as np
import warnings
from statsmodels.sandbox.stats.multicomp import tukeyhsd
from statsmodels.stats.weightstats import ttest_ind
warnings.filterwarnings('ignore')
from scipy.stats import chisquare
from scipy.stats import chi2_contingency
```

Inferência sobre Três ou mais Populações

Teste de Levene

Teste de Levene é usado para testar se k amostras foram coletadas de populações que possuem mesma variância. As hipóteses do teste são:

$H_0 : \sigma_1^2 = \sigma_2^2 = \cdots = \sigma_k^2$

$H_1 : \exists\, i, j\ , \text{tal que}\, \sigma_i^2 \neq \sigma_j^2$

Ou seja, igualdade entre as variâncias populacionais contra a alternativa de que nem todas as variâncias são iguais. Ele é o teste mais robusto contra falta de normalidade dos dados e sua estatística de teste segue com uma distribuição F sob H_0:

$L = \frac{(N-k)}{(k-1)} \cdot \frac{\sum_{i=1}^{k} n_i(\bar{Z}_i - \bar{Z})^2}{\sum_{i=1}^{k}\sum_{j=1}^{n_i}(Z_{ij} - \bar{Z}_i)^2}$

em que

$Z_{ij} = |X_{ij} - \text{Medida Central do Grupo } i|$

é o desvio absoluto dos X_{ij} em relação à média de cada grupo. Já

$\bar{Z}_i = \frac{\sum_{j=1}^{n_i} Z_{ij}}{n_i}$

é a média dos $Z_{i,j}$ no grupo i. E

$\bar{Z} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n_i} Z_{ij}}{N}$

é a média geral dos $Z_{i,j}$

O p-valor é calculado da forma:

p-valor $= P(F_{(k-1,N-k)} \geq W)$

Para aplicarmos o teste de Levene, utilizamos a função `levene` da biblioteca Scipy, cujos argumentos de entrada são:

- `sample1`, `sample2`, … : amostras observadas.
- `center`: há três opções: "mean", "median" ou "trimmed". Para o caso da população seguir uma normal, utilize "mean".

Para mais informações clique aqui para acessar a documentação da função.

Como a maioria das funções, essa retorna o valor da estatística de teste e o p-valor.

Para aplicação do teste utilizaremos o banco de dados de uma amostra aleatória simples de avaliações de filmes do site Rotten Tomatoes. Vamos supor que os dados são provenientes de distribuições normais.

```
df = pd.read_csv('Movie-Ratings.csv')
df.head()
```

	Film	Genre	Rotten Tomatoes Ratings %	Audience Ratings %	Budget (million
0	(500) Days of Summer	Comedy	87	81	8
1	10,000 B.C.	Adventure	9	44	105
2	12 Rounds	Action	30	52	20
3	127 Hours	Adventure	93	84	18
4	17 Again	Comedy	55	70	20

Exemplo

Deseja-se saber se a variabilidade das notas dos críticos do site Rotten Tomatoes são `iguais` nos filmes dos gêneros **Comédia, Ação, Aventura e Drama**, ao nível de significância de `5%`.

$H_0 : \sigma^2_{\text{Comédia}} = \sigma^2_{\text{Ação}} = \sigma^2_{\text{Aventura}} = \sigma^2_{\text{Drama}}$

$H_1 : \exists\, i, j$, tais que , $\sigma_i^2 \neq \sigma_j^2$, com $i, j \in \{\text{Comédia, Ação, Aventura, Drama}\}$

```
df_acao, df_adventure, df_comedia, df_drama = df[df["Genre"] == 'Action'], df[df["Genre"]
```

```
LeveneResult(statistic=1.6923208903592077, pvalue=0.16786851919967521)
```

Como p-valor=0.16 é maior que o nível de significância proposto, não rejeitamos a hipótese nula, ou seja, há evidências de que as variâncias das avaliações dos filmes dos gêneros Comédia, Ação, Drama e Aventura são iguais.

Teste One-Way ANOVA

A One-Way ANOVA é usada para testar diferenças entre médias de pelo menos três populações, uma vez que a comparação entre dois grupos pode ser obtida através do teste t. Assim como no caso do teste t, o modelo da ANOVA exige que as populações possuam distribuição Normal e, além disso, as variâncias devem ser **iguais**. Considere que são extraídas amostras aleatórias simples de tamanhos n_1, n_2,
$ldots, nk$ dessas populações. A hipótese de interesse é

$H_0 : \mu_1 = \mu_2 = \cdots = \mu_k$

$H_1 : \exists\, i, j \,\text{tais que}\, \mu_i \neq \mu_j$, com $i, j \in \{1, 2, \ldots, k\}$

A estatística de teste é dada por:

$F = \frac{\text{QM\_entre}}{\text{QM\_dentro}}$

onde

$\text{QM\_entre} = \frac{\text{SQ\_entre}}{k-1}$

e

$\text{QM\_dentro} = \frac{\text{SQ\_dentro}}{N-k}$

O p-valor é calculado da forma:

$\text{p-valor} = P(F_{(k-1, N-k)} \geq F)$

Para aplicarmos o teste One Way ANOVA utilizamos a função `f_oneway` da biblioteca **Scipy**, cujos argumentos de entrada são apenas:

- `sample1`, `sample2`, … : amostras observadas.

Para mais informações clique aqui para acessar a documentação da função.

Vamos exemplificar a utilização do teste com o banco de dados de avaliações de filmes.

Exemplo

Deseja-se verificar se as médias dos avaliações do site Rotten Tomatoes são `iguais` para os filmes de **Ação, Comédia, Drama e Aventura**, utilizando um nível de significância de `5%`.

$H_0 : \mu_{\text{Ação}} = \mu_{\text{Comédia}} = \mu_{\text{Drama}} = \mu_{\text{Aventura}}$

$H_1 : \exists\, i, j \,\text{tais que}\, \mu_i \neq \mu_j$, com $i, j \in \{\text{Ação, Comédia, Drama, Aventura}\}$

Primeiro passo a ser realizado é verificar se as variâncias são iguais, aplicando o Teste de Levene. Logo, através da aplicação no exemplo anterior, temos que as variâncias são consideradas **iguais**. Agora realizaremos o teste de ANOVA.

```
ss.f_oneway(df_acao['Rotten Tomatoes Ratings %'],df_adventure['Rotten Tomatoes Ratings %']
```

```
F_onewayResult(statistic=6.039948218319176, pvalue=0.0004869684243719264)
```

Como p-valor é menor que o nível de significância proposto, rejeitamos a hipótese nula, ou seja, pelo menos uma das médias é considerada diferente. Para saber quais médias são diferentes, realize um teste de comparações múltiplas.

Teste de Shapiro-Wilk

Utiliza-se o teste de Shapiro-Wilk para verificar se uma amostra é proveniente de uma população com distribuição normal. O teste avalia (ou verifica) as seguintes hipóteses:

H_0 : Os dados são provenientes de uma distribuição Normal

H_1 : os dados não são provenientes de uma distribuição Normal.

Para realizar o teste de Shapiro-Wilk utilizamos a função `shapiro` da biblioteca Scipy, cujo único argumento de entrada é:

- x : a amostra observada.

Para mais informações clique aqui para acessar a documentação da função.

Exemplo

Deseja-se saber se as avaliações da audiência constada no site Rotten Tomatoes para os filmes de **Aventura** são provenientes de uma `distribuição normal`, utilizando um nível de significância de `5%`.

```
ss.shapiro(df_adventure['Audience Ratings %'])
```

```
(0.9456037282943726, 0.14058126509189606)
```

Como p-valor=0.14 é maior que o nível de significância proposto, não rejeitamos a hipótese nula, ou seja, podemos afirmar que a amostra é proveniente de uma população com distribuição normal.

Teste de Kolmogorov–Smirnov

O teste Kolmogorov–Smirnov é um teste não paramétrico sobre a igualdade de distribuições de probabilidade contínuas e unidimensionais que pode ser usado para comparar uma amostra com uma distribuição de probabilidade de referência ou duas amostras uma com a outra. Dado isso, podemos ter as seguintes hipóteses:

H_0 : os dados seguem uma certa distribuição

H_1 : os dados não seguem uma certa distribuição

Ou

H_0 : as duas amostras seguem a mesma distribuição

H_1 : as duas amostram não seguem a mesma distribuição

Para realizar o teste de Kolmogorov-Smirnov para uma amostra utilizamos a função `kstest` da biblioteca **Scipy**, cujos argumentos de entrada são:

- `rvs`: a amostra observada.
- `cdf`: o nome da distribuição: "norm","expon", ...
- `args`: parâmetros da distribuição.

Para mais informações sobre essa função clique aqui.

Para realizar o teste de Kolmogorov-Smirnov para duas amostras utilizamos a função `ks_2sample` da biblioteca Scipy, cujos argumentos principais são:

- data1: primeira amostra observada.
- data2: segunda amostra observada.

E sobre essa função, clique aqui.

Agora vamos aplicá-las em alguns exemplos.

Exemplo 1

Deseja-se verificar se as avaliações da audiência constada no site Rotten Tomatoes para os filmes de Terror são provenientes de uma distribuição normal, utilizando um nível de significância de 5%.

H_0: os dados seguem uma distribuição normal

H_1: os dados não seguem uma distribuição normal

```
df_horror = df[df["Genre"] == 'Horror']
df_horror.head()
```

	Film	Genre	Rotten Tomatoes Ratings %	Audience Ratings %	Budget (m
7	30 Days of Night	Horror	50	57	32
12	A Nightmare on Elm Street	Horror	13	40	35
20	Alien vs. Predator -- Requiem	Horror	14	37	40
28	Apollo 18	Horror	23	31	5
59	Case 39	Horror	23	42	26

```
ss.kstest(df_horror['Audience Ratings %'], "norm", args=[np.mean(df_horror['Audience Ratin
```

```
KstestResult(statistic=0.05351444278807435, pvalue=0.9989829445848811)
```

Como p-valor = 0.99 é maior que o nível de significância proposto, não rejeitamos a hipótese nula, ou seja, há evidências de que os dados são provenientes de uma normal.

Exemplo 2

Deseja-se verificar se as avaliações da audiência constadas no site Rotten Tomatoes para os filmes de **Terror** possuem a **mesma distribuição** do que as avaliações da audiência para os filmes de **Ação**, utilizando um nível de significância de `5%`.

H_0: os dados seguem a mesma distribuição.

H_1: os dados não seguem a mesma distribuição.

Como p-valor=0.004 é menor que o nível de significância proposto, rejeitamos a hipótese nula, ou seja, as avaliações da audiência para os filmes de Terror e Ação não seguem a mesma distribuição.

Análise sobre Dados Categóricos

Teste de Aderência

O teste de Aderência busca avaliar o comportamento, em uma população, de uma variável categórica que pode assumir k valores. Sejam $p_1, p_2, p_3, ..., p_k$, respectivamente, as proporções populacionais das categorias $1, 2, 3, ..., K$, teremos as seguintes hipóteses:

$H_0 : p_1 = p_{1,A}$, $p_2 = p_{2,A}$, $p_3 = p_{3,A}, \dots, p_k = p_{k,A}$

$H_1 : \exists\, i$, tal que $p_i \neq p_{i,A}$, com $i \in \{1, 2, 3, \dots, k\}$

Sob H_0:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

Onde:

- O_i: é o número de vezes que a categoria i é observada na amostra
- E_i: valor esperado; obtido pela multiplicação do tamanho da amostra com p_i

E o p-valor é calculado da seguinte forma:

Para o teste de Aderência usaremos a função `chisquare` da biblioteca **Scipy**, cujos argumentos de entrada são

- `f_obs`: amostra observada
- `f_exp`: valores esperados, sob H_0

Para mais informações sobre a função clique aqui para acessar sua documentação.

Exemplo

Deseja-se verificar se a proporção de filmes de **Aventura**, **Romance** e **Suspense** avaliados são, respectivamente, `0.26`, `0.29`, `0.45`, ao nível de significância de `5%`.

$H_0 : p_1 = 0.26$
,
$p_2 = 0.29$
,
$p_3 = 0.45$
H
$_1 : p$
$_i \neq p$
$_{i,A}$,
, para pelo menos um i
,
$i = 1, 2, 3$

Obtendo os valores observados:

