sid.inpe.br/mtc-m21d/2022/05.10.16.53-PUD

APLICAÇÕES DE PYTHON EM GEOCIÊNCIAS

José Guilherme Martins dos Santos
Jonatas Leon Dias Ribeiro Simões
Diogo Nunes da Silva Ramos
Cristiano Wickboldt Eichholz

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34T/46RTMU5>

INPE
São José dos Campos
2022

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Coordenação de Ensino, Pesquisa e Extensão (COEPE)
Divisão de Biblioteca (DIBIB)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

sid.inpe.br/mtc-m21d/2022/05.10.16.53-PUD

APLICAÇÕES DE PYTHON EM GEOCIÊNCIAS

José Guilherme Martins dos Santos
Jonatas Leon Dias Ribeiro Simões
Diogo Nunes da Silva Ramos
Cristiano Wickboldt Eichholz

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34T/46RTMU5>

INPE
São José dos Campos
2022

Sumário

Agradecimentos

Agradecemos pelas informações disponibilizadas na internet de forma gratuita que possibilitam avançar no eterno aprendizado da programação, em particular, Python. Os nossos sinceros agradecimentos aos colegas que de alguma forma contribuíram de maneira direta ou indireta com sugestões ou dúvidas que motivaram a criação deste material. Agradeço ao Marcelo Guatura pelas dicas iniciais de Python e o site Stack Overflow, um site fantástico para aprender programação, além de sanar muitas dúvidas.

Sobre os autores

Guilherme Martins é Bacharel em Meteorologia pela Universidade Federal do Pará (2002), mestre em Meteorologia pela Universidade Federal de Pelotas (2005) e doutor em Ciência do Sistema Terrestre pelo Instituto Nacional de Pesquisas Espaciais (2015). Atua na área de Geociências, com ênfase em meteorologia, modelagem numérica da atmosfera (BRAMS, INLAND e MCGA), climatologia e interação biosfera-atmosfera. Possui mais de 10 anos na área de geoprocessamento e na manipulação e visualização de dados ambientais nos formatos NetCDF e GRIB (NCEP, CMIP5, CMIP6, ECMWF, CRU, entre outros). Realiza treinamentos com as ferramentas NCAR Command Language (NCL), Grid Analysis e Display System (GrADS) e Climate Data Operators (CDO) tendo capacitado mais de 200 pessoas ao longo de mais de 10 anos de experiência. É responsável por realizar capacitação para interpretar e utilizar informações meteorológicas no combate aos incêndios florestais. Atualmente, é analista em geoprocessamento no Programa Queimadas do Instituto Nacional de Pesquisas Espaciais (INPE). Realiza pesquisa e desenvolvimento neste instituto no contexto do modelo de risco de fogo para a América do Sul bem como na geração de produtos meteorológicos. Para mais informações, acessar o Currículo Lattes disponível em: http://lattes.cnpq.br/5997657584785803.

Jonatas Leon possui graduação em Análise e Desenvolvimento de Sistemas pelo Centro Estadual de Educação Tecnológica Paula Souza (2017). Tem experiência na área de Ciência da Computação, com ênfase em Metodologia e Técnicas da Computação. Além disso, é desenvolvedor de software e entusiasta da ciência de dados. Para mais informações, acessar o Currículo Lattes disponível em: http://lattes.cnpq.br/6707812894667096.

Diogo Ramos é Meteorologista com experiência em meteorologia aplicada, modelagem atmosférica, meteorologia da energia, micrometeorologia, ciência de dados e inteligência artificial aplicada a energias renováveis, como eólica e solar fotovoltaica. Atualmente é Consultor em Modelagem Computacional, *High Performance Computing*, Meteorologia da Energia e Inteligência Artificial no Campus Integrado de Manufatura e Tecnologia - SENAI CIMATEC em Salvador-BA, uma unidade da Federação das Indústrias do Estado da Bahia. Possui Doutorado em Meteorologia pelo Instituto Nacional de Pesquisas Espaciais - INPE, sendo Bacharel e Mestre em Meteorologia pela Universidade Federal de Alagoas - UFAL. Desde 2010 vem produzindo vários ativos de produção técnica, como consultorias e materiais didáticos, cursos de curta duração para estudantes de graduação e pós-graduação em ciências atmosféricas. Nos últimos anos tem se dedicado na aplicação de técnicas de ciência de dados e inteligência computacional para previsão de séries temporais meteorológicas direcionadas aos setores de indústria, geração híbrida de energias renováveis, sensoriamento remoto, defesa civil e hidrometeorologia. Recentemente tem se dedicado na elaboração e participação de projetos nacionais e internacionais relacionados com novos desafios, tais como: computação quântica aplicada à meteorologia; meteorologia no multiverso; *Digital Twins*; e *Deep Learning Weather Prediction* - DLWP. Para mais informações, acessar o Currículo Lattes disponível em: http://lattes.cnpq.br/1800868291881642.

Cristiano Eichholz é Bacharel (2008) e Mestre (2011) em Meteorologia pela Universidade Federal de Pelotas e Doutor (2017) em Meteorologia pelo Instituto Nacional de Pesquisas Espaciais, com estágio sanduíche na Texas A&M University (2015). Participou dos experimentos de campo dos projetos CHUVA-GOAmazon-ACRIDICON em Manaus em 2014 e RELAMPAGO em 2018. Concluiu Pós-Doutorado (2019) em Meteorologia pelo Instituto Nacional de Pesquisas Espaciais. Foi bolsista do Programa de Capacitação Institucional no CPTEC/INPE, desenvolvendo atividades associadas ao Projeto Aprimoramento dos processos físicos do MCGA/CPTEC para previsão de tempo e clima sazonal. Atualmente ocupa o cargo de analista GIS na empresa IACIT Soluções Tecnológicas. Tem experiência na área de Geociências, com ênfase em Meteorologia de Mesoescala e Sensoriamento Remoto da Atmosfera, atuando principalmente nos temas: Sistemas Convectivos; Tempo Severo (granizo, vendaval, casos extremos de precipitação); Previsão de

Curtíssimo Prazo (Nowcasting). Dentre os Interesses Profissionais estão: a) Convecção Atmosférica - Busca aprimorar o entendimento de processos convectivos da atmosfera que determinam a evolução de simples células de nuvens até grandes sistemas convectivos de mesoescala (geralmente associados a eventos de tempo severo como granizo, vendaval, enxurrada); b) Previsão de Curtíssimo Prazo / Nowcasting - Foco no estudo de diferentes técnicas/metodologias de previsão de evolução de eventos de tempo severo; c) Sensoriamento Remoto da Atmosfera - Uso de dados de satélite e radar meteorológico no entendimento de processos atmosféricos; d) Estatística / Análise de Dados - Busca de assinaturas e/ou padrões característicos, associados a eventos de tempo significativos, em diferentes conjuntos de dados, usando diferentes linguagens de programação (FORTRAN, R, Python, BASH-Shell Script, PHP, GrADS). Para mais informações, acessar o Currículo Lattes disponível em: http://lattes.cnpq.br/3933039769920991.

Contribuição dos autores

Guilherme Martins é o idealizador deste documento, responsável pela escrita e pelos dados, como também pelos códigos.

Jonatas Leon contribui significativamente na refatoração dos códigos e nas correções textuais.

Diogo Ramos contribuiu na revisão e correção do texto e dos scripts. Também é responsável pelo capítulo “Outras aplicações utilizando Python”.

Cristiano Eichholz contribuiu na interpolação de dados de estação.

Download dos scripts via GitHub

Todos os scripts mostrados estão disponíveis para download no seguinte endereço eletrônico:

https://github.com/jgmsantos/Livro-Python

Como executar os scripts

O link abaixo fornece instruções de como executar corretamente todos os scripts sem erro. **Leia com atenção!**

https://github.com/jgmsantos/Livro-Python#6-lista-de-bibliotecas-a-serem-instaladas

Para executar os scripts, basta digitar no terminal:

```
python <nome_do_script.py>
```

Exemplo:

```
python ex01.py
```

Arquivos necessários para executar os scripts

Os arquivos nos formatos NetCDF, GRIB, texto e shapefiles necessários para executar os scripts encontram-se no diretório chamado `dados`, disponível no link abaixo.

Os scripts podem ser executados sem alteração do caminho para os dados.

https://github.com/jgmsantos/Livro-Python/tree/main/dados

Um pouco de história sobre o Python

O holandês Guido van Rossum (nascido em 31 de janeiro de 1956) foi o criador da linguagem Python. O nome se deve ao seu programa favorito bem como dos seus colaboradores, o *Monty Python's Flying Circus* que foi uma série de humor britânica criada pelo grupo de comediantes *Monty Python* e transmitida pela BBC entre os anos 1969 e 1974. No local em que Guido trabalhava, no *Centrum Wiskunde & Informatica* (CWI), em Amsterdã, os programadores adotavam nomes de séries de televisão para as linguagens criadas.

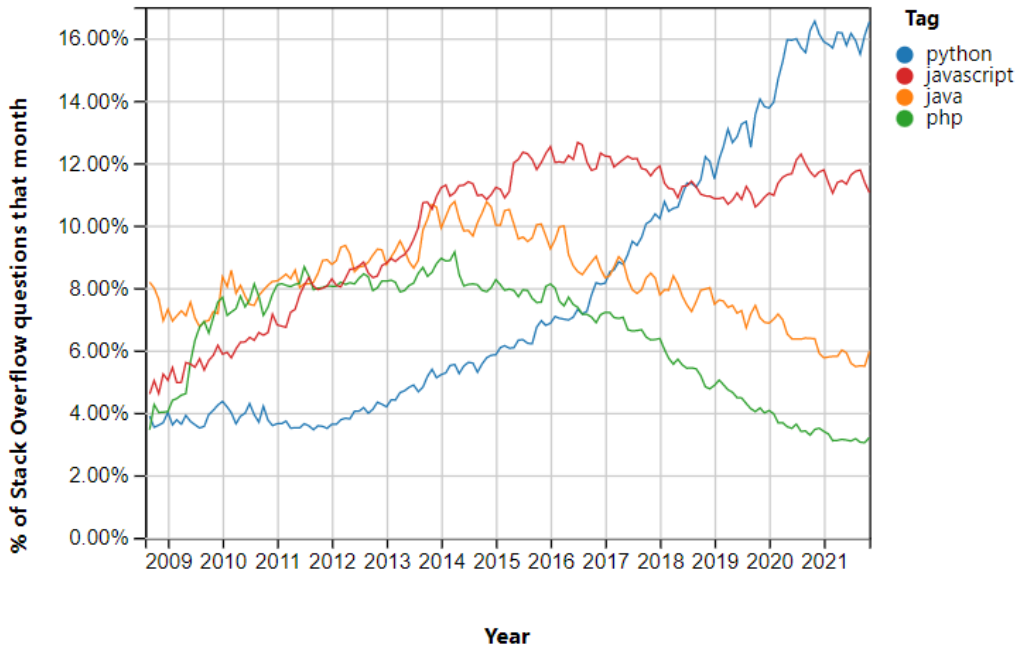
A logotipo da linguagem Python é uma serpente, Guido e companhia não ficaram felizes com essa associação que foi criada pela *O'Reilly Media* que é uma companhia de mídia americana fundada por Tim O'Reilly. Ela é responsável por publicar livros e websites, além de organizar conferências sobre assuntos relacionados à informática. Essa companhia foi responsável por lançar o primeiro livro de Python com a serpente píton na sua capa, além disso, ela tem como característica marcante adotar imagens de animais nas suas capas de livros.