```
tabela_freq = df.Genre.value_counts()
tabela_freq
```

```
Comedy 172 Action 154 Drama 101 Horror 49 Thriller 36 Adventure 29 Romance 21
Name: Genre, dtype: int64
```

Aplicando a função:

```
obs = [tabela_freq[5], tabela_freq[6], tabela_freq[4]]
total_obs = sum(obs)

chisquare(f_obs = obs, f_exp = [0.26*total_obs, 0.29*total_obs, 0.45*total_obs])
```

```
Power_divergenceResult(statistic=2.782616741718586, pvalue=0.24874963485619836)
```

Como p-valor=0.25 é maior que o nível de significância proposto, não rejeitamos a hipótese nula, ou seja, podemos afirmar que a proporção de filmes de Aventura, Romance e Suspense avaliados são respectivamente de 0.26,0.29 e 0.45.

Teste de Homogeneidade

Esse caso é similar ao teste de Aderência, com a diferença de que agora será avaliado o comportamento, em C populações **independentes**, de uma variável categórica que pode assumir L valores.

Para este teste, teremos as seguintes hipóteses:

$H_0 : p_{1,1} = p_{1,2} = ... = p_{1,C}$
;
$p_{2,1} = p_{2,2} = ... = p_{2,C}$
;

...

;
$p_{L,1} = p_{L,2} = ... = p_{L,C}$

$H_1 : \exists\, i, j$, tais que $p_{i,j} \neq p_{i,A}$, com $i \in \{1, 2, ..., L\}$, $j \in \{1, 2, ..., C\}$

Para H_0 temos que:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

Onde:

- O_{ij} : número de vezes que na amostra da população i a categoria j é observada.
- E_{ij} : valor esperado na população e na categoria j

E o p-valor é calculado da seguinte forma:

Para o teste de Homogeneidade usaremos a função `chi2_contingency` da biblioteca **Scipy**, cujo argumento de entrada é

- `observed`: tabela de contingência

Exemplo

Com o intuito de saber a eficacia da quimioterapia para dois diferentes tipos de câncer, um médico selecionou de maneira aleatória 100 pacientes com o câncer do Tipo I e 100 com o Tipo II, e dispôs os resultados na tabela abaixo. Verifique se existe relação entre a eficacia da quimioterapia e o tipo de câncer, ao nível de significância de `5%`.

Reação

Câncer

Pouca

Média

Alta

Tipo I

51

33

16

Tipo II

58

29

13

```
tabela_cancer = np.array([51, 33, 16, 58, 29, 13]).reshape(2,3)
tabela_cancer
```

```
array([[51, 33, 16],[58, 29, 13]])
```

```
chi2_contingency(tabela_cancer)
```

```
(1.0179506281189088, 0.6011112135514984, 2, array([[54.5, 31. , 14.5],[54.5, 31. , 14.5]]))
```

Como p-valor = 0.6011 é maior que o nível de significância, não rejeitamos a hipótese nula. Ou seja, há evidências de que o efeito da quimioterapia é igual para os dois tipos de câncer observados.

Teste de Independência

Agora considere que cada indivíduo da amostra será classificado conforme duas variáveis qualitativas. Em comparação ao teste de Homogeneidade, e simplificando o entendimento, ao invés de L populações, teremos uma população que pode ser classificada de L formas diferentes.

Para este teste teremos as seguintes hipóteses:

$H_0 : p_{i,j} = p_{i,\cdot} \cdot p_{\cdot,j}$, para todo $i = 1, ..., L$ e $j = 1, ..., C$

$H_1 : \exists\, i, j$, tais que $p_{i,j} \neq p_{i,\cdot} \cdot p_{\cdot,j}$, com $i \in \{1, \dots, L\}$, $j \in \{1, \dots, C\}$

Onde $p_{i,j}$ é a probabilidade do indivíduo ser classificado como i e j simultaneamente e $p_{i,\cdot}$ e $p_{\cdot,j}$ as probabilidades dos indivíduos serem classificados como i ou j de forma separada, respectivamente.

sob H_0 temos que:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

E o p-valor é calculado da seguinte forma:

Todo desenvolvimento do cálculo será igual ao teste de Homogeneidade, mas a interpretação do problema será diferente, assim como as hipóteses.

Exemplo

Agora, o médico gostaria de avaliar se a reação à quimioterapia independe do tipo de câncer do paciente. Para tanto, foram selecionados 180 pacientes que fazem tratamento de câncer com quimioterapia e suas informações foram dispostas na tabela abaixo. Verifique se "Tipo de câncer" e "Eficacia da quimioterapia" são variáveis independentes, ao nível de significância de 5%.

Reação

Câncer

Pouca

Média

Alta

Tipo III

38

32

30

Tipo IV

26

38

16

```
tabela_cancer = np.array([38, 32, 30, 26, 38, 16]).reshape(2,3)
tabela_cancer
```

```
array([[38, 32, 30],[26, 38, 16]])
```

```
chi2_contingency(tabela_cancer)
```

```
(4.862969720496892, 0.08790620719349594, 2, array([[35.55555556, 38.88888889,
25.55555556], [28.44444444, 31.11111111, 20.44444444]]))
```

Como p-valor = 0.0879 é maior que o nível de significância, não rejeitamos a hipótese nula. Ou seja, há evidências de que "Tipo de câncer" e "Eficacia da quimioterapia" são variáveis independentes.

Referências

- Casella, G. e Berger, R. L. **Inferência estatística**. Cengage Learning, 2010.
- Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) **SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.** Nature Methods, in press.
- Seabold, Skipper, and Josef Perktold. **"statsmodels: Econometric and statistical modeling with python."** Proceedings of the 9th Python in Science Conference, 2010.
- Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. **The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering**, 2011.
- John D. Hunter. **Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering**, 2007.
- Wes McKinney. **Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference**, 2010.
- Velarde, L. G. C. CAVALIERE, Y. F. **Apostila Inferência Estatística. Departamento de Estatística**. Universidade Federal Fluminense
- FARIAS, A. M. L. **Apostila de Estatística II**. Departamento de Estatística. 2017. Universidade Federal Fluminense.

Predição

Neste capítulo, vamos estudar sobre os algoritmos de predição, que são capazes de inferir se um dado pertence ou não a uma certa categoria. Dessa forma, eles são construidos utilizando as seguintes etapas: Pergunta → Amostra de entrada → Características → Algoritmo → Parâmetros → Avaliação.

Pergunta

O nosso objetivo é responder a uma pergunta de tipo "O dado A é do tipo x ou do tipo y?". Por exemplo, podemos querer saber se é possível detectar automaticamente se um e-mail é um spam ou um "ham", isto é, não spam. O que na verdade queremos saber é: "É possível usar características quantitativas para classificar um e-mail como spam?".

Amostra de Entrada

Uma vez formulada a pergunta, precisamos obter uma amostra de onde tentaremos extrair informações que caracterizam a categoria a qual um dado pertence (dados rotulados) e então usar essas informações para classificar outros dados não categorizados. O ideal é que se tenha uma amostra grande, assim teremos melhores parâmetros para construir nosso preditor.

No caso da pergunta sobre um e-mail ser spam ou não, temos acesso a base de dados "spambase" disponível no UCI Machine Learning Repository, onde cada linha dessa base é um e-mail e nas colunas temos a frequência (em porcentagem) de palavras, números e caracteres especiais presente em cada um deles. Nesse sentido, nossa variável de interesse (dependente) é a "Class" que classifica o e-mail como spam ou ham (não spam).

```
# instalando o pacote ucimlrepo pelo terminal
# pip install ucimlrepo
```

```
# importando a função necessária do pacote
from ucimlrepo import fetch_ucirepo

# busca a base de dados por id e retorna um objeto com os valores das variáveis dependente
spambase = fetch_ucirepo(id=94)

# separando as variáveis dependentes e independentes e transformando em um data frame
X = spambase.data.features
y = spambase.data.targets
```

```
# 1 - Spam / 0 - Ham
y
```

	Class
0	1
1	1
2	1
3	1
4	1

	Class
...	...
4596	0
4597	0
4598	0
4599	0
4600	0

Obtida a amostra, precisamos dividi-la em duas partes: o Conjunto de Treino e o Conjunto de Teste. O Conjunto de Treino será usado para construir e treinar o algoritmo, extraindo informações que ajudem a classificar ou prever uma categoria de dado. É crucial que o modelo seja desenvolvido apenas com base no Conjunto de Treino, para garantir que ele aprenda padrões gerais e não se ajuste excessivamente aos dados (evitando o overfitting).

Após o treinamento, o modelo deve ser aplicado ao Conjunto de Teste, que contém dados que não foram usados durante o treinamento. Isso permite avaliar o desempenho do modelo em situações reais, simulando como ele se comportará com dados novos.

```
# Instalando o pacote scikit-learn
# pip install scikit-learn
```

```
# Importando a função para dividir a amostra
from sklearn.model_selection import train_test_split
```

```
# Dividindo a amostra em teste e treino
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Características

Temos que encontrar agora características que possam indicar a categoria dos dados. Podemos, por exemplo, vizualizar algumas variáveis graficamente para obter uma ideia do que podemos fazer. No nosso exemplo de e-mails, podemos querer avaliar se a frequência de palavras "your" em um e-mail pode indicar se ele é um spam ou não.

```
# Instalando os pacotes necessários
# pip install pandas
# pip install matplotlib
# pip install seaborn
# pip install numpy
```

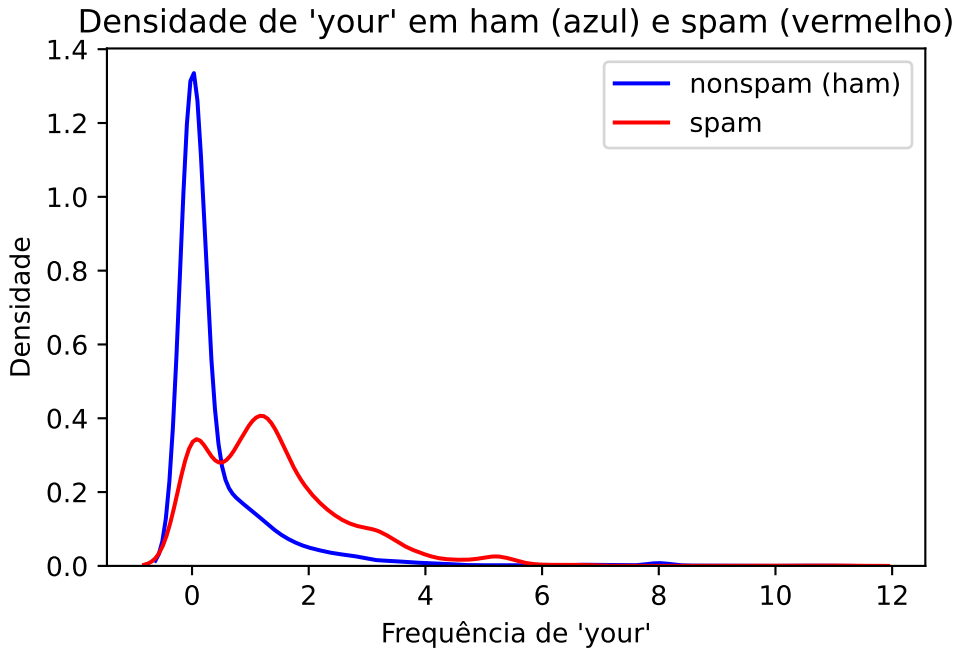
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# transformando em uma série pandas para podermos realizara filtragem
y = y.iloc[:, 0]
```

```
# filtrando os dados da coluna selecionada por característica
nonspam_data = X[y==0]['word_freq_your']
spam_data = X[y==1]['word_freq_your']
```

```
# gerando a linha dos ham
ax = sns.kdeplot(nonspam_data, color='blue', label='nonspam (ham)')
# gerando a linha dos spam
ax = sns.kdeplot(spam_data, color='red', label='spam')

# colocando título no gráfico e nos labels e adicionando legenda
ax.set_title("Densidade de 'your' em ham (azul) e spam (vermelho)")
ax.set_xlabel("Frequência de 'your'")
ax.set_ylabel("Densidade")
```

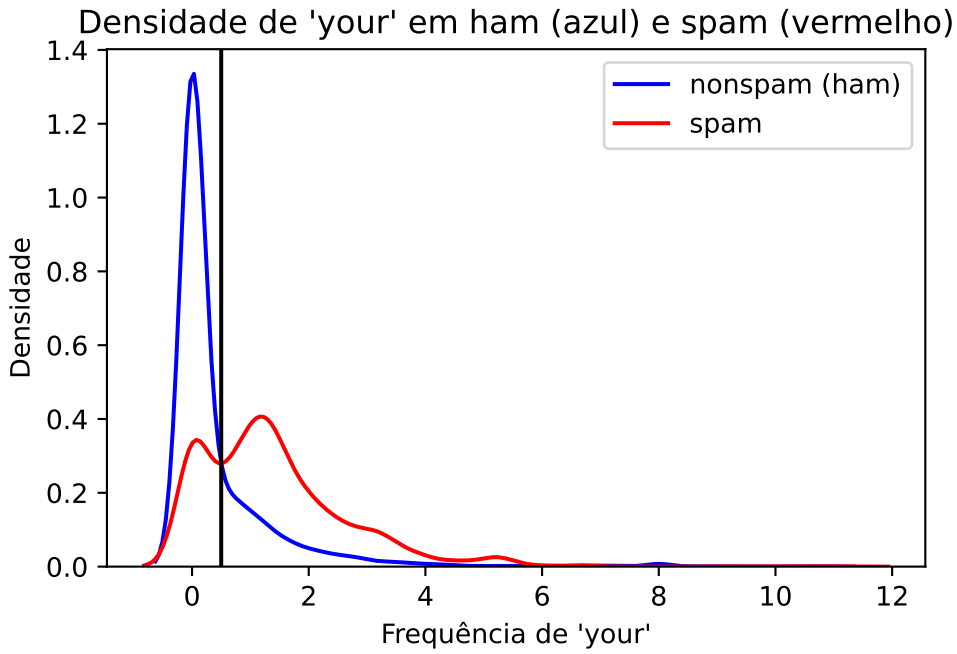
```
ax.legend()

# pset_lotando o gráfico
plt.show()
```



Pelo gráfico podemos notar que a maioria dos e-mails que são spam têm uma frequência maior da palavra "your". Por outro lado, aqueles que são classificados como ham (não spam) têm um pico mais alto perto do 0.

```
# gerando as linhas dos ham e spam
ax = sns.kdeplot(nonspam_data, color='blue', label='nonspam (ham)')
ax = sns.kdeplot(spam_data, color='red', label='spam')

# colocando título no gráfico e nos labels e adicionando legenda
ax.set_title("Densidade de 'your' em ham (azul) e spam (vermelho)")
ax.set_xlabel("Frequência de 'your'")
ax.set_ylabel("Densidade")

# adicionando o 'c'
ax.axvline(x=0.5, color='black', linestyle='-')
```

```
# mostrando a legenda
ax.legend()

# plotando o gráfico
plt.show()
```



Os e-mails à direita da linha preta seriam classificados como spam, enquanto que os à esquerda seriam classificados como não spam.

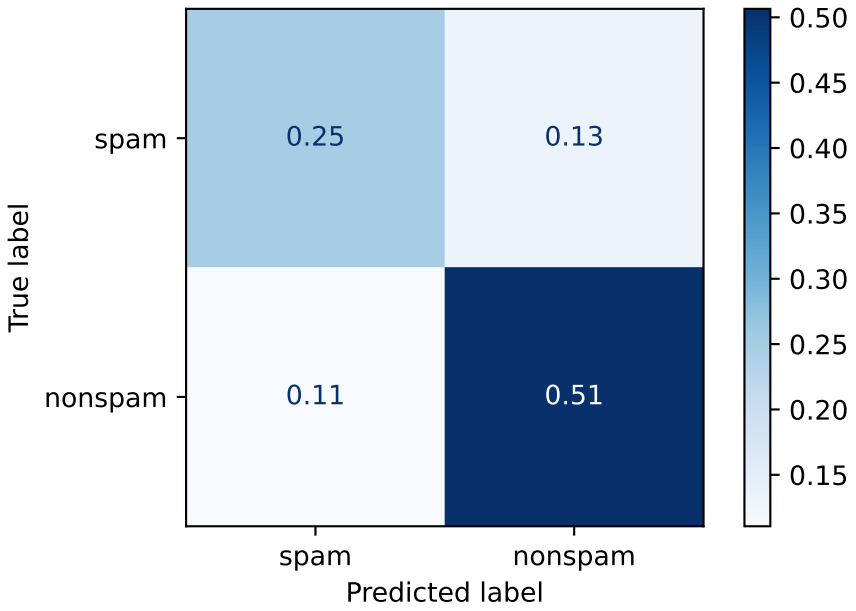
Avaliação

Agora vamos avaliar nosso modelo de predição

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
# Selecionando os dados e transformando em um array
predicao_treino = X_train['word_freq_your']

# Aplicando a regra que desejamos ("c")
predicao_treino = np.where(predicao_treino > 0.8, "spam", "nonspam")

# Traduzindo a legenda 0 e 1 para as strings
y_train_cm = np.where(y_train["Class"] == 1, "spam", "nonspam")

# Gerando a matriz de confusão do modelo
cm = confusion_matrix(y_train_cm, predicao_treino, labels=["spam", "nonspam"]) # labels se
cm = (cm / len(y_train_cm))

# Gerando a visualização da matriz de confusão
disp_treino = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["spam", "nonspam

# Plotando a visualização
disp_treino.plot(cmap=plt.cm.Blues)
```

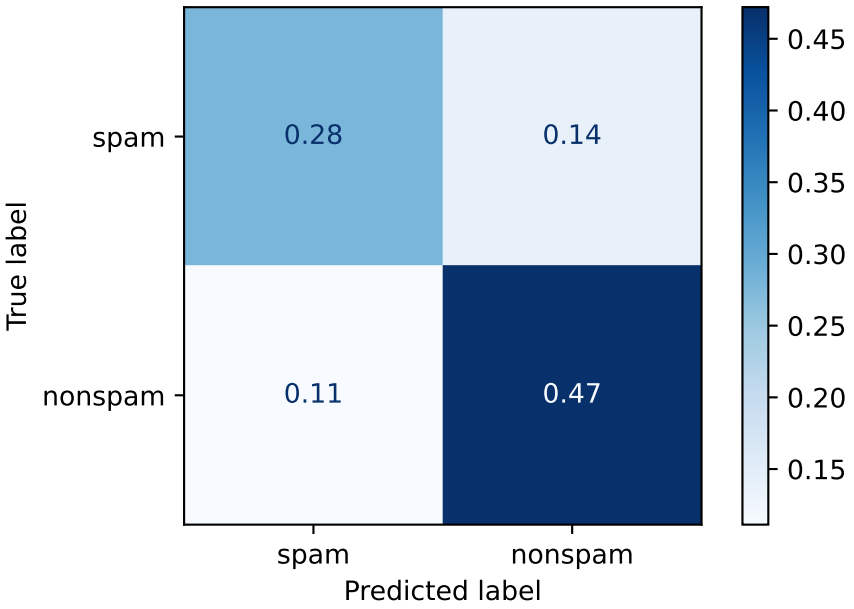
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x201957f1730>
```



Podemos ver que quando os e-mails não eram spam e classificamos como "não spam", de acordo com nosso modelo, em 50% do tempo nós acertamos. Quando os e-mails eram spam e classificamos ele em spam, por volta de 25% do tempo nós acertamos. Então, ao total, nós acertamos por volta de 51+25=76% do tempo. Então nosso algoritmo de previsão tem uma precisão por volta de 76% na amostra treino.

```
# realizando o mesmo processo acima para a amostra de teste
predicao_teste = np.where(X_test["word_freq_your"] > 0.8, "spam", "nonspam")
y_test_cm = np.where(y_test["Class"] == 1, "spam", "nonspam")
cm_teste = confusion_matrix(y_test_cm, predicao_teste, labels=["spam", "nonspam"]) / len(y
disp_teste = ConfusionMatrixDisplay(confusion_matrix=cm_teste, display_labels=["spam", "no
disp_teste.plot(cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2019afabbc0>
```



Já na amostra teste acertamos 47+28=75% das vezes. O erro na amostra teste é o que chamamos de erro real. É o erro que esperamos em amostras novas que passarem por nosso preditor.

Como construir um bom algoritmo de aprendizado de máquina?

O “melhor” método de aprendizado de máquina é caracterizado por:

- Uma boa base de dados;
- Reter informações relevantes;
- Ser bem interpretável;
- Fácil de ser explicado e entendido;
- Ser preciso;
- Fácil de se construir e de se testar em pequenas amostras;
- Fácil aplicar a um grande conjunto de dados.

Os erros mais comuns, que se deve tomar um certo cuidado, são:

- Tentar automatizar a seleção de variáveis (características) de uma maneira que não permita que você entenda como essas variáveis estão sendo aplicadas para fazer previsões;
- Não prestar atenção a peculiaridades específicas de alguns dados, como comportamentos estranhos de variáveis específicas;
- Jogar fora informações desnecessariamente.

Erros Amostrais

Este é um dos conceitos mais fundamentais com os quais lidamos na aprendizagem de máquina e previsão. Temos dois tipos de erros amostrais: o erro dentro da amostra (in sample error) e o erro fora da amostra (out of sample error).

Erro dentro da Amostra (In Sample Error)

A taxa de erro dentro da amostra refere-se ao erro calculado no mesmo conjunto de dados utilizado para treinar o modelo preditivo. Na literatura, isso é frequentemente denominado como “erro de resubstituição”. Em outras palavras, essa taxa de erro mede o quanto algoritmo de previsão se ajusta exatamente aos mesmos dados utilizados para o treinamento do modelo. No entanto, quando o modelo é aplicado a um novo conjunto de dados, é esperado que essa precisão diminua.

Erro fora da Amostra (Out of Sample Error)

É a taxa de erro que você recebe em um novo conjunto de dados. Na literatura às vezes é chamado de erro de generalização. Uma vez que coletamos uma amostra de dados e construímos um modelo para ela, podemos querer testá-lo em uma nova amostra, por exemplo uma amostra coletada em um horário diferente ou em um local diferente. Daí podemos analisar o quão bem o algoritmo executará a predição nesse novo conjunto de dados.

Algumas ideias-chave

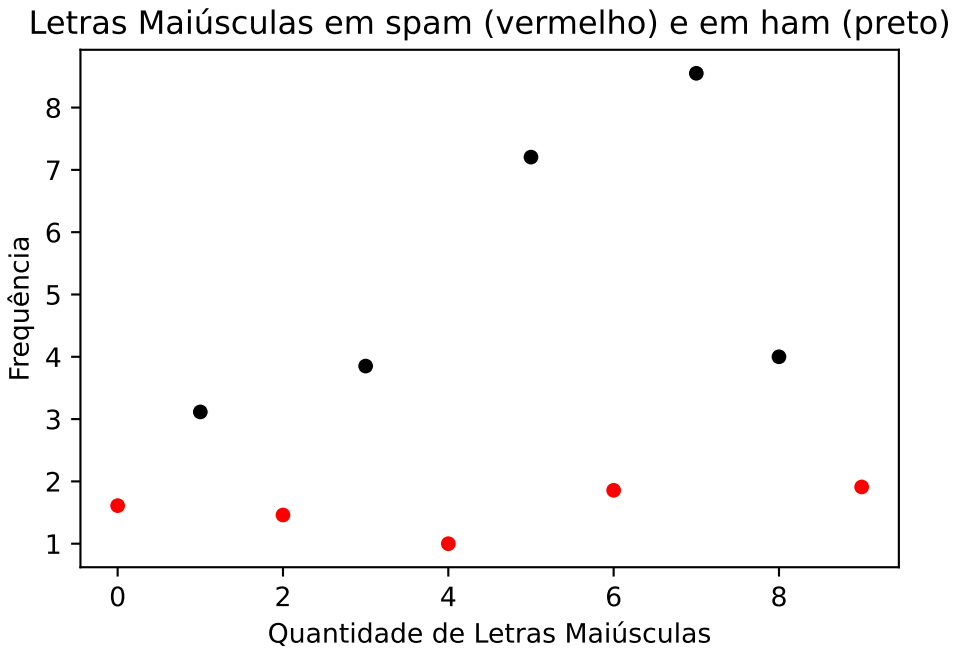
1. Quase sempre o erro fora da amostra é o que interessa.
2. Erro dentro da amostra é menor que o erro fora da amostra.
3. Um erro frequente é ajustar muito o algoritmo aos dados que temos. Em outras palavras, criar um modelo sobreajustado (também chamado de overfitting(*)).

(*) Overfitting é um termo usado na estatística para descrever quando um modelo estatístico se ajusta muito bem a um conjunto de dados anteriormente observado e, como consequência, se mostra ineficaz para prever novos resultados.

Vejamos um exemplo de erro dentro da amostra vs erro fora da amostra:

```
from sklearn.utils import resample
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Seed


# importando a base utilizando o pacote pandas
spam = pd.read_csv("Cadernos Grupo Python\\Criação e Avaliação de Preditores\\spam.csv")
spam["type_code"] = np.where(spam["type"] == "spam", 1, 0)
```

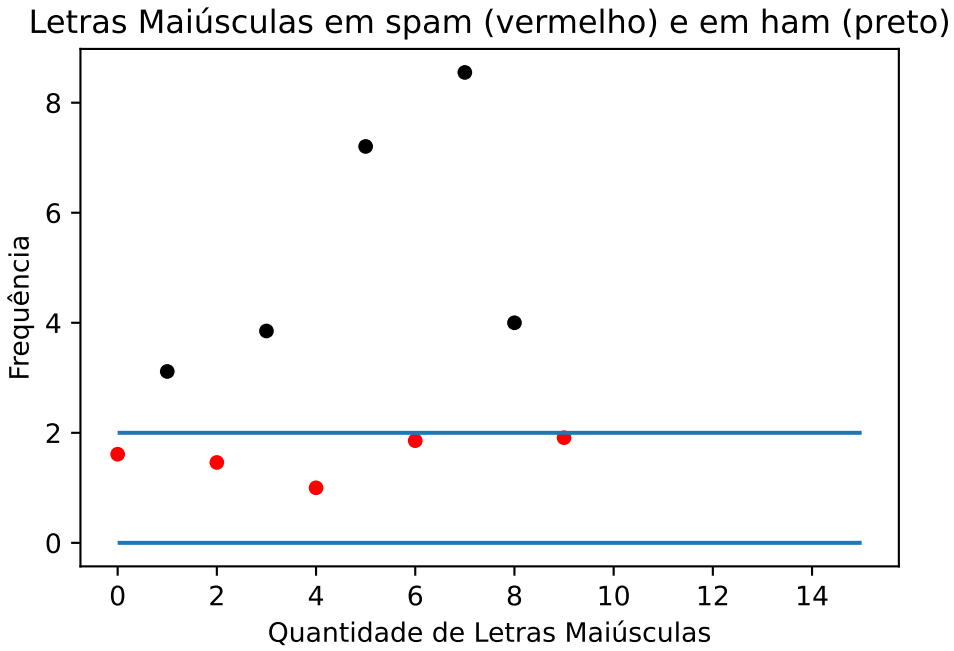
```
# vamos selecionar um amostra de tamanho dez da base
np.random.seed(151)
amostra_spam = resample(spam, n_samples=10, replace=False)

# criando gráfico de dispersão para a média de letras maiúsculas
cores = ['red', 'black']

sns.scatterplot(x=range(len(amostra_spam['capitalAve'])),
                y=amostra_spam['capitalAve'],
                c=[cores[label] for label in amostra_spam['type_code']])

plt.title('Letras Maiúsculas em spam (vermelho) e em ham (preto)')
plt.xlabel('Quantidade de Letras Maiúsculas')
plt.ylabel('Frequência')
plt.show()

# 322
```



Podemos notar que, em geral, as mensagens classificadas como spam possuem uma frequência maior de letras maiúsculas do que as mensagens classificadas como não spam. Com base nisso queremos construir um preditor, onde podemos classificar e-mails como spam se a frequência de letras maiúsculas for maior que uma determida constante, e não spam caso contrário.

Veja que se separarmos os dados pela frequência de letras maiúsculas maior que 2,5 e classificarmos o que está acima como spam e abaixo como não spam, ainda teríamos duas observações que não são spam acima da linha.

```
# Criando um gráfico de dispersão para a média de letras maiúsculas
cores = ['red', 'black']

sns.scatterplot(x=range(len(amostra_spam['capitalAve'])),
                y=amostra_spam['capitalAve'],
                c=[cores[label] for label in amostra_spam['type_code']])
plt.hlines(y = [0, 2], xmin=[0,0], xmax=[15, 15])
plt.title('Letras Maiúsculas em spam (vermelho) e em ham (preto)')
plt.xlabel('Quantidade de Letras Maiúsculas')
plt.ylabel('Frequência')
plt.show()
```




```
def modelo_sobreajustado(x):
    predicao = np.where((x >= 0) & (x <= 2), 'spam',
                        np.where((x < 0) | (x > 2), 'nonspam', 'nonspam'))

    return(predicao)
```

```
# Avaliando o modelo sobreajustado
resultado = modelo_sobreajustado(amostra_spam['capitalAve'])
confusion_matrix(resultado, amostra_spam['type'])
```

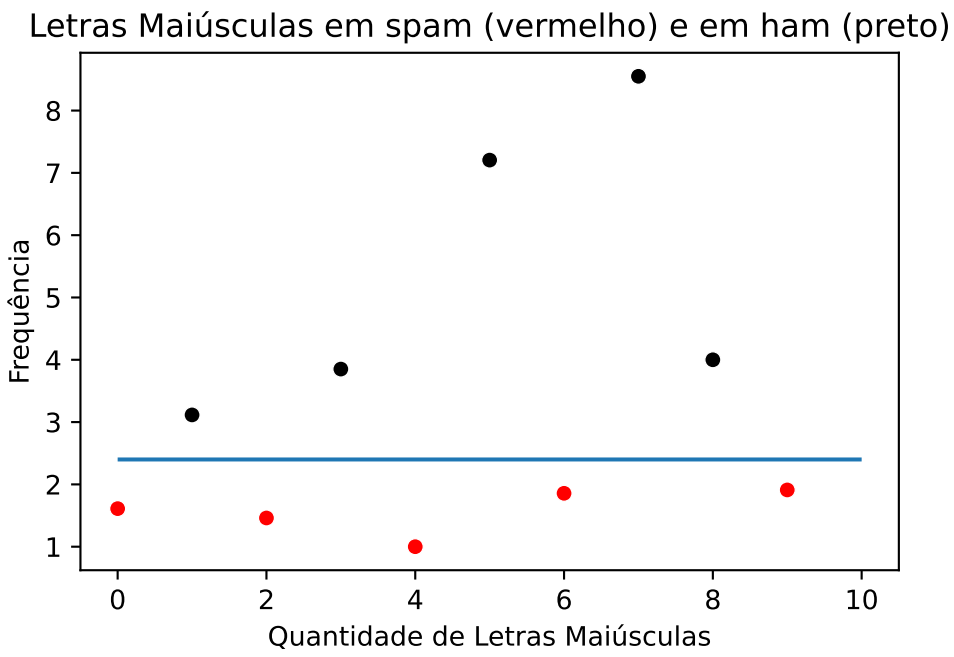
```
array([[0, 5],
       [5, 0]], dtype=int64)
```

Note que obtivemos uma precisão perfeita nessa amostra, como já era esperado. Nesse caso, o erro dentro da amostra é de 0%. Mas será que esse modelo é o mais eficiente em outros dados também?

Vamos usar essa segunda regra para criarmos um modelo mais geral:

- letras maiúsculas $> 2{,}4 \quad$ spam;
- letras maiúsculas $<= 2{,}4 \quad$ não spam.

```
cores = ['red', 'black']

sns.scatterplot(x=range(len(amostra_spam['capitalAve'])),
                y=amostra_spam['capitalAve'],
                c=[cores[label] for label in amostra_spam['type_code']])
plt.hlines(y = [2.4], xmin=[0], xmax=[10])
plt.title('Letras Maiúsculas em spam (vermelho) e em ham (preto)')
plt.xlabel('Quantidade de Letras Maiúsculas')
plt.ylabel('Frequência')
plt.show()
```




```
def modelo_geral(x):
    predicao = np.where(x >= 0, 'spam',
                        np.where(x < 0, 'nonspam', 'nonspam'))

    return predicao

# Avaliando o modelo sobreajustado
resultado_geral = modelo_geral(amostra_spam['capitalAve'])
confusion_matrix(resultado_geral, amostra_spam['type'])
```

```
array([[0, 0],
       [5, 5]], dtype=int64)
```

Observe que dessa forma temos um erro dentro da amostra de 20%. Vamos agora aplicar esses dois modelos para toda a base de dados:

```
# Aplicando o modelo na base toda e criando a tabela
confusion_matrix(modelo_sobreajustado(spam['capitalAve']), spam['type'])
```

```
array([[1183, 1493],
       [1605,  320]], dtype=int64)
```

```
confusion_matrix(modelo_geral(spam['capitalAve']), spam['type'])
```

```
array([[   0,    0],
       [2788, 1813]], dtype=int64)
```

Olhando para a precisão de nossos modelos:

```
np.sum(modelo_geral(spam['capitalAve']) == spam['type'])
```

```
1813
```

```
np.sum(modelo_sobreajustado(spam['capitalAve']) == spam['type'])
```

```
1503
```

Observe que utilizando o modelo sobreajustado obtivemos um erro fora da amostra de 40,77%, enquanto que com o modelo geral esse erro foi de 27,95%. Note que se queremos construir um modelo que melhor representa qualquer amostra que pegarmos, um modelo não sobreajustado possuirá uma precisão maior.

Criação e Avaliação de Preditores

O pacote scikit-learn é um pacote muito útil para o machine learning pois entrega muitas ferramentas que simplificam a criação de modelos preditivos. A função train_test_split exposta em capítulos anteriores é um exemplo. A construção de preditores feitos usando esse pacote será o foco desse capítulo.

Imports

Segue abaixo o código da importação dos módulos que usaremos nesse caderno:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import all_estimators
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LinearRegression
from sklearn.metrics import classification_report
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
```

Classificadores

Vamos utilizar a base de dados spam novamente para realizarmos o procedimento de predição para um e-mail (se ele é spam ou não spam), dessa vez utilizando o pacote scikit-learn.