O Python é uma linguagem de código aberto orientada a objeto de propósito geral, isto é, não é focada em algo, como por exemplo, PHP que se destina a programação WEB, ou seja, Python pode ser utilizado em vários setores como medicina, física, matemática, geografia, *data science, big data, machine learning*, dentre outras.

A filosofia do Python consiste em que ela seja simples, fácil e intuitiva. Outra característica está no fato de ser multiplataforma, ou seja, o mesmo programa pode ser executado em diferentes sistemas operacionais como o Windows, o Linux e o MacOS.

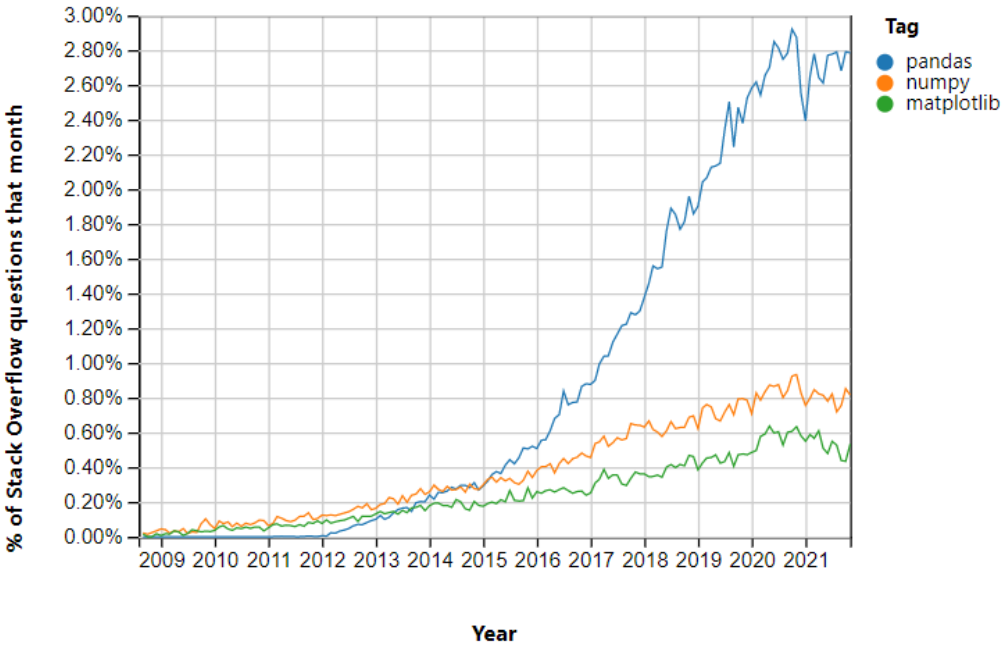
Quem utiliza Python? Alguns exemplos: a Microsoft, o YouTube, a NASA, o GIMP (editor de imagens), e o Google. Porém o único sistema operacional que não tem o Python pré-instalado, é o Windows da Microsoft, os demais, Linux e MacOS, já vem pré-instalado.

A figura abaixo (dados compilados até dezembro de 2021) mostra a tendência do uso da linguagem Python desde 2009 até 2021. Nota-se a crescente utilização dessa linguagem em comparação com as demais (javascript, java e php).



Fonte: https://insights.stackoverflow.com/trends?tags=python%2Cjava%2Cphp%2Cjavascript

E dentro do universo Python (figura abaixo, dados compilados até dezembro de 2021), uma biblioteca tem chamado a atenção, o Pandas, como mostra a figura abaixo. Isso demonstra que a utilização dela para Ciência de Dados tem grande importância. Além dela, as bibliotecas *NumPy* e *Matplotlib* também têm mostrado um crescimento considerável.



Fonte: https://insights.stackoverflow.com/trends?tags=pandas%2Cnumpy%2Cmatplotlib

A necessidade de se criar documentos em língua portuguesa está no fato de que a maioria dos livros de programação são escritos na língua inglesa. Por isso, este material surgiu com o objetivo de somar aos já existentes, com a diferença de ter um foco mais direcionado para geociências.

Os livros de programação sempre apresentam as mesmas informações sobre o básico de codificação, e aqui não será diferente na sua parte introdutória. A diferença encontra-se nos capítulos em que são utilizados dados reais para o seu processamento e sua visualização (temperatura, vento, chuva, umidade relativa, dentre outros) a partir das informações iniciais de codificação apresentada. Portanto, espera-se que esse livro possa auxiliar os usuários a terem uma noção mais aplicada de Python em geociências a partir de exemplos práticos e scripts comentados para melhor entendimento.

Antes de iniciar os estudos em Python, é recomendado que se leia as informações contidas no link abaixo para se ter uma visão geral sobre esta linguagem fascinante.

https://docs.python.org/3.9/tutorial/index.html

Instalação do ambiente virtual

A criação de ambientes virtuais é aconselhada para trabalhar com diferentes versões do Python e suas bibliotecas. Além disso, é interessante porque não danifica o seu sistema operacional por quaisquer problemas de incompatibilidade. Desta forma, cria-se um ambiente de trabalho independente para instalar versões distintas para efeito de teste sendo que o ambiente original continua preservado. Uma vez que o ambiente criado apresente instabilidade, pode-se removê-lo e criar uma nova instalação limpa, sem danificar o sistema operacional.

Neste livro será utilizada a versão miniconda que representa uma distribuição mais leve e o sistema operacional Linux a partir da sua distribuição Ubuntu 18.04.

Vale ressaltar que o miniconda é uma versão da plataforma Anaconda, e também está disponível em outros sistemas operacionais, como Windows e MacOS. Além disso, existem outros métodos para criação de um ambiente virtual no Python, como o VirtualEnv, que é ainda mais leve e rápido que o miniconda. A diferença, e talvez uma vantagem, é que o VirtualEnv faz uso do pip como gerenciador de pacotes para instalação, atualização, remoção e etc.

O miniconda também é compatível com o pip, porém possui acesso aos canais oficiais do Anaconda, com um repositório que detém milhares de pacotes. A desvantagem destes canais é que se o ambiente virtual começar a ter vários pacotes instalados, a instalação de um novo ou a atualização de outro, por resultar em um longo tempo de verificação de incompatibilidade com quaisquer outros pacotes já instalados.

Para instalar, basta realizar o download do arquivo do miniconda utilizando o comando abaixo:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

Após realizar o download, basta digitar o comando abaixo para realizar a instalação. O “chmod +x” torna o arquivo executável.

```
chmod +x Miniconda3-latest-Linux-x86_64.sh
```

```
./Miniconda3-latest-Linux-x86_64.sh
```

Basta seguir os passos concordando com os termos.

Uma vez a instalação concluída, deve-se reiniciar o terminal para que a instalação seja finalizada corretamente.

A seguir, são mostrados alguns comandos utilizados para criar, desativar, ativar e remover um ambiente.

Criar um novo ambiente

```
conda create --name <nome_ambiente>
```

Exemplo de criação do ambiente virtual chamado “trabalho”. O nome fica a critério do usuário.

```
conda create --name trabalho
```

Por exemplo, caso o usuário queira criar um ambiente virtual com o Python 2.7, basta utilizar o comando abaixo:

```
conda create -n trabalho python=2.7
```

Ativar um ambiente

```
conda activate <nome_ambiente>
```

Acessando o ambiente virtual “trabalho”.

```
conda activate trabalho
```

```
(trabalho) guilherme@DESKTOP-LD7TCRV:~$
```

Deve-se observar que na frente do nome do usuário (linha acima destacada em negrito) aparecerá o nome do ambiente que está sendo acessado, nesse caso, o “trabalho”.

Desativar um ambiente

```
conda deactivate
```

Ao digitar o comando acima aparecerá o nome “base” (linha abaixo destacada em negrito) que representa o ambiente virtual original.

```
(base) guilherme@DESKTOP-LD7TCRV:~$
```

Remover um ambiente

Caso o ambiente esteja ativo, é necessário desativá-lo com o comando abaixo antes de removê-lo.

```
conda deactivate
```

Para remover o ambiente de interesse, basta digitar o comando abaixo.

```
conda env remove --name <nome_do_ambiente>
```

Exemplo: Remover o ambiente “trabalho”.

```
conda env remove --name trabalho
```

Listar os ambientes virtuais

Para saber o nome dos ambientes virtuais disponíveis, basta digitar o comando abaixo.

```
conda env list
```

Listar o nome dos pacotes instalados em um ambiente

```
conda list -n <nome_do_ambiente>
```

O comando abaixo lista o nome de todos os pacotes instalados no ambiente virtual “inpe”.

```
conda list -n inpe
```

Para listar a versão de uma biblioteca em particular.

```
conda list -n inpe <nome_da_biblioteca>
```

Exemplo: Saber a versão do Python que está instalada.