Analisando a base spam

Vamos analisar a base spam.

```
#trocar essa url pelo caminho da base spam de vocês.
spam = pd.read_csv("Cadernos Grupo Python\\Criação e Avaliação de Preditores\\spam.csv")
print(spam)
```

```
      make  address   all  num3d   our  over  remove  internet  order  mail  \
0     0.00     0.64  0.64    0.0  0.32  0.00    0.00      0.00   0.00  0.00
1     0.21     0.28  0.50    0.0  0.14  0.28    0.21      0.07   0.00  0.94
2     0.06     0.00  0.71    0.0  1.23  0.19    0.19      0.12   0.64  0.25
3     0.00     0.00  0.00    0.0  0.63  0.00    0.31      0.63   0.31  0.63
4     0.00     0.00  0.00    0.0  0.63  0.00    0.31      0.63   0.31  0.63
...    ...      ...   ...    ...   ...   ...     ...       ...    ...   ...
4596  0.31     0.00  0.62    0.0  0.00  0.31    0.00      0.00   0.00  0.00
4597  0.00     0.00  0.00    0.0  0.00  0.00    0.00      0.00   0.00  0.00
4598  0.30     0.00  0.30    0.0  0.00  0.00    0.00      0.00   0.00  0.00
4599  0.96     0.00  0.00    0.0  0.32  0.00    0.00      0.00   0.00  0.00
4600  0.00     0.00  0.65    0.0  0.00  0.00    0.00      0.00   0.00  0.00

      ...  charSemicolon  charRoundbracket  charSquarebracket  \
0     ...          0.000             0.000                0.0
1     ...          0.000             0.132                0.0
2     ...          0.010             0.143                0.0
3     ...          0.000             0.137                0.0
4     ...          0.000             0.135                0.0
...   ...            ...               ...                ...
4596  ...          0.000             0.232                0.0
4597  ...          0.000             0.000                0.0
```

```
4598  ...          0.102             0.718               0.0
4599  ...          0.000             0.057               0.0
4600  ...          0.000             0.000               0.0

      charExclamation  charDollar  charHash  capitalAve  capitalLong  \
0               0.778       0.000     0.000       3.756           61
1               0.372       0.180     0.048       5.114          101
2               0.276       0.184     0.010       9.821          485
3               0.137       0.000     0.000       3.537           40
4               0.135       0.000     0.000       3.537           40
...               ...         ...       ...         ...          ...
4596            0.000       0.000     0.000       1.142            3
4597            0.353       0.000     0.000       1.555            4
4598            0.000       0.000     0.000       1.404            6
4599            0.000       0.000     0.000       1.147            5
4600            0.125       0.000     0.000       1.250            5

      capitalTotal     type
0              278     spam
1             1028     spam
2             2259     spam
3              191     spam
4              191     spam
...            ...      ...
4596            88  nonspam
4597            14  nonspam
4598           118  nonspam
4599            78  nonspam
4600            40  nonspam

[4601 rows x 58 columns]
```

Os dados são valores quantitativos que foram extraidos de 4601 e-mails e associam o e-mail a uma classificação: `spam` ou `nom-spam`. Há 2788 spams e 1813 e-mails normais.

Mesmo sem ter a íntegra dos e-mails originais, os dados que a base apresenta já são suficientes para determinar se o conteúdo é ou não spam para o escopo original da base. Em outros estudos, a seleção de dados pode ser diferente.

As primeiras 48 colunas possuem a frequência de certas palavras que estão fortemente associadas com a classificação, como "business". Da coluna 49 à coluna 54, frequência de caracteres de pontuação como a interrogação, também fortemente associado a spams. As colunas 55 a 57 contém a média de letras maiúsculas por palavra, o número de letras maiúsculas na

maior palavra e o número absoluto de letras maiúsculas respectivamente, todas importantes estatísticas no problema de classificação de e-mails.

A coluna 58, type, possui valores diferentes dos demais, categóricos, spam e nonspam. Ela é a coluna de rótulo da base. Com ela conseguimos saber a classificação original dos e-mails em spam ou não spam. Precisamos entregar ao scikit learn essa coluna separada das demais para não influenciar o modelo, o modelo estaria "colando" se fosse mantida.

Separação treino e teste

Para fazer a separação das amostras em treino e teste vamos primeiramente particionar a base de dados com a função `train_test_split()`. Ela divide a base mantendo a proporção das classes de classificação, nesse caso, spam e nonspam, mantendo a proporção da base original. Isso evita que ao acaso possa sair da separação uma base para treino muito diferente da original, que afeta severamente a qualidade do modelo.

```
# Separando a coluna de rótulo das demais
XSpam = spam.loc[:, spam.columns != "type"]
YSpam = spam["type"]

# Separação de amostras treino e teste
x_trainSpam, x_testSpam, y_trainSpam, y_testSpam = train_test_split(XSpam, YSpam, test_siz
```

No caderno de Introdução a Programação com Python falamos que é importante criar variáveis com nomes elucidativos.Esse X e Y podem parecer estranho a partir desse princípio, porém, essa nomenclatura de variáveis é um padrão utilizado por muitos no Python para fazer a separação do rótulo das demais colunas. Assim como numa função matemática, X -> Y, colunas de dados -> coluna de rótulo, por isso X e Y.

O parâmetro `test_size` recebe a proporção de dados teste dentro do total, logo, seguindo o exemplo, 40% da base é teste e 60% é treino.

O parâmetro `random_state` é uma maneira de assegurar que a separação que está sendo feita aqui é a separação que está sendo feita por vocês, para fins didáticos. Ele trapaceia a aleatoriedade do sorteio, pré-determinando o resultado a partir do número inserido, conhecido como seed. Isso é útil também se quisermos guardar a aleatoriedade usada para testar novamente depois. Normalmente sempre inserimos uma seed para que possamos replicar o experimento caso seja do nosso desejo no futuro.

O resultado da função train_test_split são 4 bases novas, `x_train`, `x_teste`, `y_train` e `y_test`. Dados para treino sem rótulo, dados para teste sem rótulo, rótulo dos dados para treino, rótulo dos dados para teste, gerados a partir do `X` (dados sem rótulo) e do `Y` (rótulo

dos dados) que inserimos dentro da função `train_test_split`. Acrescentei `Spam` no final de cada variável para diferenciar do regressor que faremos a seguir.

Seleção de algoritmo de aprendizado de máquina para o classificador.

Dado que já foi feito a separação das amostras treino e teste, o próximo passo é realizarmos o treinamento. Para isso é preciso escolher um dos algoritmos para ser utilizado. O scikit-learn suporta uma infinidade de diferentes modelos que podemos escolher.

Uma lista com todos os algoritmos implementados no pacote scikit-learn pode ser obtida com o seguinte comando:

```
# para ver todos os algoritmos (ou como o scikit-learn chama, estimadores)
print(all_estimators())

# para filtrar os algoritmos por seu tipo
print(all_estimators(type_filter='classifier'))
```

```
[('ARDRegression', <class 'sklearn.linear_model._bayes.ARDRegression'>), ('AdaBoostClassifie
[('AdaBoostClassifier', <class 'sklearn.ensemble._weight_boosting.AdaBoostClassifier'>), ('Ba
```

`type_filter` pode receber: - “regressor” para regressores - “classifier” para classificadores - “cluster” para agrupadores - “transformer” para transformadores - “outlier_detector” para detectores de anomalia

Nesse e nos próximos cadernos focaremos mais nos regressores e classificadores, de aprendizado de máquina supervisionado.

Escolher entre quais algoritmos utilizar pode variar de acordo com o problema que queremos resolver. (escrever aqui sobre como selecionar modelos a ser utilizado.)

Para o nosso exemplo vamos utilizar regressão logística (em inglês, LogisticRegression), um algoritmo simples comumente utilizado em classificação.

Cada modelo no scikit-learn precisa ser importado, podemos importar a biblioteca inteira ou apenas o modelo que queremos. Como o scikit-learn é uma biblioteca pesada, idealmente apenas importamos aquilo que formos utilizar. No início desse caderno, nos imports, importamos a regressão logística com `from sklearn.linear_model import LogisticRegression`, esse é o comando para importar a regressão logística. Cada algoritmo está localizado em um lugar diferente da biblioteca e possui formas de uso que podem variar. Nesse e nos próximos cadernos falaremos apenas de alguns.

Criação e treino do classificador

Agora vamos criar o nosso classificador. Para isso vamos usar a função `LogisticRegression()` para instanciá-lo e o método `.fit()` para treiná-lo.

```
modelSpam = LogisticRegression(max_iter=3000)
modelSpam.fit(x_trainSpam, y_trainSpam)
```

```
LogisticRegression(max_iter=3000)
```

A função `LogisticRegression` que cria o modelo de regressão logística possui diversos parâmetros que possuem valores padrão. Alterei apenas o `max_iter` para 3000 porque o padrão de 100 não é suficiente para o modelo convergir na base de dados utilizada. Falaremos mais detalhadamente sobre como cada algoritmo funciona nos próximos cadernos.

Utilização do classificador

Uma vez que ajustamos o classificador podemos aplicá-lo na amostra teste, para estimarmos sua qualidade. Para isso utilizamos o método `.predict()`. Dentro do método passamos a base de dados gostaríamos de realizar a predição, nesse caso a base teste.

```
y_predSpam = modelSpam.predict(x_testSpam)
print(y_predSpam)
```

```
['nonspam' 'spam' 'nonspam' ... 'nonspam' 'nonspam' 'nonspam']
```

Ao fazermos isso obtemos uma série de predições para as classes dos e-mails do conjunto teste.

Avaliação do classificador

Podemos realizar a avaliação do modelo comparando os resultados da predição com os reais rótulos dos e-mails que salvamos na variável `y_test`, por meio da função `confusion_matrix()`. Essa função cria uma matriz de confusão a partir do resultado da predição e os rótulos originais.

A matriz de confusão é a matriz de comparação feita após a predição, onde as linhas correspondem ao que foi previsto e as colunas correspondem à verdade conhecida.

A matriz de confusão para o problema de predição dos e-mails em spam ou não spam fica da seguinta forma:

		Real	
		Spam	Não Spam
Previsão	Spam		
	Não Spam		

Onde na primeira coluna se encontram os elementos que possuem a característica de interesse (os e-mails que são spam), e, respectivamente nas linhas, os que foram corretamente identificados - o qual são chamados de **Verdadeiros Positivos (VP)** - e os que foram erroneamente identificados - os **Falsos Negativos (FP)**. Na segunda coluna se encontram os elementos que não possuem a característica de interesse (os e-mails que são ham) e, respectivamente nas linhas, os que foram erroneamente identificados - o qual são chamados de **Falsos Positivos (FN)** - e os que foram corretamente identificados - os **Verdadeiros Negativos (VN)**.