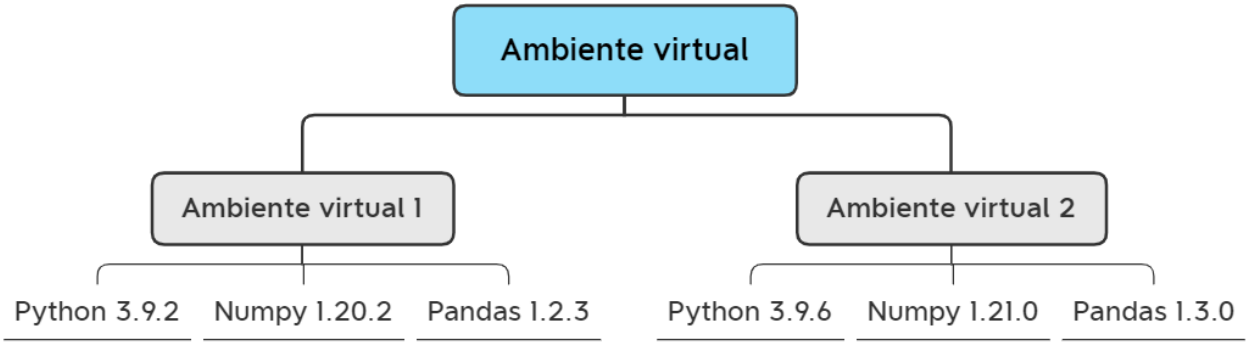
```
conda list -n inpe python
```

Após a instalação, cria-se o ambiente virtual com um nome qualquer definido pelo usuário do sistema operacional, e com isso, pode-se trabalhar normalmente neste novo ambiente. Caso ele

passe por algum tipo de instabilidade, basta removê-lo, e criar um novo seguindo os passos acima.

A criação de ambiente virtual é muito útil quando se deseja testar alguma biblioteca ou várias, sem danificar o sistema operacional.

A figura abaixo mostra um resumo do que se entende por ambiente virtual. Supondo que o Ambiente virtual 1 (nome fictício) tem o Python 3.9.2 instalado e várias bibliotecas (Numpy, Pandas, etc). Por outro lado, o Ambiente virtual 2 tem o Python 3.9.6 e bibliotecas diferentes do primeiro ambiente virtual. Essa é a representação de um ambiente virtual.



Fonte: Produzido pelo autor.

O que devo saber antes

Series nada mais é que um array (arranjo) de 1 dimensão. Você pode considerar um Series também como uma coluna de uma tabela.

O DataFrame é simplesmente um conjunto de Series. É uma estrutura de dados representada por duas dimensões, isto é, linhas e colunas que pode ser entendida como uma tabela, semelhante ao Excel.

Informações sobre algumas bibliotecas mais utilizadas em Python

O pip é o instalador de bibliotecas do Python.

Para instalar um pacote:

```
pip install <pacote>
```

Para desinstalar um pacote:

```
pip uninstall <pacote>
```

Exemplo: Desinstalar o matplotlib.

```
pip uninstall matplotlib
```

Pandas

É uma biblioteca de código fonte aberto escrita sobre o Numpy. Com ela, é possível fazer visualizações rápidas bem como a limpeza de dados. É muito semelhante ao Excel (dados armazenados em Data Frames, linhas e colunas) e possui a característica de trabalhar com dados de diferentes formatos (strings, números e etc). Além disso, possui um método próprio de visualização de dados.

Instalação:

Via pip: `pip install pandas`

Via conda: conda install pandas

No script, a importação dessa biblioteca é feita da seguinte forma:

```
import pandas as pd
```

Referência: https://pandas.pydata.org

Matplotlib

É a biblioteca de visualização de dados mais popular do Python. Ela permite o controle sobre todos os aspectos da figura.

Instalação:

Via pip: `pip install matplotlib`

Via conda: `conda install matplotlib`

No script, a importação dessa biblioteca é feita da seguinte forma:

```
import matplotlib.pyplot as plt
```

Referência: https://matplotlib.org/stable/contents.html

Cartopy

É um pacote para processamento de dados geoespaciais com o objetivo de produzir mapas e outras análises geoespaciais.

Instalação:

Via pip: `pip install cartopy`

Via conda: `conda install cartopy`

Referência: https://pypi.org/project/Cartopy

NetCDF4

Útil para manipular (leitura e escrita) arquivos no formato NetCDF4.

Instalação:

Via pip: `pip install netcdf4`

Via conda: `conda install netcdf4`

No script, a importação dessa biblioteca é feita da seguinte forma:

```
from netCDF4 import Dataset as nc
```

Referência: https://pypi.org/project/netCDF4

Xarray

É um pacote Python de código aberto que visa tornar o trabalho com arranjos de dados multidimensionais (exemplo, NetCDF) uma tarefa mais simples e eficiente.

Instalação:

Via pip: `pip install xarray`

Via conda: `conda install xarray`

Para uma instalação completa com suporte a diferentes formatos como: GRIB, NetCDF, HDF, projeções, dentre outros, utiliza-se o comando:

```
pip install xarray[complete]
```

No script, a importação dessa biblioteca é feita da seguinte forma:

```
import xarray as xr
```

Referência:

https://pypi.org/project/xarray

https://xarray.pydata.org/en/stable/getting-started-guide/installing.html

Windrose

Biblioteca para criação da rosa dos ventos a partir da sua direção e velocidade.

Instalação:

Via pip: `pip install windrose`

Referência: https://pypi.org/project/windrose

GeoPandas

O GeoPandas é uma extensão do Pandas que facilita a manipulação de dados espaciais e georreferenciados.

Instalação:

Via pip: `pip install geopandas`

Via conda: `conda install geopandas`

No script, a importação dessa biblioteca é feita da seguinte forma:

```
import geopandas
```

Referência: https://pypi.org/project/geopandas

SkillMetrics

Este pacote contém uma coleção de funções para calcular a habilidade das previsões do modelo em relação às observações. Os cálculos incluem: bias (Mean error), BS (Brier score), BSS (Brier skill score), r (Correlation coefficient), CRMSD (centered root-mean-square error deviation), NSE (Nash-Sutcliffe efficiency), RMSD (root-mean-square error deviation), SS (Murphy's skill score) e SDEV (standard deviation). Além disso, é possível gerar os diagramas de Taylor e Target bem como realizar a sua personalização.

Instalação:

Via pip: `pip install xlsxwriter`

Via pip: `pip install SkillMetrics`

Para atualizar o pacote: `pip install SkillMetrics --upgrade`

No script, a importação dessa biblioteca é feita da seguinte forma:

```
import skill_metrics as sm
```

Referência: https://pypi.org/project/SkillMetrics

Scipy

Essa biblioteca foi feita para matemáticos, cientistas e engenheiros e foi escrita sobre NumPy.

Instalação:

Via pip: `pip install scipy`

Via pip: `conda install -c anaconda scipy`

No script, a importação dessa biblioteca é feita da seguinte forma:

```
from scipy import stats
```

Referência: https://docs.scipy.org/doc/scipy/reference/

NumPY

Essa biblioteca é muito eficiente para computação científica em Python, pois permite manipular matrizes de forma eficiente. Além disso, é possível realizar cálculos matemáticos, lógica, álgebra linear básica, operações estatísticas básicas, dentre outras opções.

Instalação:

Via pip: `pip install numpy`

Via conda: `conda install -c anaconda numpy`

No script, a importação dessa biblioteca é feita da seguinte forma:

```
import numpy as np
```

Referência: https://numpy.org/doc/stable/user/whatisnumpy.html

O que saber sobre uma excelente codificação

A programação como um idioma tem o seu estilo de escrita, isso facilita o entendimento do programa bem como a sua portabilidade para outros sistemas operacionais. Portanto, é interessante conhecer um pouco sobre o PEP (*Python Enhancement Proposals*) indicado no link abaixo.

https://www.python.org/dev/peps

Uma excelente prática, é ler sobre as PEP8. Essa PEP (link abaixo) representa um guia de estilo de codificação em Python, ou seja, como escrever um programa de forma adequada.

https://www.python.org/dev/peps/pep-0008

Exemplos de PEP8:

Utilize sempre 4 espaços para indentação em vez de TAB. O TAB pode ter configurações diferentes em computadores distintos.

Utilize sempre letras minúsculas separadas por "_" para funções ou variáveis.

Utilize sempre duas linhas em branco para separar funções e definições de classe com duas linhas em branco.

Métodos dentro de uma classe devem ser separados por uma única linha em branco.

`Imports` devem ser feitos em linhas separadas e são sempre declarados no topo do script.

O link abaixo mostra uma excelente ferramenta para checagem de PEP8.

http://pep8online.com/checkresult

O básico de programação

Leitura recomendada:

Tutorial de Python:

https://docs.python.org/3.9/tutorial/index.html

Informações sobre programação:

https://docs.python.org/3.9/tutorial/controlflow.html

Loop for

É uma estrutura de repetição. Utiliza-se *loops* para iterar sobre sequências ou sobre valores iteráveis.

Exemplo 1: Imprime cada uma das letras da palavra Meteorologia.

```
nome = "Meteorologia"

for letra in nome:
    print(letra)
```

Exemplo 2: Imprime os valores de 1 a 9, o 10 é exclusivo (não é considerado), isso é uma característica do Python.

```
for numero in range(1, 10):
    print(numero)
```

Loop while

O bloco do while será repetido enquanto a expressão booleana for verdadeira. Expressão booleana é toda expressão em que o resultado é True (verdadeiro) ou False (falso). Em um loop while é importante cuidar do critério de parada para não causar um loop infinito.

Exemplo 1: Imprime os valores de 1 a 9.

```
numero = 1

while numero < 10:
    print(numero)
    numero = numero + 1
```

Exemplo 2: Enquanto o usuário não digitar “sim”, o loop será executado.

```
resposta = ''
```

```
while resposta != 'sim':
    resposta = input("Já acabou Jéssica?")
```

Condicional

Na estrutura condicional simples, uma condição é testada, caso o valor seja verdadeiro, um conjunto de ações são tomadas. No caso da condicional composta, quando a condição for falsa, o programa executará uma ou várias instruções.

Exemplo 1: Estrutura *if*.

```
temperatura = 30

if temperatura < 40:
    print("Temperatura menor que 40 graus Celsius")
```

Exemplo 2: Estrutura *if-else*.

```
temperatura = 45

if temperatura < 40:
    print("Temperatura menor que 40 graus Celsius")
else:
    print("Temperatura maior que 40 graus Celsius")
```

Exemplo 3: Estrutura *if-elif-else*. É possível utilizar vários *elif* que dependerá da condição desejada.

```
temperatura = 60

if temperatura < 30:
    print("Temperatura menor que 30 graus Celsius")
elif temperatura == 50:
    print("Temperatura igual a 50 graus Celsius")
else:
    print("Temperatura maior que 30 graus Celsius")
```

break

É utilizado para sair de loops de maneira projetada.

Exemplo 1:

```
for numero in range(1, 11):
    if numero == 6:
        break
    else:
        print(numero)

print('Sai do loop') # O print está fora do bloco for.
```

Exemplo 2:

```
while True:
    comando = input("Digite 'sair' para finalizar o programa.")
    if comando == 'sair':
        break
```

Estruturas lógicas

Estruturas lógicas: and (e), or (ou) e not (não).