Com as devidas classificações a matriz de confusão fica da seguinte forma:

		Real	
		Spam	Não Spam
Previsão	Spam		
	Não Spam		

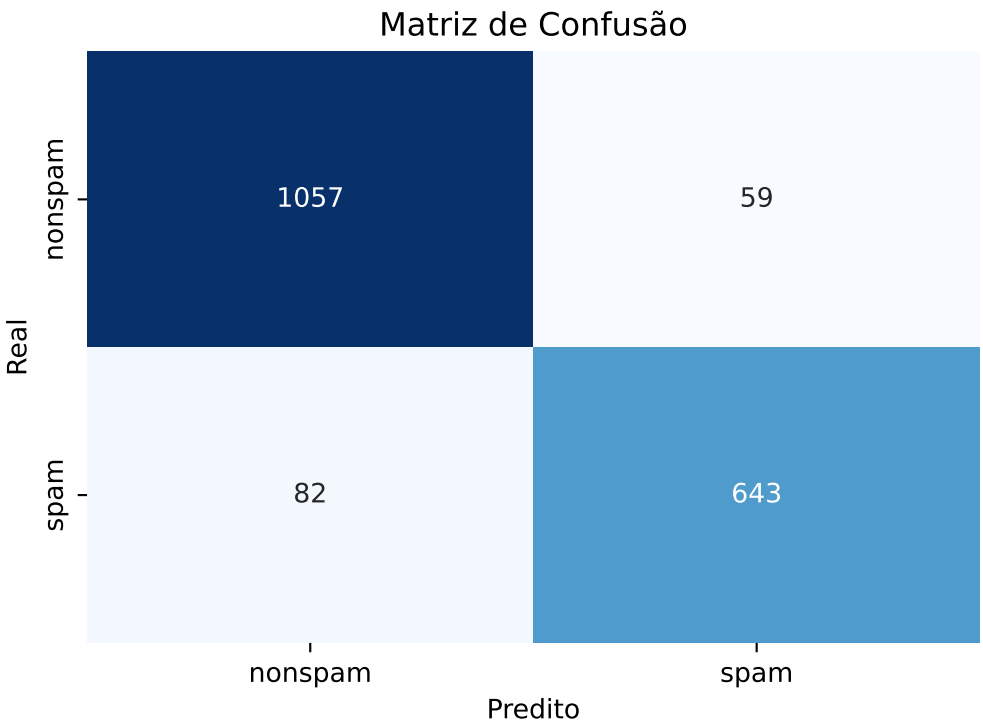
Dentro da função passamos as predições que obtemos pelo modelo ajustado e as reais classificações dos e-mails do conjunto teste.

```
cmSpam = confusion_matrix(y_testSpam, y_predSpam)
print(cmSpam)
```

```
[[1057   59]
 [  82  643]]
```

O que acharam da matriz confusão? Alguns números numa matriz do Python não é a melhor apresentação. Para deixar mais apresentável podemos fazer um gráfico:

```
plt.figure(figsize=(6,4))
sns.heatmap(cmSpam, annot=True, fmt='d', cmap='Blues', xticklabels=["nonspam", "spam"], yt
plt.xlabel('Predito')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()
```



Só o gráfico com números em valor absoluto não é o suficiente para avaliar o modelo. Podemos a partir desses números usar algumas métricas, muitas existem que podemos usar a partir desses valores. Falaremos mais sobre sensibilidade, especificidade e acurácia.

Existem classificadores com mais de duas classes. A matriz de confusão na verdade tem cardinalidade de acordo com o número de classes e a nomenclatura verdadeiro positivo, verdadeira negativo, falso positivo, falso negativo, ganha uma complexidade maior. Por simplicidade, estamos usando apenas um classificador com duas classes.

Sensibilidade

Definição (Sensibilidade): A sensibilidade de um método de predição é a porcentagem dos elementos da amostra que possuem a característica de interesse e foram corretamente identificados. Para o nosso exemplo dos e-mails, a sensabilidade é a porcentagem dos e-mails que são spam e foram classificados pelo nosso algoritmo de predição como spam.

Ou seja, podemos escrever $Sensibilidade = \frac{VP}{VP+FN}$

Especificidade

Definição (Especificidade): A especificidade de um método de predição é a porcentagem dos elementos da amostra que não possuem a característica de interesse e foram corretamente identificados. Para o nosso exemplo dos e-mails, a especificidade é a porcentagem dos e-mails que são "ham" e o algoritmo de predição os classificou como tal.

Ou seja, podemos escrever $Especificidade = \frac{VN}{VN+FP}$

Acurácia

Avalia a porcentagem de acertos que tivemos em geral. Ou seja, somamos o número de Verdadeiros Positivos com o número de Verdadeiros Negativos e dividimos pelo tamanho da amostra.

$Acurcia = \frac{VP+VN}{VP+VN+FN+FP}$

Relação entre métricas

Quando obtemos as sensibilidades e as especificidades de diferentes preditores, naturalmente surge o questionamente: qual deles é melhor para estimar as verdadeiras características de interesse? A resposta depende do que é mais importante para o problema.

Se identificar corretamente os positivos for mais importante, utilizamos o preditor com maior sensibilidade. Se identificar corretamente os negativos for mais importante, utilizamos o preditor com maior especificidade.

Acurácia da uma visão mais geral. Muitos cientistas de dados a consideram a métrica mais importante para classificadores, porém, acurácia não é tudo. Um modelo pode ser feito de maneira errada e mesmo assim ter alta acurácia. Em certas situações o debate Sensibilidade X Especificidade pode acabar reduzindo a acurácia e produzir um modelo melhor para o uso.

Outro exemplo de situação adversa para análise exclusiva da acurácia é quando uma das classes da variável resposta é muito mais presente no conjunto de dados que outra, o algoritmo do classificador pode acabar chutando tudo ou quase tudo como uma classe só e teremos um modelo com grande acurácia e péssima especificidade. Falaremos sobre como lhidar com base de dados desbalanceadas no futuro.

Em classificadores com mais de duas classes a nomenclatura de especificidade e sensibilidade perde o sentido. Nesse caso é apenas `recall` entre as diferentes classes. Para duas classes, recall da classe positiva é sensibilidade e da negativa, especificidade.

Análise do resultado final do classificador de Spam

Faremos a análise do resultado final do classificador da base Spam com as funções prontas do scikit-learn para o cálculo da sensibilidade, especificidade e acurácia.

A função `classification_report()` entrega todas essas métricas e mais algumas. Por suportar classificadores com mais de duas classes, ele usa a coluna recall. Sensibilidade é o recall da classe positiva e especificidade é o recall da negativa.

```
print(classification_report(y_testSpam, y_predSpam))
```

```
              precision    recall  f1-score   support

     nonspam       0.93      0.95      0.94      1116
        spam       0.92      0.89      0.90       725

    accuracy                           0.92      1841
   macro avg       0.92      0.92      0.92      1841
weighted avg       0.92      0.92      0.92      1841
```

Podemos observar que obtivemos uma sensibilidade de 0.89, acurácia de 0.92 e especificidade de 0.95. Podemos afirmar com esses números que o modelo acertou a classificação em 92% dos casos e que ele é melhor para identificar mensagem que não são spam (95% de acerto) do que mensagens que são spam (89% de acerto). Em muitos casos desejamos apenas que o modelo alcance uma % em específica para ter um modelo bom o suficiente. Se considerarmos o objetivo como 90% de acurácia e no mínimo 85% para sensibilidade e especificidade, alcançamos o objetivo.

Regressores

Agora vamos utilizar a base de dados faithful para tentar prever o tempo de espera (waiting) entre uma erupção e outra de um gêiser dado a duração das erupções (eruption).

Analisando a base faithful

```
faithfulDF = pd.read_csv("Cadernos Grupo Python\\Criação e Avaliação de Preditores\\faithf
print(faithfulDF)
```

```
     eruptions  waiting
0        3.600       79
1        1.800       54
2        3.333       74
3        2.283       62
4        4.533       85
..         ...      ...
267      4.117       81
268      2.150       46
269      4.417       90
270      1.817       46
271      4.467       74

[272 rows x 2 columns]
```

A base de dados `faithful` contém dados sobre as eurupções do geiser Old Faithful no parque nacional de Yellowstone nos Estados Unidos da América. A base de dados possui medições de das variáveis:

- `eruptions`: Eurupções, traduzido do inglês, contém a duração das eurupções em minutos.
- `waiting`: Espera, traduzido do inglês, contém o tempo de espera até a eurupção seguinte em minutos.

Estamos assumindo antes de criar o modelo computacional que existe uma relação entre as eurupções e o tempo de espera para a eurupção seguinte. Vamos usar a variável eruptions para prever a variável waiting.

Separação treino e teste

Faremos identicamente a como fizemos para o classificador da base spam.

```
# Separando a coluna de rótulo das demais
XFaithful = faithfulDF.loc[:, faithfulDF.columns != "waiting"]
YFaithful = faithfulDF["waiting"]

# Separação de amostras treino e teste
x_trainFaithful, x_testFaithful, y_trainFaithful, y_testFaithful = train_test_split(XFaith
```

Seleção de algoritmo de aprendizado de máquina para o modelo.

Para a seleção do algoritmo de aprendizado de máquina que usaremos podemos consultar novamente a lista do scikit-learn, agora para regressores.

```
# para ver todos os algoritmos (ou como o scikit-learn chama, estimadores)
print(all_estimators())

# para filtrar os algoritmos por seu tipo
print(all_estimators(type_filter='regressor'))
```

```
[('ARDRegression', <class 'sklearn.linear_model._bayes.ARDRegression'>), ('AdaBoostClassifie
[('ARDRegression', <class 'sklearn.linear_model._bayes.ARDRegression'>), ('AdaBoostRegressor
```

Usaremos o LinearRegression, usado quando queremos estimar a relação entre duas variáveis contínuas. É um dos modelos de aprendizado de máquina mais simples e rápidos. Falaremos mais sobre o funcionamento dele em próximos cadernos.

Criação e treino do regressor

Agora vamos criar o nosso regressor. Para isso vamos usar a função `LinearRegression()` para instanciá-lo e o método `.fit()` para treiná-lo. Assim como o classificador de regressão

logística, precisamos importá-lo antes de instanciá-lo. No início do caderno o import está lá: `from sklearn.linear_model import LinearRegression`

```
modelFaithful = LinearRegression()
modelFaithful.fit(x_trainFaithful, y_trainFaithful)
```

```
LinearRegression()
```

Utilização do regressor

A utilização do regressor é a mesma do classificador. Basta passar pelo método predict o conjunto de dados que queremos usar para prever a variável resposta. Como queremos testar o modelo, usaremos o conjunto teste que haviámos separado.

```
y_predFaithful = modelFaithful.predict(x_testFaithful)
print(y_predFaithful)
```

```
[77.01894052 86.86457141 54.052711   77.53713162 81.34065429 73.39160283
 72.0132145  54.052711   52.84014382 81.85884539 79.95190215 77.01894052
 55.0890932  75.80637335 75.29854607 79.60989602 52.66395885 82.71904262
 84.79180701 53.35833492 59.57662812 73.5677878  83.57923984 80.82246319
 79.26788989 69.24607403 78.2315077  64.75853911 55.60728429 77.37131047
 53.70034105 78.57351382 85.48618309 55.25491435 70.4586412  57.84586984
 77.01894052 52.84014382 58.36406094 77.1951255  55.0890932  80.48045707
 85.30999811 77.7133166  79.96226597 54.052711   80.82246319 80.48045707
 54.5709021  85.48618309 54.21853215 83.41341869 57.16185759 74.60417
 79.96226597 59.05843702 84.27361592 79.96226597 52.84014382 79.78608099
 81.34065429 84.44980089 53.35833492 83.41341869 69.08025288 80.64627822
 81.68266042 78.7496988  58.02205482 84.96799199 58.02205482 54.21853215
 76.50074942 85.82818921 84.44980089 80.12808712 82.03503037 77.88950157
 55.43109932 55.43109932 79.78608099 53.5345199 ]
```

Ao fazermos isso obtemos uma série de predições para waiting a partir do conjunto teste.

Avaliação do regressor

Assim como há diversas formas de compararmos a qualidade dos classificadores, há também diversas formas de compararmos regressores. O que estudaremos agora é o MAE, MSE e R^2. Mais formas de comparação de regressores também serão vistas futuramente.

MAE

O **erro médio absoluto** é uma métrica estatística utilizada para medir a **distância entre os valores previstos e os reais**, ou seja, o **erro**. Para calculá-lo, obtemos a média das diferenças absolutas entre os valores previstos e os valores reais, como mostrado na fórmula a seguir:

$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

- `n`: número de observações
- `yi`: valor real
- `ŷi`: valor predito
- `|yi - ŷi|`: Erro absoluto de uma predição

Interpretação: O cálculo é baseado na distância entre os dados previstos e reais (erro), portanto, quanto **menor** o valor do MAE, melhor será o desempenho do modelo pois isso indica uma **maior proximidade** entre esses valores.