A Tabela abaixo exemplifica o uso do operador “and”.

O operador “and” resulta em um valor True se os dois valores de entrada da operação forem True, caso contrário, o resultado é False. É também conhecido como operador binário.

Valor 1	**Valor 2**	**Operação and**
True	True	True
True	False	False
False	True	False
False	False	False

Exemplo:

```
umidade_relativa = 80
temperatura = 20

if umidade_relativa <= 30 and temperatura >= 40:
    print("Condição perigosa")
else:
    print("Condição favorável")
```

A Tabela abaixo exemplifica o uso do operador “or”.

O operador “or” resulta em um valor True se ao menos um dos dois valores de entrada da operação for True, caso contrário, o resultado é False. É também conhecido como operador binário.

Valor 1	**Valor 2**	**Operação or**
True	True	True
True	False	True
False	True	True
False	False	False

Exemplo:

```
umidade_relativa = 80
temperatura = 45

if umidade_relativa <= 30 or temperatura >= 40:
    print("Condição perigosa")
else:
    print("Condição favorável")
```

A Tabela abaixo exemplifica o uso do operador “not”.

O operador “not” é o único operador que recebe como entrada apenas um valor (operador unário). Sua função consiste em inverter os valores, ou seja, se o valor de entrada for True, o resultado será False, e se for False, o resultado será True.

Valor de entrada	**Operação not**
True	False
False	True

Exemplo:

```
umidade_relativa = 80

if not umidade_relativa <= 30:
    print("Umidade relativa alta")
else:
    print("Umidade relativa baixa")
```

Lista

Listas em Python funcionam como vetores/matrizes (arrays), com a diferença de serem dinâmicos com a possibilidade de inserir nela qualquer tipo de dado.

- As listas em Python são representadas por [].
- Dinâmico: Não possui tamanho fixo, ou seja, pode-se criar a lista e adicionar/remover elementos a ela;
- Qualquer tipo de dado: Não possuem tipo de dado fixo, ou seja, pode-se incluir qualquer tipo de dado.
- Listas são mutáveis: elas podem mudar constantemente.

Leitura recomendada:

Sobre listas:

https://docs.python.org/3.9/tutorial/introduction.html#lists

https://docs.python.org/pt-br/3.9/tutorial/datastructures.html

Operações com listas:

https://docs.python.org/3.9/tutorial/datastructures.html

Exemplos de listas:

```
lista1 = [1, 99, 4, 27, 15]  # Lista de inteiros.
lista2 = ['M', 'e', 't', 'e', 'o', 'r', 'o', 'l', 'o', 'g', 'i', 'a']  # Lista de strings.
lista3 = []  # Lista vazia.
lista4 = list(range(11))  # Cria uma lista com 10 elementos (0 a 10) do tipo inteiro.
lista5 = list('Meteorologia')  # ['M', 'e', 't', 'e', 'o', 'r', 'o', 'l', 'o', 'g', 'i', 'a']
```

Ordenar uma lista

```
x = [1, 30, 5, 8, 0]
x.sort()
print(x)  # [0, 1, 5, 8, 30]
```

Contar o número de elementos que se repete na lista

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]
print(x.count(5))  # 3
```

Adicionar um elemento a lista

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0]

x.append(100)  # Inclui um elemento no fim da lista.

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0, 100]
```

Adicionar mais de um elemento a lista

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0]

x.extend([100, 120, -5])  # Inclui mais de um elemento no fim da lista.

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0, 100, 120, -5]
```

Inserir um elemento em uma dada posição na lista

O novo valor (-999) a ser inserido não substitui o existente na lista. O valor original da posição 4 (valor 0) foi deslocado para a direita.

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]
#    0   1  2  3  4   5  6  7  8  <= Posição original.

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0]

x.insert(4, -999)  # Adiciona na posição 4 o valor -999.

print(x)  # [1, 30, 5, 8, -999, 0, 30, 5, 5, 0]
```

Unir duas listas

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]
y = [-999, 7, -10]

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0]

print(y)  # [-999, 7, -10]

x.extend(y)  # Junta as duas listas.

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0, -999, 7, -10]
```

Inverter a ordem de uma lista

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0]

x.reverse()  # Inverte a ordem da lista.

print(x)  # [0, 5, 5, 30, 0, 8, 5, 30, 1]
```

Remover o último elemento da lista

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0]

x.pop()  # Remove o último elemento da lista.

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5]
```

Remover um elemento da lista pelo seu índice

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]
#    0   1  2  3  4   5  6  7  8  <= Posições.

print(x)  # [1, 30, 5, 8, 0, 30, 5, 5, 0]

x.pop(2)  # Remove o elemento que está no índice 2, ou seja, o valor 5.

print(x)  # [1, 30, 8, 0, 30, 5, 5, 0]
```

Iterando em uma lista

```
x = [1, 30, 5, 8, 0, 30, 5, 5, 0]

for numeros in x:
    print(numeros)
```

Indexando em uma lista

Entende-se o índice negativo como uma roda, ou seja, antes do valor 1 (índice 0) qual seria o índice antes dele? É o -1.

```
x = [1, 30, 5, 8, 2]

print(x[0])  # 1
print(x[1])  # 30
print(x[2])  # 5
print(x[3])  # 8
print(x[4])  # 2

# Índice negativo.

print(x[-1]) # 2
print(x[-2]) # 8
print(x[-3]) # 5
print(x[-4]) # 30
print(x[-5]) # 1
```

Outros exemplos.

```
x = [1, 30, 5, 8, 2]
#    0   1  2  3  4  <= Posições.

print(x[1:])  # [30, 5, 8, 2]

print(x[::])  # [1, 30, 5, 8, 2] => Todos os elementos da lista.

print(x[:2])  # [1, 30] => Lembrando que o índice 2 não é incluído.

print(x[1:3])  # [30, 5] => Lembrando que o índice 3 não é incluído.

print(x[1::2])  # [30, 8] => Vai do índice 1 até o final com passo 2.

print(x[::2])  # [1, 5, 2] => Vai do índice 0 até o final com passo 2.

print(x[::-2])  # [2, 5, 1] => Vai do índice 0 até o final com passo -2.
```

Lista aninhadas

Em listas aninhadas o acesso é feito via lista[linha][coluna].

```
x = [[1, 30, 5], [1, 2, 3], [10, 20, 30]]  # Matriz 3 x 3.

print(x[0])  # [1, 30, 5]

print(x[1])  # [1, 2, 3]

print(x[2])  # [10, 20, 30]

print(x[0][2])  # Acessando a lista no índice 0 ([1, 30, 5]) e acessando o índice
2 que representa o valor 5.
```

Tuplas

Tuplas são bastante parecidas com listas e são utilizadas sempre que não há necessidade de modificar os dados contidos em uma coleção. E qual a razão de utilizar tuplas? Tuplas são mais rápidas do que listas e deixam o seu código mais seguro.

Sobre as tuplas:

- As tuplas são representadas por parênteses ().
- As tuplas são imutáveis. Isso significa que ao se criar uma tupla ela não muda. Toda operação em uma tupla gera uma nova tupla.
- A indexação em tuplas é a mesma feita em listas.

Leitura recomendada:

https://docs.python.org/3.9/tutorial/datastructures.html#tuples-and-sequences

Exemplos de tuplas:

```
tupla1 = (1, 2, 3)
print(type(tupla1))
```

É o mesmo que:

```
tupla1 = 1, 2, 3
```

Observação: Tuplas com 1 elemento.

```
tupla1 = (4)  # Isso não é uma tupla.
print(tupla1)
print(type(tupla1))

tupla2 = (4,)  # Isso é uma tupla.
print(tupla2)
print(type(tupla2))

tupla3 = 4,  # Isso é uma tupla.
print(tupla3)
print(type(tupla3))
```

Conclusão: As tuplas são definidas pela vírgula e não pelo uso do parênteses.

Pode-se gerar uma tupla dinamicamente por meio do:

range(início, fim, passo)

Exemplo:

```
tupla = tuple(range(4))
print(tupla)  # (0, 1, 2, 3)
print(type(tupla))
```

Desempacotamento de tupla:

```
tupla = ('Meteorologia', 'Previsão de tempo')
curso, funcao = tupla
print(curso)  # Meteorologia
print(funcao)  # Previsão de tempo
```

Dicionário

Em algumas linguagens de programação, os dicionários Python são conhecidos por mapas e são coleções do tipo chave e valor. Eles são representados por chaves {}.

Sobre dicionários:

- Chave e valor são separados por “:”, exemplo: 'chave:valor';
- Tanto chave quanto valor podem ser de qualquer tipo de dado;
- Pode-se misturar diferentes tipos de dados.

Leitura recomendada:

https://docs.python.org/3.9/tutorial/datastructures.html#dictionaries

Exemplos:

```
variaveis = {'Temp': '30', 'UR': '80', 'Prec': '10'}
print(variaveis)
print(type(variaveis))  # <class 'dict'>
```

Exemplo: Acessando as informações utilizando a chave.

Forma 1:

```
variaveis = {'Temp': '30', 'UR': '80', 'Prec': '10'}
print(variaveis)
print(variaveis['Temp'])
print(variaveis['VV'])  # KeyError: 'VV'
```

Acesso via get:

Forma 2: Acessando via get (Forma Recomendada).

Caso o *get* não encontre o objeto com a chave informada será retornado o valor None e será gerado o erro *KeyError*.

```
print(variaveis.get('UR'))
print(variaveis.get('VV'))  # None
```

Exemplo: Pode-se definir um valor padrão para o caso de não encontrar o objeto com a chave informada.

```
variaveis = {'Temp': '30', 'UR': '80', 'Prec': '10'}
nome = variaveis.get('VV', 'Não encontrado')  # Procurar pelo 'VV', e caso não
encontre, retorna 'Não encontrado'
print(f'Valor da variável {nome}')
```

List comprehension

Utilizando List Comprehension pode-se gerar novas listas com dados processados a partir de outro iterável.

Leitura recomendada:

https://docs.python.org/3.9/tutorial/datastructures.html#list-comprehensions

Sintaxe de List Comprehension:

```
[dado for dado in iteravel]
```

Exemplo:

```
numeros = [1, 2, 3, 4, 5]  # Lista de números.
res = [numero * 10 for numero in numeros]
print(res)  # [10, 20, 30, 40, 50]
```

Para entender melhor o que está acontecendo, divide-se a expressão em duas partes:

1. A primeira parte: `for numero in numeros`
2. A segunda parte: `numero * 10`

Exemplo usando função:

```
numeros = [1, 2, 3, 4, 5]  # Lista de valores.

def funcao(valor):
    return valor * valor

res = [funcao(numero) for numero in numeros]
print(res)  # [1, 4, 9, 16, 25]
```

Exemplo utilizando loop:

```
numeros_dobrados = []  # Lista vazia para armazenar os valores.

for numero in [1, 2, 3, 4, 5]:
    numeros_dobrados.append(numero * 2)  # Armazena o resultado na lista que
antes estava vazia.

print(numeros_dobrados)  # [2, 4, 6, 8, 10]
```

Outro exemplo:

```
numeros = [1, 2, 3, 4, 5, 6]  # Lista de números.

res = [numero * 2 if numero%2 == 0 else numero/2 for numero in numeros]
print(res)  # [0.5, 4, 1.5, 8, 2.5, 12]
```

Como usar o groupby

Leitura recomendada:

https://pandas.pydata.org/docs/user_guide/groupby.html
https://pandas.pydata.org/pandas-docs/stable/reference/groupby.html
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html

Groupby por ano

```
import pandas as pd
import numpy as np

# Dado mensal de precipitação desde jan/1979 a dez/2005 = 324 meses.
# São utilizados 27 anos.
df = pd.read_csv('pr.GPCP.txt', sep='\t', names=['Ano', 'Mes', 'Prec'])

print(df)

x = df.groupby('Ano')  # Agrupa por ano.

# Calcula o acumulado anual para cada um dos anos.
acumulado_anual = x['Prec'].agg(np.sum)

print(acumulado_anual)
```

Resultado do print(df):

```
      Ano  Mes  Prec
0    1979    1   258
1    1979    2   307
2    1979    3   322
3    1979    4   254
4    1979    5   187
..    ...  ...   ...
319  2005    8    63
320  2005    9    79
321  2005   10   141
322  2005   11   159
323  2005   12   321
```

Resultado do print(acumulado_anual):

```
Ano
1979    2267
1980    2212
1981    2164
1982    2365
1983    1909
.
.
.
1999    2349
2000    2461
2001    2250
2002    2215
2003    2159
2004    2224
2005    2255
```

Groupby por mês

```
import pandas as pd
import numpy as np

# Dado mensal de precipitação desde jan/1979 a dez/2005 = 324 meses.
# São utilizados 27 anos.
df = pd.read_csv('pr.GPCP.txt', sep='\t', names=['Ano', 'Mes', 'Prec'])

x = df.groupby('Mes')  # Agrupa por mês.

# Calcula a média mensal (climatologia). O cálculo considera a média de
# todos os jan, fev, ..., dez. O resultado final será um arquivo com 12 meses.
media_mensal = x['Prec'].agg(np.mean)

# Conta o total de meses para cada mês.
total_meses = x['Prec'].agg(np.size)

print(media_mensal)

print(total_meses)
```

Resultado do print(media_mensal):

```
Mes
1     270.703704
2     297.444444
3     296.703704
4     262.444444
5     195.259259
6     123.592593
7      89.666667
8      81.111111
9     104.111111
10    141.370370
11    178.444444
12    226.222222
Name: Prec, dtype: float64
```

Resultado do print(total_meses):

```
Mes
1     27
2     27
3     27
4     27
5     27
6     27
7     27
8     27
9     27
10    27
11    27
12    27
Name: Prec, dtype: int64
```

Groupby por diferentes agregações

```
import pandas as pd
import numpy as np

# Dado mensal de precipitação.
df = pd.read_csv('pr.GPCP.txt', sep='\t', names=['Ano', 'Mes', 'Prec'])

estatistica = df.groupby('Mes').agg([np.size, np.sum, np.mean, np.std])

print(estatistica)
```

Resultado do print(estatistica):

```
     size   sum        mean        std
Mes
1      27  7309  270.703704  39.653126
2      27  8031  297.444444  25.717898
3      27  8011  296.703704  26.921274
4      27  7086  262.444444  27.691061
5      27  5272  195.259259  33.690438
6      27  3337  123.592593  24.703253
7      27  2421   89.666667  16.172151
8      27  2190   81.111111  13.956893
9      27  2811  104.111111  17.915042
10     27  3817  141.370370  21.224348
11     27  4818  178.444444  22.815874
12     27  6108  226.222222  35.262023
```

Como usar o drop_duplicates para remover linhas duplicadas

Leitura recomendada:

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop_duplicates.html

```
import pandas as pd
import numpy as np

# Dado de direção do vento no formato string: N, S, E, W.
df = pd.read_csv('direcao_vento.txt')

# Remove as linhas duplicadas e retorna apenas o índice e a direção do vento
# da primeira ocorrência.
# Exemplo: NNE apresenta 3 repetições: índice 49, 84 e 205 do arquivo acima,
porém o resultado será: 48 NNE, ou seja, retorna o primeiro índice da série
# repetida, isto é, o 48.
# O subset representa as colunas que terão as linhas excluídas.
direcao_duplicada = df.drop_duplicates(subset = 'Direcao')

print(direcao_duplicada)
```

Resultado do print(direcao_duplicada):

```
   Direção
0        N
1      ENE
3        E
13      NE
48     NNE
```

Como usar o duplicated para identificar linhas duplicadas

Leitura recomendada:

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.duplicated.html

```
import pandas as pd

direcao = pd.DataFrame(['N', 'ENE', 'ENE', 'E', 'E', 'E', 'E', 'ENE', 'ENE'])
#                        0     1      2     3    4    5    6     7      8

# Verifica as linhas que são duplicadas por meio do seu índice.
direcao_duplicada = direcao.duplicated()

# Remove as linhas duplicadas.
duplicada = direcao.drop_duplicates()

print(direcao_duplicada)

print(duplicada)
```

Resultado do print(direcao_duplicada):

```
0    False
1    False
2     True
3    False
4     True
5     True
6     True
7     True
8     True
dtype: bool
```

Resultado do print(duplicada):

```
     0
0    N
1  ENE
3    E
```

Como criar funções

O que são funções

As funções são pequenos trechos de código que realizam tarefas específicas.

A forma genérica de uma função é:

```
def nome_da_funcao(parametros_de_entrada):
    bloco_da_funcao
```

Onde:

nome_da_função: Sempre utilizar letras minúsculas, e se for nome composto, utilizar *underline* (“_”).

parametros_de_entrada: São opcionais. Tendo mais de um, separar por vírgula.

bloco_da_funcao: Onde o processamento da função ocorre. Neste bloco, pode ter ou não retorno da função.

A palavra reservada para função é def.

É interessante notar que dentro de uma função, pode-se chamar outra função.

Leitura recomendada:

https://docs.python.org/3.9/tutorial/controlflow.html#defining-functions

Criando a primeira função

A função egua() retorna uma frase.

```
# Definição da função.
def egua():
    print('Egua, mano! O Python é muito doido!')

# Chamada de execução.
egua()  # Egua, mano! O Python é muito doido!
```

Observações sobre essa função:

- Dentro de uma função (egua()), pode-se chamar outra função (print()).
- A função criada executa apenas uma tarefa, ou seja, ela imprime uma frase.
- A função não recebe nenhum parâmetro de entrada.
- A função não retorna nenhuma informação.
- Ao chamar a função, utiliza-se o parênteses sem espaço no nome da função, isto é, egua().

Funções com retorno

Em Python, quando uma função não retorna nenhum valor, o valor retornado será None.

```
# Definição da função.
def soma():
    print(2+2)

x = soma()

print(f'Resultado: {x}')  # Resultado: None
```

É feita a refatoração (reescrever) da função para que ela retorne um resultado. Para isso, basta adicionar a palavra reservada return (finaliza a execução da função) no lugar do print, ou seja, será retornado o valor da função e não a sua impressão na tela.

```
def soma():
    return 2+2

x = soma()

print(f'Resultado: {x}')  # Resultado: 4
```

Não necessariamente, precisa-se criar uma variável para receber o valor da função. Pode-se passar a sua execução para outras funções. O exemplo acima ficaria da seguinte forma:

```
def soma():
    return 2+2

print(f'Resultado: {soma()}')  # Resultado: 4
```

Funções com parâmetro(s)

São funções que recebem dados para serem processados dentro dela. Na função abaixo é obrigatório fornecer o(s) parâmetro(s), caso contrário, será retornado erro (`TypeError`).

```
def kelvin_to_celsius(tk):
    return tk - 273.17


# Forma 1: Executando por meio do print.
print(kelvin_to_celsius(300))  # 26.829999999999984

# Forma 2: Atribuindo a função a uma variável (tc).
tc = kelvin_to_celsius(300)
print(tc)  # 26.829999999999984
```

Outro exemplo. Dois parâmetros são fornecidos para realizar o cálculo da umidade relativa (%).

```
def calcula_umidade_relativa(e, es):
    ur = (e/es) * 100
    return ur


rh = calcula_umidade_relativa(209, 383)
print(f'O valor da UR é: {rh}%')  # O valor da UR é: 54.56919060052219%
```

Funções com parâmetro padrão

São funções com passagem de parâmetros opcionais. Além disso, evita erros por meio da introdução de parâmetros incorretos. Podem ser utilizados dados do tipo: números, strings, booleanos, float, listas, tuplas, dicionários, funções, dentre outros.

```
print('Tédoido!')  # Passagem de parâmetro.

print()  # Sem passagem de parâmetro. Não retorna nada.
```

Outro exemplo. Neste caso, os parâmetros possuem valores iguais a 5 e 10, respectivamente. Caso o usuário não forneça os parâmetros, esses dois valores serão utilizados, logo o resultado será 15. No segundo caso, passou-se apenas o primeiro parâmetro com valor 2, ou seja, 2 + 10 = 12, e por fim, no terceiro caso, somam-se os dois valores, 4 + 4 = 8.

```python
def soma(num1=5, num2=10):
    return num1 + num2

print(soma())  # 15
print(soma(2))  # 12
print(soma(4, 4))  # 8
```

Um ponto importante.

```python
nome = 'Egua'  # Variável global.

def diz_egua():
    nome = 'Mano'  # Variável local.
    return f'{nome}! Não acredito!'

print(diz_egua())  # Mano! Não acredito!
```

Caso seja utilizada uma variável local (variável nome) com o mesmo nome da variável global (variável nome), a variável local será utilizada.