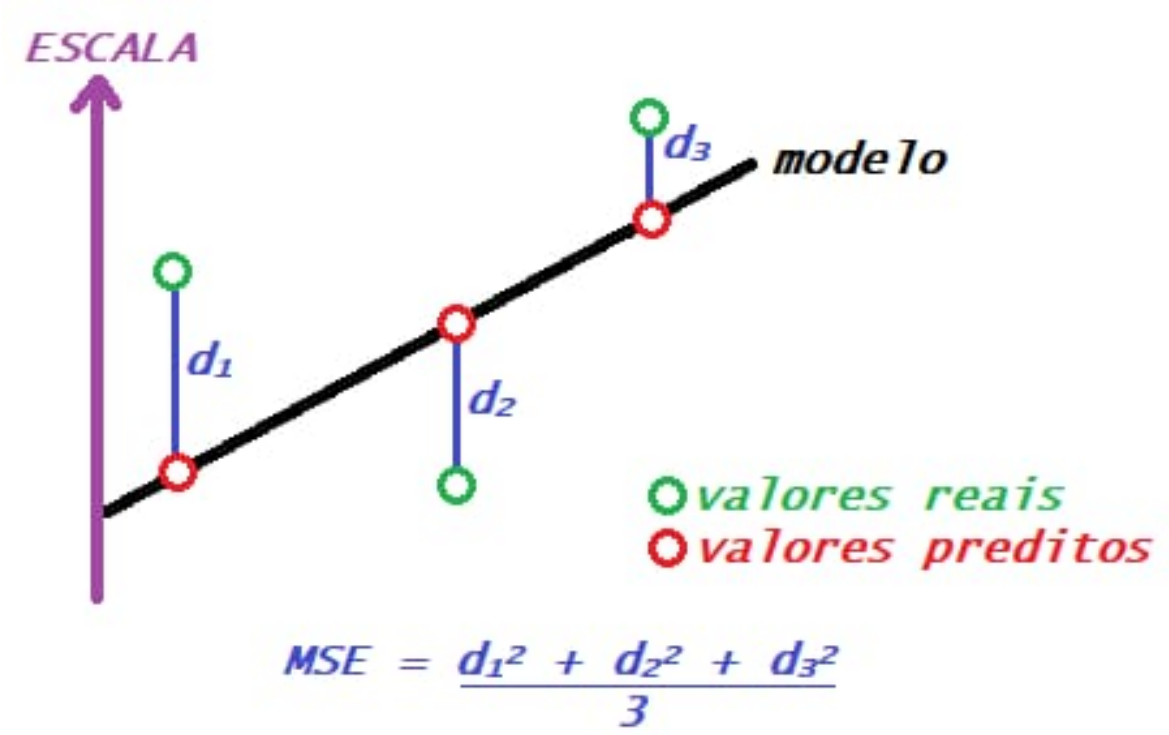
Limitações: O MAE é pouco sensível a outliers, ou seja, **erros grandes e pequenos tendem a ter a mesma influência** e isso pode ser um problema em situações onde erros grandes seriam inaceitáveis por exemplo.

MSE

O **erro médio quadrático** segue a mesma lógica do MAE, calcula a distância entre o valor predito e o real, mas utilizando o **quadrado** ao invés do módulo, o que o torna mais **sensível a outliers** e penalizando mais **erros grandes**. Ele também pode ser utilizado como **função de custo** (loss function) em modelos de regressão, como forma de **otimizar os parâmetros do modelo**, especialmente em **regressão linear** e em **redes neurais**, pois costumar ser **diferenciável**, o que facilita minimizar a função utilizando **algoritmos de otimização**, como o **gradiente descendente**, por exemplo. Sendo assim, sua fórmula é:

$MSE = \frac{1}{n}\sum_{i=1}^{n}(yi - \hat{y}i)^2$

- `n`: número de observações
- `yi`: valor real
- `ŷi`: valor predito
- `(yi - ŷi)^2`: Erro ao quadrado de uma predição



Interpretação: Assim como o MAE, quanto **menor** o MSE de um modelo, melhor. Contudo, sua interpretação requer cuidado, pois um mesmo valor de MSE pode ser suficientemente próximo do zero ou não dado o contexto, além de **não estar na mesma unidade dos dados**.

Limitações: Como citado anteriormente, o MSE tem uma **sensibilidade maior a outliers**, o que pode aumentar seu valor e indicar que o desempenho do modelo está pior quando isso não é verdade. Além disso, ele é **dependente da escala dos dados**, de maneira que modelos que preveem faixas de valor mais altas terão naturalmente um MSE maior, tornando a comparação entre diferentes bases de dados injusta. Para contornar esse problema, podemos **normalizar ou padronizar** os dados antes de realizar o cálculo.

R^2

R^2, também conhecido como coeficiente de determinação, é uma medida que **explica a proporção da variância** da variável dependente que pode ser explicada pelas variáveis independentes, ou seja, o quão bem os dados se **ajustam ao modelo**. Dessa forma, ele assume valores de zero até um, onde 1 significaria que os dados se encaixaram perfeitamente. Existem diversas maneiras de se calcular seu valor, vamos abordar a mais comum nesse texto e ela se baseia em utilizar os **erros ao quadrado** e a **variância total do modelo**, de maneira que

essa abordagem destaca o quão melhor o modelo de regressão funciona em comparação com simplesmente prever a média da resposta

$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$

onde RSS (residual sum of squares) é a soma dos erros ao quadrado e TSS (total sum of squares) é a variância total.

Interpretação: R^2 é a porcentagem de variação explicada pela relação entre duas variáveis. R^2 é sempreuma porcentagem entre 0% e 100% com 0%, o que significa que seus valores previstos não lhe disseram absolutamente nada sobre Y e 100%, o que significa que seus valores previstos eram perfeitamente precisos. Então, para responder ao exemplo que você deu, se $R^2 = 0.8$, então diríamos: "80% da variação observada na variável Y é explicada por nossa regressão (ou então você gerou previsões para Y)". Contudo, devemos nos atentar a valores muito altos pois eles podem indicar overfitting.

Limitações: É importante lembrar que o R^2 mede a **correlação**, mas não mede a **causalidade**. Só porque nossos preditores explicam o resultado não significa que eles causam isso, porque correlação ainda não significa causalidade. Além disso, o R^2 não indica se as previsões são precisas.

Relação entre métricas

- O MAE e o MSE tendem a diminuir juntos quando o modelo melhora
- O MSE e R^2 estão inversamente relacionados
- MSE $\geq MAE^2$ (por desigualdade de Jensen)

Análise do resultado final do regressor de Faithful

Faremos a análise do resultado final do regressor da base Faithful com as funções prontas do scikit-learn para o cálculo do MAE, MSE e R^2.

As funções são `mean_absolute_error`, `mean_squared_error`, `r2_score` respectivamente.

```
#MAE
print(f"MAE: {mean_absolute_error(y_testFaithful, y_predFaithful)}")

#MSE
print(f"MSE: {mean_squared_error(y_testFaithful, y_predFaithful)}")

#R-Squared
print(f"R ao quadrado: {r2_score(y_testFaithful, y_predFaithful)}")
```

```
MAE: 4.656047512563938
MSE: 35.3906300883027
R ao quadrado: 0.8332386607649062
```

Pelo MAE podemos aferir que nosso regressor erra, em média, por 4.65 minutos a previsão de espera entre as eurupções. Se nosso critério de aceitação do modelo for pior que 4.65 minutos de erro, então esse modelo é bom.

O R^2 afere que 83.3% da variável waiting pode ser explicado pela variável etuption, logo podem existir mais variáveis que melhorariam o desempenho final do modelo que não entraram no conjunto de dados.

Validação Cruzada

Imports

Segue abaixo o código da importação dos módulos que usaremos nesse caderno:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.utils import resample
```

Validação Cruzada

Como exposto no caderno de criação e avaliação de preditores, existem diversos métodos de aprendizado de máquina que podemos usar para construir um preditor. Então como saber qual método é melhor? Quais parâmetros usar? Um jeito de resolver essas questões é usando a validação cruzada.

A Validação Cruzada nos permite comparar diferentes métodos de aprendizado de máquina ou parâmetros e avaliar qual funcionará melhor na prática.

Então o que vamos fazer é, para cada método,

1) Separar os dados em conjunto de treino e conjunto de teste.
2) Treinar um modelo no conjunto de treino.
3) Avaliar no conjunto de teste.
4) Repetir os passos 1-3 e estimar o erro.

Já sabemos que não é uma boa ideia usar toda a base de dados para treinar o nosso preditor e então podemos dividir por exemplo os primeiros 75% dos dados para treino e 25% finais para teste. Mas, e se esse não for o melhor jeito de dividir nossos dados? E se o melhor jeito de fazer essa divisão for usando os primeiros 25% para teste e o restante para treino? A Validação cruzada leva em consideração todas essas divisões usando uma de cada vez e tirando a média dos resultados no final. Para isso veremos como realizar alguns métodos de reamostragem, para utilizarmos várias amostras possíveis e não ficarmos dependentes de uma única amostra.

Utilizaremos a base Spam ao longo do caderno.

```
# trocar essa url pelo caminho da base spam de vocês.
Spam = pd.read_csv("Cadernos Grupo Python\\Validação Cruzada\\spam.csv")
# Separando o rótulo das demais variáveis
XSpam = Spam.loc[:, Spam.columns != "type"]
YSpam = Spam["type"]
```

Alguns Métodos de Reamostragem

FALAR SOBRE PORQUE FAZER REAMOSTRAGEM E O QUE ELA É DE FATO

Falaremos de 3 métodos de reamostragem: K-fold, Repeated K-fold e Bootstrap. Daremos uma definição e exemplo de código de cada um para depois entrar em como utilizar na validação cruzada.

K-fold

Definição

Este método consiste em fatiar os dados em k pedaços iguais. Utilizamos um pedaço para o teste e os demais para o treino. Então realizamos esse procedimento k vezes, de modo que em cada repetição um novo pedaço seja utilizado para o teste. Para avaliar o erro nós tiramos a média de todos os erros de todas as replicações.

Exemplo: K-fold com 10 partes:

Quanto maior o k escolhido obtemos menos viés, porém mais variância. Em outras palavras, você terá uma estimativa muito precisa do viés entre os valores previstos e os valores verdadeiros, porém altamente variável. Agora quanto menor o k escolhido, mais viés e menos variância. Ou seja, não iremos necessariamente obter uma boa estimativa do viés, mas ela será menos variável.

OBS: Quando o k é igual ao tamanho da amostra, o método é também conhecido como *leave-one-out*.

Código de exemplo

Vamos utilizar reamostragem por `k-fold` no conjunto de dados spam usando a implementação do scikit-learn. O scikit-learn possui algumas diferentes implementações para a reamostragem por k-folds. Usaremos a função `StratifiedKFold` porque ela faz a separação de k-folds preservando a proporção original da coluna com os rótulos.

```
# Criação dos k-folds
kfSpam = StratifiedKFold(n_splits=10, shuffle=True, random_state=11).split(XSpam, YSpam)
```

Parâmetros da função `StratifiedKFold()`:

- `n_splits`: Recebe o número de partições, o padrão é 5.
- `shuffle`: Se `False` os dados seguirão a ordem em que aparecem no conjunto de dados, se `True` os dados serão embaralhados. Padrão é `False`.
- `random_state`: Semente de aleatoriedade para o embaralhamento. Padrão é `None`.

Logo após chamar a função `StratifiedKFold()` chamamos um método, `.split()`. Esse método é aplicado em cima do resultado do StratifiedKFold(). Ele recebe os dados da base utilizada e o rótulo para realizar a separação em k-folds. Fizemos a chamada do método desse jeito para encurtar o processo de separação em k-folds, não usaremos o resultado da função StratifiedKFold() para nada sem antes inserir os dados, assim também evitamos ter uma variável interemediária desnecessária.

O resultado da função StratifeidKFold(), assim como qualquer outra implementação de k-folds no scikit-learn é um `generator`:

```
# Tipo da variável resultado do StratifiedKFold().split()
print(type(kfSpam))
```

```
<class 'generator'>
```

Eles são um tipo especial de vetor, possuindo um comportamento diferente de uma lista, notavelmente ele só pode ser iterado uma vez que para o scikit-learn é o ideal em termos de eficiência computacional. Vamos criar uma lista a partir desse generator apenas para poder entender como o scikit-learn fez a divisão, mas usaremos sempre a variável gerada a partir da chamada do método .split() em cima do resultado da função StratifiedKFold().