No exemplo abaixo, será retornado NameError porque a variável nome está apenas no contexto da função, ou seja, ela é uma variável local.

```python
def diz_egua():
    nome = 'Egua'  # Variável local.
    return f'{nome}! Não acredito!'

print(diz_egua())  # Egua! Não acredito!

print(nome)  # NameError: name 'nome' is not defined
```

Documentando funções com DocString

São informações fornecidas para o melhor entendimento da função.

```
def diz_egua():
    """Função que retorna uma expressão paraense."""
    nome = 'Egua'  # Variável local.
    return f'{nome}! Não acredito!'

print(diz_egua())  # Egua! Não acredito!

print(help(diz_egua))
```

```
Help on function diz_egua in module __main__:

diz_egua()
    Função que retorna uma expressão paraense.
(END)
```

```
# Utilizando a propriedade especial __doc__.
print(diz_egua.__doc__)  # Função que retorna uma expressão paraense.
```

Outro exemplo.

```
def calcula_umidade_relativa(e, es):
    """
    Função que calcula a umidade relativa em %
    dada as variáveis "e" e "es".
    e = pressão de vapor (hPa).
    es = pressão de saturação do vapor (hPa).
    """
    ur = (e/es) * 100
    return ur


rh = calcula_umidade_relativa(209, 383)
print(f'O valor da UR é: {rh}%')  # O valor da UR é: 54.56919060052219%

print(calcula_umidade_relativa.__doc__)
```

```
    Função que calcula a umidade relativa em %
    dada as variáveis "e" e "es".
    e = pressão de vapor (hPa).
    es = pressão de saturação do vapor (hPa).
```

Entendendo a estrutura de um arquivo NetCDF

Utilizando a biblioteca xarray

Serão mostrados alguns comandos para explorar o conteúdo de um arquivo no formato NetCDF. Essa é uma boa prática antes de iniciar o processamento porque os metadados fornecem informações importantes sobre a variável e suas dimensões.

```
import xarray as xr


# Abertura do arquivo NetCDF.
ds = xr.open_dataset('prec.2020.nc', decode_times=False)


print(ds)
```

O resultado será:

```
<xarray.Dataset>
Dimensions:  (lat: 240, lon: 320, time: 9)
Coordinates:
  * time     (time) float64 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
  * lon      (lon) float32 -74.95 -74.85 -74.75 -74.65 ... -43.25 -43.15 -43.05
  * lat      (lat) float32 -17.95 -17.85 -17.75 -17.65 ... 5.65 5.75 5.85 5.95
Data variables:
    prec     (time, lat, lon) float32 ...
Attributes:
    CDI:                       Climate Data Interface version 1.9.7.1 (http:/...
    Conventions:               CF-1.6
    history:                   Thu Oct 01 11:38:46 2020: cdo -s -sellonlatbox...
    NCO:                       "4.5.4"
    nco_openmp_thread_number:  1
    BeginDate:                 2020-01-01
    BeginTime:                 00:00:00.000Z
    EndDate:                   2020-01-01
    EndTime:                   23:59:59.999Z
    FileHeader:                StartGranuleDateTime=2020-01-01T00:00:00.000Z;...
    InputPointer:              3B-HHR-E.MS.MRG.3IMERG.20200101-S000000-E00295...
    title:                     GPM IMERG Early Precipitation L3 1 day 0.1 deg...
    DOI:                       10.5067/GPM/IMERGDE/DAY/06
    ProductionTime:            2020-01-02T15:43:16.718Z
    frequency:                 mon
    CDO:                       Climate Data Operators version 1.9.7.1 (http:/...
```

Na linha Dimensions, nota-se que esta variável possui 3 dimensões, a saber: lat: 240 pontos (variando de -17.95 a 5.95), lon: 320 pontos (variando de -74.95 a -43.05) e time: 9 tempos (variando do índice 0 a 8). O nome da variável do arquivo prec.2020.nc chama-se prec e possui a seguinte disposição das dimensões: prec (time, lat, lon). Portanto, ao manipular essa variável, é importante respeitar a ordem das dimensões, isto é, a primeira dimensão de prec é time, a segunda, lat e a terceira, lon.

É importante conhecer as informações (metadados) do arquivo antes de tudo para posteriormente, realizar-se qualquer tipo de análise. A maioria dos erros ocorrem porque os usuários não entendem a disposição das dimensões, estrutura e limitações do arquivo.

Uma vez que são conhecidas as dimensões e o nome da variável, é interessante explorar o seu conteúdo por meio dos comandos abaixo:

```
import xarray as xr

# Abertura do arquivo NetCDF.
ds = xr.open_dataset('prec.2020.nc', decode_times=False)

print('====================================')
print('Informações sobre a dimensão latitude')
print(ds.lat)

print('====================================')
print('Informações sobre a dimensão longitude')
print(ds.lon)

print('====================================')
print('Informações sobre a dimensão tempo')
print(ds.time)
```

O resultado será:

```
====================================
Informações sobre a dimensão latitude
<xarray.DataArray 'lat' (lat: 240)>
array([-17.95    , -17.850002, -17.749996, ...,   5.750005,   5.850003,
         5.950002], dtype=float32)
Coordinates:
  * lat      (lat) float32 -17.95 -17.85 -17.75 -17.65 ... 5.65 5.75 5.85 5.95
Attributes:
    standard_name:  latitude
    long_name:      Latitude
    units:          degrees_north
    axis:           Y
====================================
Informações sobre a dimensão longitude
<xarray.DataArray 'lon' (lon: 320)>
array([-74.95    , -74.85    , -74.74999 , ..., -43.250004, -43.149998,
       -43.04999 ], dtype=float32)
Coordinates:
  * lon      (lon) float32 -74.95 -74.85 -74.75 -74.65 ... -43.25 -43.15 -43.05
Attributes:
    standard_name:  longitude
    long_name:      Longitude
    units:          degrees_east
    axis:           X
====================================
Informações sobre a dimensão tempo
<xarray.DataArray 'time' (time: 9)>
array([0., 1., 2., 3., 4., 5., 6., 7., 8.])
Coordinates:
  * time     (time) float64 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
Attributes:
    standard_name:  time
    units:          months since 2020-1-1 00:00:00
    calendar:       standard
    axis:           T
```

E por fim, a visualização das informações da variável prec:

```
import xarray as xr

# Abertura do arquivo NetCDF.
ds = xr.open_dataset('prec.2020.nc', decode_times=False)

print('====================================')
print('Informações sobre a variável')
print(ds.prec)
```

O resultado será:

```
====================================
Informações sobre a variável
<xarray.DataArray 'prec' (time: 9, lat: 240, lon: 320)>
[691200 values with dtype=float32]
Coordinates:
  * time     (time) float64 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
  * lon      (lon) float32 -74.95 -74.85 -74.75 -74.65 ... -43.25 -43.15 -43.05
  * lat      (lat) float32 -17.95 -17.85 -17.75 -17.65 ... 5.65 5.75 5.85 5.95
Attributes:
    long_name:  Daily accumulated precipitation (combined microwave-IR) estimate
    units:      mm
```

Apenas lembrando que a ordem das dimensões a serem manipuladas é: time, lat, lon. Isso é muito importante pois evitará erros no momento de realizar operações com a variável prec.

Utilizando a biblioteca netCDF4

Outra forma de visualizar o conteúdo de um arquivo NetCDF pode ser feita por meio da biblioteca netCDF4. As linhas abaixo representam um pequeno trecho de código:

```
from netCDF4 import Dataset as nc


# Abertura do arquivo NetCDF.
ds = nc('prec.2020.nc', mode='r')


print(dir(ds))
```

O resultado será:

```
['BeginDate', 'BeginTime', 'CDI', 'CDO', 'Conventions', 'DOI', 'EndDate', 'EndTime', 'FileHeader', 'InputPoin
ter', 'NCO', 'ProductionTime', '__class__', '__delattr__', '__dir__', '__doc__', '__enter__', '__eq__', '__ex
it__', '__format__', '__ge__', '__getattr__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__ini
t__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__new__', '__orthogonal_indexing__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__unicode__', '_clo
se', '_close_mem', '_enddef', '_getname', '_grpid', '_isopen', '_ncstring_attrs__', '_redef', 'close', 'cmpty
pes', 'createCompoundType', 'createDimension', 'createEnumType', 'createGroup', 'createVLType', 'createVariab
le', 'data_model', 'delncattr', 'dimensions', 'disk_format', 'enumtypes', 'file_format', 'filepath', 'frequen
cy', 'get_variables_by_attributes', 'getncattr', 'groups', 'history', 'isopen', 'keepweakref', 'name', 'ncatt
rs', 'nco_openmp_thread_number', 'parent', 'path', 'renameAttribute', 'renameDimension', 'renameGroup', 'rena
meVariable', 'set_always_mask', 'set_auto_chartostring', 'set_auto_mask', 'set_auto_maskandscale', 'set_auto_
scale', 'set_fill_off', 'set_fill_on', 'set_ncstring_attrs', 'setncattr', 'setncattr_string', 'setncatts', 's
ync', 'title', 'variables', 'vltypes']
```

A partir do resultado obtido, pode-se explorar as informações de cada uma das opções disponíveis. Inicialmente, vamos utilizar o *variables* que se encontra na lista acima.

```
from netCDF4 import Dataset as nc

# Abertura do arquivo NetCDF.
ds = nc('prec.2020.nc', mode='r')

print(ds.variables.keys())  # dict_keys(['time', 'lon', 'lat', 'prec'])

print('=========================================')
print(ds.variables)
```

O resultado será:

```
dict_keys(['time', 'lon', 'lat', 'prec'])
=========================================
{'time': <class 'netCDF4._netCDF4.Variable'>
float64 time(time)
    standard_name: time
    units: months since 2020-1-1 00:00:00
    calendar: standard
    axis: T
unlimited dimensions: time
current shape = (9,)
filling on, default _FillValue of 9.969209968386869e+36 used, 'lon': <class 'netCDF4._netCDF4.Variable'>
float32 lon(lon)
    standard_name: longitude
    long_name: Longitude
    units: degrees_east
    axis: X
unlimited dimensions:
current shape = (320,)
filling on, default _FillValue of 9.969209968386869e+36 used, 'lat': <class 'netCDF4._netCDF4.Variable'>
float32 lat(lat)
    standard_name: latitude
    long_name: Latitude
    units: degrees_north
    axis: Y
unlimited dimensions:
current shape = (240,)
filling on, default _FillValue of 9.969209968386869e+36 used, 'prec': <class 'netCDF4._netCDF4.Variable'>
float32 prec(time, lat, lon)
    long_name: Daily accumulated precipitation (combined microwave-IR) estimate
    units: mm
    _FillValue: -999.0
    missing_value: -999.0
unlimited dimensions: time
current shape = (9, 240, 320)
filling on}
```

Nota-se a descrição das variáveis (time, lon, lat, prec) com os seus metadados (informações). Uma dica importante, o nome que está entre parênteses (lon (lon) ou prec (time, lat, lon)) representa o nome da dimensão. Isto é, a variável lon tem como dimensão lon, por sua vez, a variável prec possui as dimensões time, lat e lon.

Caso seja necessário visualizar apenas as informações de uma variável em particular, basta digitar:

```
print(ds.variables['prec'])
```

Agora, deseja-se visualizar o conteúdo da variável lat, basta digitar:

```
print(ds.variables['lat'][:])
```

Manipulação de arquivo texto utilizando o pandas

O DataFrame a ser utilizado para exemplificar o uso do pandas é apresentado abaixo. O objetivo consiste em mostrar algumas aplicações com pandas para explorar o conjunto de dados. Existem várias formas de realizar essa manipulação. Para mais informações, acessar o link abaixo.

https://pandas.pydata.org

Neste exemplo será utilizado o pd.read_csv para ler o arquivo ".csv". Mais informações podem ser obtidas no link abaixo.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Observação: Caso seu arquivo possua valores indefinidos ou ausentes, por exemplo, -999, no momento de abri-lo, basta adicionar o parâmetro na_values=[-999] no pd.read_csv().

Ano	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
2003	3	12	5	17	14	27	31	26	11	8	4	11
2004	5	4	8	7	13	22	11	14	30	9	8	4
2005	3	4	6	21	12	18	31	31	10	14	4	4
2006	15	7	4	12	12	30	26	19	8	6	4	3
2007	2	4	13	10	17	30	27	26	29	18	4	7
2008	5	8	7	13	31	30	31	24	14	15	4	4
2009	5	7	6	10	15	12	26	21	12	3	8	3
2010	6	7	5	24	11	23	31	31	12	5	3	6
2011	15	13	6	20	15	19	31	31	23	4	4	3
2012	13	8	10	13	8	16	31	31	17	12	4	7
2013	9	20	5	8	25	17	31	17	9	7	4	2
2014	7	15	12	17	11	16	24	24	17	17	7	7
2015	19	7	6	12	10	29	26	18	15	21	6	5
2016	4	9	8	26	18	23	30	12	12	11	4	5
2017	9	17	7	21	16	30	31	22	29	17	5	4
2018	6	10	8	12	15	25	31	13	10	5	5	10
2019	11	4	8	13	14	26	16	22	24	9	6	5
2020	2	3	12	14	7	25	31	16	20	4	7	4

Imprimir linhas

Imprime as 5 primeiras linhas de cada coluna do arquivo.

```
import pandas as pd

# Leitura do arquivo no formato ".csv".
df1 = pd.read_csv('eca_cdd.R.txt')

print(df1.head())
```

Resultado:

```
    Ano  Jan  Fev  Mar  Abr  Mai  Jun  Jul  Ago  Set  Out  Nov  Dez
0  2003    3   12    5   17   14   27   31   26   11    8    4   11
1  2004    5    4    8    7   13   22   11   14   30    9    8    4
2  2005    3    4    6   21   12   18   31   31   10   14    4    4
3  2006   15    7    4   12   12   30   26   19    8    6    4    3
4  2007    2    4   13   10   17   30   27   26   29   18    4    7
```

Imprime as 5 últimas linhas de cada coluna do arquivo.

```
print(df1.tail())
```

Resultado:

```
     Ano  Jan  Fev  Mar  Abr  Mai  Jun  Jul  Ago  Set  Out  Nov  Dez
13  2016    4    9    8   26   18   23   30   12   12   11    4    5
14  2017    9   17    7   21   16   30   31   22   29   17    5    4
15  2018    6   10    8   12   15   25   31   13   10    5    5   10
16  2019   11    4    8   13   14   26   16   22   24    9    6    5
17  2020    2    3   12   14    7   25   31   16   20    4    7    4
```

Imprimir o formato do conjunto de dados

```
print(df1.shape)  # (18, 13) ==> 18 linhas x 13 colunas
```

Imprimir o tipo do dado

```
print(type(df1))  # <class 'pandas.core.frame.DataFrame'>
```

Imprimir os índices do dado

```
print(df1.index)  # RangeIndex(start=0, stop=18, step=1)
```

Imprimir a estatística descritiva sobre os dados

```
print(df1.describe())
```

Resultado:

```
               Ano        Jan        Fev  ...        Dez
count    18.000000  18.000000  18.000000  ...  18.000000
mean   2011.500000   7.722222   8.833333  ...   5.222222
std       5.338539   5.015336   4.853743  ...   2.414553
min    2003.000000   2.000000   3.000000  ...   2.000000
25%    2007.250000   4.250000   4.750000  ...   4.000000
50%    2011.500000   6.000000   7.500000  ...   4.500000
75%    2015.750000  10.500000  11.500000  ...   6.750000
max    2020.000000  19.000000  20.000000  ...  11.00000
```

Imprimir os títulos das colunas

```
print(df1.columns)
```

Resultado:

```
Index(['Ano', 'Jan', 'Fev', 'Mar', 'Abr', 'Mai', 'Jun', 'Jul', 'Ago', 'Set',
       'Out', 'Nov', 'Dez'],
      dtype='object')
```

Checar dado ausente em alguma coluna

```
print(df1.isnull())
```

Resultado:

```
      Ano    Jan    Fev ...    Jul    Ago    Set    Out    Nov    Dez
0   False  False  False ...  False  False  False  False  False  False
1   False  False  False ...  False  False  False  False  False  False
2   False  False  False ...  False  False  False  False  False  False
3   False  False  False ...  False  False  False  False  False  False
4   False  False  False ...  False  False  False  False  False  False
5   False  False  False ...  False  False  False  False  False  False
6   False  False  False ...  False  False  False  False  False  False
7   False  False  False ...  False  False  False  False  False  False
8   False  False  False ...  False  False  False  False  False  False
9   False  False  False ...  False  False  False  False  False  False
10  False  False  False ...  False  False  False  False  False  False
11  False  False  False ...  False  False  False  False  False  False
12  False  False  False ...  False  False  False  False  False  False
13  False  False  False ...  False  False  False  False  False  False
14  False  False  False ...  False  False  False  False  False  False
15  False  False  False ...  False  False  False  False  False  False
16  False  False  False ...  False  False  False  False  False  False
17  False  False  False ...  False  False  False  False  False  False
```

Somar o total de valores nulos em cada coluna

```
print(df1.isnull().sum())
```

Resultado:

```
Ano    0
Jan    0
Fev    0
Mar    0
Abr    0
Mai    0
Jun    0
Jul    0
Ago    0
Set    0
Out    0
Nov    0
Dez    0
```

Deletar a linha no caso de valores faltantes

```
print(df1.dropna(how='any'))
```

Deletar a linha no caso de todos os valores serem faltantes

```
print(df1.dropna(how='all'))
```

Deletar a linha caso os meses Fev ou Jul tenham valores faltantes

```
print(df1.dropna(subset=['Fev', 'Jul'], how='any'))
```

Realizar o agrupamento dos valores por categoria e sua contagem

Os valores faltantes não são considerados.

```
# Abertura do arquivo.
df2 = pd.read_csv('direcao_vento.txt')


print(df2['Direção'].value_counts())
```

Resultado:

```
ENE    139
E       64
NE       9
NNE      3
N        1
Name: Direção, dtype: int64
```

No caso de considerar valores faltantes, utiliza-se a opção dropna=False.

```
print(df2['Direção'].value_counts(dropna=False))
```

Resultado:

```
ENE    139
E       64
NE       9
NNE      3
N        1
Name: Direção, dtype: int64
```

O resultado é o mesmo porque não há dados faltantes.

Retornar informações sobre o arquivo aberto

```
print(df1.info())
```

Resultado:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Ano     18 non-null     int64
 1   Jan     18 non-null     int64
 2   Fev     18 non-null     int64
 3   Mar     18 non-null     int64
 4   Abr     18 non-null     int64
 5   Mai     18 non-null     int64
 6   Jun     18 non-null     int64
 7   Jul     18 non-null     int64
 8   Ago     18 non-null     int64
 9   Set     18 non-null     int64
 10  Out     18 non-null     int64
 11  Nov     18 non-null     int64
 12  Dez     18 non-null     int64
dtypes: int64(13)
memory usage: 2.0 KB
None
```

Deletar uma coluna

Neste caso, será removida a coluna Ano.

```
df1.drop(['Ano'], axis=1, inplace=True)
print(df1.info())
```

Resultado:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Jan     18 non-null     int64
 1   Fev     18 non-null     int64
 2   Mar     18 non-null     int64
 3   Abr     18 non-null     int64
 4   Mai     18 non-null     int64
 5   Jun     18 non-null     int64
 6   Jul     18 non-null     int64
 7   Ago     18 non-null     int64
 8   Set     18 non-null     int64
 9   Out     18 non-null     int64
 10  Nov     18 non-null     int64
 11  Dez     18 non-null     int64
dtypes: int64(12)
memory usage: 1.8 KB
None
```

Filtrar dados por limiares

Retorna o menor e o maior valor do conjunto de dados do DataFrame df1. Neste exemplo, serão retornados os 4 menores e os 4 maiores valores da coluna Jan, respectivamente.

```
print(df1.nsmallest(4, 'Jan'))
print(df1.nlargest(4, 'Jan'))
```

Filtrar dados por coluna

Seleciona linhas específicas por meio das colunas. Retorna os 6 primeiros valores das colunas Jan e Fev. Lembrando que no Python os índices iniciam em 0, logo, 0 a 5, são seis valores.

```
x = df1.loc[:5, ['Jan', 'Fev']]
print(x)
```

Resultado:

```
   Jan  Fev
0    3   12
1    5    4
2    3    4
3   15    7
4    2    4
5    5    8
```

Filtrar dados por linha e coluna

Seleciona linhas particulares de acordo com a sua posição específica por linhas e colunas. Retorna as 4 primeiras linhas e as 4 primeiras colunas.

```
x = df1.iloc[:3, :3]
print(x)
```

Resultado:

```
    Ano  Jan  Fev
0  2003    3   12
1  2004    5    4
2  2005    3    4
```

Selecionar valores por intervalo de interesse

Seleciona os valores de Jul maiores que 15 e menores que 25.

```
x = df1.query('15 < Jul < 25').head()
print(x)
```

Resultado:

```
     Ano  Jan  Fev  Mar  Abr  Mai  Jun  Jul  Ago  Set  Out  Nov  Dez
11  2014    7   15   12   17   11   16   24   24   17   17    7    7
16  2019   11    4    8   13   14   26   16   22   24    9    6    5
```