```
# criação de uma lista para reutilização do generator
kfSpamLista = list(kfSpam)

# Exibindo cada k-fold
for K, (train, test) in enumerate(kfSpamLista, 1):
    print("K:", K)
    print("Treino:", train)
    print("Teste:", test)
    print(f"len(train): {len(train)}\nlen(test): {len(test)}\n\n")
    if K == 3:
        break
```

```
K: 1
Treino: [   0    1    2 ... 4598 4599 4600]
Teste: [   7   13   24  ... 4557 4558 4568]
len(train): 4140
len(test): 461

K: 2
Treino: [   0    2    3 ... 4598 4599 4600]
Teste: [   1   5   6  ... 4579 4590 4591]
len(train): 4141
len(test): 460

K: 3
Treino: [   0    1    2 ... 4596 4598 4600]
Teste: [   8   9   14  ... 4585 4597 4599]
len(train): 4141
len(test): 460
```

K, train e test são gerada a partir de kfSpamLista através da função enumerate, um jeito de lidar com a lista gerada a partir do gerador, e são prontamente iterados. O foco dessa explicação é entender como os k-folds funcionam, entender o enumerate e o generator é secundário, não iremos abordar isso futaramente porque são detalhes muito específicos do python que não é importante para entender validação cruzada.

Restringi para exibir apenas 3 k-folds para ficar melhor formatado aqui no caderno. Caso seja

do interesse de vocês visualizar o exemplo com todos os k-folds, apenas retire a condição `if K == 3:` e o `break` do loop.

No computador de vocês poderá aparecer mais elementos nas listas, formatei para mostrar os 3 primeiros e os 3 últimos apenas para fim explicativo.

Analisando o resultado inteiro com todos os k-folds chegamos a algumas observações importantes:

- a lista treino e teste de cada fold são complementares
- a união entre lista treino e teste de cada fold tem como resultado os índices da base original.
- índices na lista de teste não se repetem em outras listas de teste
- todos os índices passam uma vez pelo conjunto teste de um k-fold
- os índices para treino se repetem K-1 vezes em diferentes folds

REVISAR QUANDO USAR O NOME K-FOLD OU FOLD

REVISAR E POR NOMENCLATURA VARIAVEL RESPOSTA VS EXPLICATIVA

Repeated K-fold

O repeated k-fold se resume a repetir o método k-fold várias vezes, com o objetivo de melhorar nossa reamostragem.

Código de exemplo

Vamos aplicar um método de treino 3 vezes em 10 folds. Para isso, utilizamos a função `RepeatedStratifiedKFold()`. Ela funciona semelhante ao `StratifiedKFold()`, porém com um argumento a mais: `n_repeats`, devemos inserir aqui o número de repetições que desejamos fazer.

```
repeatedKfSpam = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=11).split(
print(len(list(repeatedKfSpam)))
```

```
30
```

Repare que o tamanho da lista gerada é justamente o número de folds vezes o número de repetições, nesse caso 30.

Observações:

- a lista treino e teste de cada fold K são complementares
- a união entre lista treino e teste de cada fold K é tem como resultado os índices da base original.
- índices na lista de teste podem se repetir em outras listas de teste
- todos os índices passam *n_repeats* vezes pelo conjunto teste de algum k-fold
- os índices para treino se repetem (K-1) * *n_repeats* vezes em diferentes k-folds

Comparando Repeated K-fold com K-fold tradicional:

Uma vantagem do repeated k-fold em relação ao k-fold é que ele permite que um elemento na lista passe mais de uma vez pelo conjunto teste, o número inserido em n_repeats.

Uma limitação do repeated k-fold em relação ao k-fold é que, até o dia 04/06/2025, a implementação no scikitlearn, a função RepeatedStratifiedKFold(), não possui o argumento `shuffle` que o StratifiedKFold() possui. Ele por dentro roda a função StratifiedKFold() com o argumento shuffle em True, logo, se por alguma razão quisermos os dados não embaralhados, não devemos usar essa função e criar uma função como essa manualmente ou usar outro método de reamostragem.

Bootstrap

O bootstrap é uma técnica de reamostragem com o propósito de reduzir desvios e realizar amostragem dos dados de treino com repetições. Consiste em realizar amostragens com reposição a partir da amostra original. Quando fazemos amostragem com reposição, estamos recolocando o valor após cada amostra. Cada amostra, portanto, é independente do valor que veio antes dela.

Quando fazemos amostragem sem reposição, não estamos recolocando os valores, uma vez que um valor é selecionado, ele não pode ser selecionado novamente, portanto não geram resultados independetes já que o valor obtido em uma amostra afeta a possibilidade dos valores da próxima amostra. Dessa forma podemos escolher um tamanho de amostras maior ou menor do que o vetor de entrada.

Um ponto negativo do Bootstrap é que ao realizar um Bootstrap com menos amostras que o conjunto de dados original, a estratificação pode ficar um pouco diferente que a original, que para um número pequeno de amostras pode ser relevante.

Código de exemplo

O scikitlearn possui uma implementação do Bootstrap quando usamos a função `resample` com `replace = True`. Segue abaixo um exemplo de uso:

```
BootstrapSpam = resample(XSpam, YSpam, n_samples= 10000, random_state=11, replace=True, st
```

- Primeiro argumento posicional recebe as variáveis explicativas, nesse caso, XSpam
- Segundo argumento posicional recebe a variável resposta, nesse caso, YSpam
- `n_samples` recebe o número de amostras que queremos.
- `random_state` recebe a semente de aleatoriedade.
- `replace` se True, permitirá reposição de amostras, se False, não. É essa variável em True que implementa o método Bootstrap. Se False e n_samples maior que número de amostras originais, um erro será gerado.
- `stratify` recebe a variável resposta ou None. Se for None, não será feito estratificação e portanto, não haverá garantia da manutenção da proporção da variável resposta.

A função gerou um vetor com dois vetores dentro: array(variaveisExplicativas, variavelResposta) para utilizarmos posteriormente. Repare que a função resample() automáticamente faz o embaralhamento dos dados.

```
print(BootstrapSpam[1])
```

```
260        spam
2562    nonspam
3228    nonspam
3233    nonspam
1994    nonspam
         ...
326        spam
2193    nonspam
3188    nonspam
254        spam
428        spam
Name: type, Length: 10000, dtype: object
```

Como Spam possui um número muito alto de amostras e o embaralhemnto é sempre feito, visualizar o Bootstrap pode não ser tão fácil. Para facilitar o entendimento, vamos usar um exemplo mais simples: supomos que temos 10 indivíduos, numerados de 1 a 10. Ao realizar 3 bootstraps do mesmo tamanho da amostra obtemos:

```
B1 = [1, 1, 1, 2, 4, 6, 6, 7, 8, 10]
B2 = [1, 1, 2, 2, 3, 5, 7, 7, 8, 9]
B3 = [3, 4, 4, 5, 6, 6, 6, 8, 8, 8]
```

Criando modelos com métodos de reamostragem.

Tunagem de hiperparâmetros

Métricas para análise de preditores

MAE

Definição

O **erro médio absoluto** é uma métrica estatística utilizada para medir a **distância entre os valores previstos e os reais** (erro), ou seja, o **erro**. Para calculá-lo, obtemos a média das diferenças absolutas entre os valores previstos e os valores reais, como mostrado na fórmula a seguir:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Sendo y_i o **valor predito** e x_i o **valor real**.

Interpretação

Como comentado anteriormente, o calculo é baseado na distância entre os dados previstos e reais, portanto, quanto **menor** o valor do MAE, melhor será o desempenho do modelo pois isso indica uma **maior proximidade** entre esses valores.

Limitações

O MAE é pouco sensível a outliers, ou seja, **erros grandes e pequenos tendem a ter a mesma influência** e isso pode ser um problema em situações onde erros grandes seriam inaceitáveis por exemplo.

MSE

Definição

O **erro médio quadrático** segue a mesma lógica do MAE, calcula a distância entre o valor predito e o real, mas utilizando o **quadrado** ao invés do módulo, o que o torna mais **sensível a outliers** e penalizando mais **erros grandes**. Ele também pode ser utilizado como **função de custo** (loss function) em modelos de regressão, como forma de **otimizar os parâmetros do modelo**, especialmente em **regressão linear** e em **redes neurais**, pois costumar ser **diferenciável**, o que facilita minimizar a função utilizando **algoritmos de otimização**, como o **gradiente descendente**, por exemplo. Sendo assim, sua fórmula é:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - x_i)^2}{n}$$

Onde y_i corresponde ao valor predito e x_i ao real.

Interpretação

Assim como o MAE, quanto **menor** o MSE de um modelo, melhor. Contudo, sua interpretação requer cuidado, pois um mesmo valor de MSE pode ser suficientemente próximo do zero ou não dado o contexto, além de **não estar na mesma unidade dos dados**.

Limitações

Como citado anteriormente, o MSE tem uma **sensibilidade maior a outliers**, o que pode aumentar seu valor e indicar que o desempenho do modelo está pior quando isso não é verdade. Além disso, ele é **dependente da escala dos dados**, de maneira que modelos que preveem faixas de valor mais altas terão naturalmente um MSE maior, tornando a comparação entre diferentes bases de dados injusta. Para contornar esse problema, podemos **normalizar ou padronizar** os dados antes de realizar o cálculo.

R^2

Definição

O R^2, também conhecido como coeficiente de determinação, é uma medida que **explica a proporção da variância** da variável resposta que pode ser explicada pelas variáveis explicativas, ou seja, o quão bem os dados se **ajustam ao modelo**. Dessa forma, ele assume valores de zero até um, onde 1 significaria que os dados se encaixaram perfeitamente. Existem diversas maneiras de se calcular seu valor, vamos abordar a mais comum nesse texto e ela se baseia em utilizar os **erros ao quadrado** e a **variância total do modelo**, de maneira que essa abordagem destaca o quão melhor o modelo de regressão funciona em comparação com simplesmente prever a média da resposta.

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum(y_i - x_i)^2}{\frac{1}{n}\sum(x_i - \bar{x})^2}$$

onde RSS (residual sum of squares) é a soma dos erros ao quadrado e TSS (total sum of squares) é a variância total da variável resposta.

Interpretação

R^2 é a porcentagem de variação explicada pela relação entre duas variáveis. R^2 é sempre uma porcentagem entre 0% e 100% com 0%, o que significa que seus valores previstos não lhe disseram absolutamente nada sobre Y e 100%, o que significa que seus valores previstos eram perfeitamente precisos. Então, para responder ao exemplo que você deu, se $R^2 = 0.8$, então diríamos: "80% da variação observada na variável Y é explicada por nossa regressão (ou então você gerou previsões para Y)". Contudo, devemos nos atentar a valores muito altos pois eles podem indicar overfitting.

Limitações

É importante lembrar que o R-quadrado não mede a **causalidade**. Só porque nossas variáveis explicativas explicam a variância da variável resposta não significa que elas sejam a causa disso. Além disso, o R-squared não indica se as previsões são precisas.

Relação entre métricas

- O MAE e o MSE tendem a diminuir juntos quando o modelo melhora
- O MSE e o R^2 estão inversamente relacionados
- $MSE \geq MAE^2$ (por desigualdade de Jensen)