Definir a coluna como index

Definindo a coluna “Ano” como *index*.

```
print(df1.set_index('Ano').head(4))
```

Resultado:

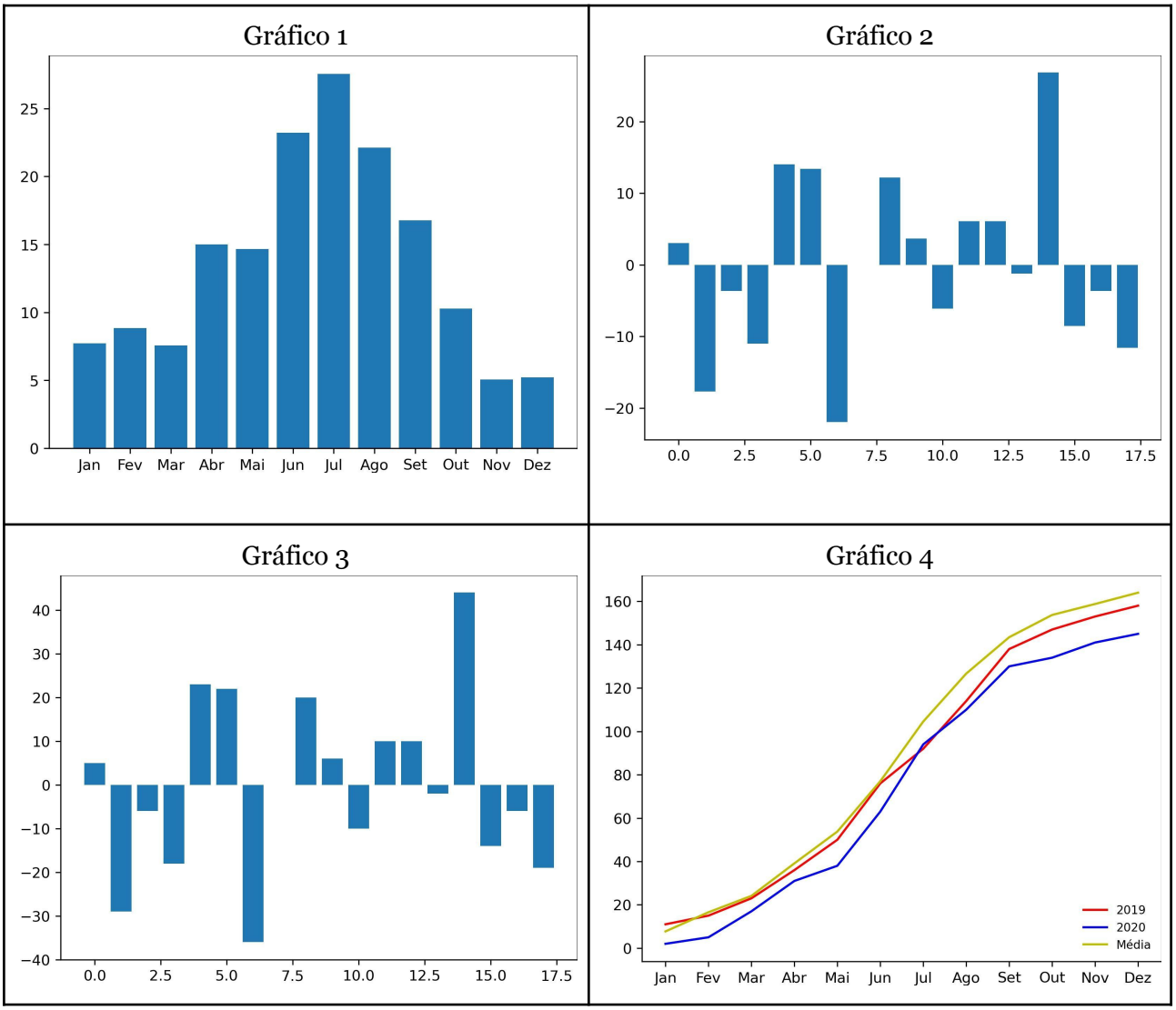
```
      Jan  Fev  Mar  Abr  Mai  Jun  Jul  Ago  Set  Out  Nov  Dez
Ano
2003    3   12    5   17   14   27   31   26   11    8    4   11
2004    5    4    8    7   13   22   11   14   30    9    8    4
2005    3    4    6   21   12   18   31   31   10   14    4    4
2006   15    7    4   12   12   30   26   19    8    6    4    3
```

O objetivo do script abaixo consiste em processar o conjunto de dados (eca_cdd.R.txt) descrito no início deste capítulo para reforçar o que foi aprendido. São feitas algumas manipulações, e por fim, são gerados alguns gráficos.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from itertools import accumulate

# Leitura do arquivo no formato ".csv".
df1 = pd.read_csv('eca_cdd.R.txt')

df1.set_index('Ano', inplace=True)  # Define a coluna "Ano" como o índice.

mes = ['Jan', 'Fev', 'Mar', 'Abr', 'Mai', 'Jun',
       'Jul', 'Ago', 'Set', 'Out', 'Nov', 'Dez']

med_mensal = []  # Cria uma lista vazia para armazenar os valores.

# Calcula a média mensal (climatologia - jan, ..., dez).
for linha in mes:
    med_mensal.append(df1[linha[:]].mean())

plt.bar(mes, med_mensal)  # Gera o gráfico 1.

plt.savefig('ex01.jpg', transparent=True, dpi=300, bbox_inches='tight',
             pad_inches=0)

plt.show()  # Mostra o gráfico na tela.

# Limpa o ambiente gráfico para evitar erros com o próximo gráfico.
plt.close("all")

# Calcula para cada ano o total anual.
total_anual = df1.sum(axis=1)
# Calcula a média de todos os anos.
mensal_cumulativo = total_anual.mean()
# Calcula o acréscimo ou decréscimo anual em porcentagem.
pct_anual = list(((total_anual-mensal_cumulativo)/mensal_cumulativo)*100)
# Calcula a anomalia (desvio em relação à média).
anomalia_anual = list(total_anual-mensal_cumulativo)

x = np.arange(0, 18)  # Gera um vetor com 18 posições (0 a 17 valores).

plt.bar(x, pct_anual)  # Gera o gráfico 2.
```

```
plt.savefig('ex02.jpg', transparent=True, dpi=300, bbox_inches='tight',
            pad_inches=0)

plt.show()  # Mostra o gráfico na tela.

# Limpa o ambiente gráfico para evitar erros com o próximo gráfico.
plt.close("all")

plt.bar(x, anomalia_anual)  # Gera o gráfico 3.

plt.savefig('ex03.jpg', transparent=True, dpi=300, bbox_inches='tight',
            pad_inches=0)

plt.show()  # Mostra o gráfico na tela.

# Limpa o ambiente gráfico para evitar erros com o próximo gráfico.
plt.close("all")

acumulado_anual = []  # Cria uma lista vazia para armazenar os valores.

# Calculando os valores cumulativos para cada ano.
for i in x:
     coluna = df1.iloc[i,:]
     acumulado_anual.append(list(accumulate(coluna)))

# Calcula a média mensal (climatologia) cumulativo de todos os anos.
media_mensal_cumulativo = np.average(acumulado_anual, axis=0)

# Gera o gráfico 4.
plt.plot(mes, acumulado_anual[16], label='2019', color='r')
plt.plot(mes, acumulado_anual[17], label='2020', color='b')
plt.plot(mes, media_mensal_cumulativo, label='Média', color='y')

# Gera a legenda sem borda, define localização e tamanho da fonte.
plt.legend(frameon =False, loc='lower right', fontsize=8)

plt.savefig('ex04.jpg', transparent=True, dpi=300, bbox_inches='tight',
            pad_inches=0)

plt.show()  # Mostra o gráfico na tela.
```

O resultado gráfico será:



Outras aplicações utilizando Pandas

Será utilizado o mesmo arquivo do exercício anterior (eca_cdd.R.txt) com duas novas colunas (Total e Rank) que serão mostradas como foram obtidas. Nesta nova seção, outras opções de manipulação do DataFrame serão apresentadas.

	Ano	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez	Total	Rank
0	2003	3	12	5	17	14	27	31	26	11	8	4	11	169	8.0
1	2004	5	4	8	7	13	22	11	14	30	9	8	4	135	17.0
2	2005	3	4	6	21	12	18	31	31	10	14	4	4	158	11.0
3	2006	15	7	4	12	12	30	26	19	8	6	4	3	146	15.0
4	2007	2	4	13	10	17	30	27	26	29	18	4	7	187	2.0
5	2008	5	8	7	13	31	30	31	24	14	15	4	4	186	3.0
6	2009	5	7	6	10	15	12	26	21	12	3	8	3	128	18.0
7	2010	6	7	5	24	11	23	31	31	12	5	3	6	164	9.0
8	2011	15	13	6	20	15	19	31	31	23	4	4	3	184	4.0
9	2012	13	8	10	13	8	16	31	31	17	12	4	7	170	7.0
10	2013	9	20	5	8	25	17	31	17	9	7	4	2	154	13.0
11	2014	7	15	12	17	11	16	24	24	17	17	7	7	174	5.0
12	2015	19	7	6	12	10	29	26	18	15	21	6	5	174	6.0
13	2016	4	9	8	26	18	23	30	12	12	11	4	5	162	10.0
14	2017	9	17	7	21	16	30	31	22	29	17	5	4	208	1.0
15	2018	6	10	8	12	15	25	31	13	10	5	5	10	150	14.0
16	2019	11	4	8	13	14	26	16	22	24	9	6	5	158	12.0
17	2020	2	3	12	14	7	25	31	16	20	4	7	4	145	16.0

```
import pandas as pd
import numpy as np


df = pd.read_csv("eca_cdd.R.txt", sep=',')

Numero_De_Anos = len(df.index)
coluna = []  # Cria uma lista vazia.

# Calcula o total anual.
for i in np.arange(0, Numero_De_Anos):
    total_anual = df.iloc[i, 1:].sum()
    coluna.append(total_anual)

df['Total'] = coluna  # Cria a coluna "Total" no DataFrame com os valores da
                      # soma anual.
```

Criar um ranking dos valores

Será utilizado o rank e o seu resultado pode ser visto nas colunas adicionais (Total e Rank) do DataFrame acima. O ranking é feito do maior para o menor valor.

```
df['Rank'] = df['Total'].rank(method='first', ascending=False)
```

Extrair uma coluna do DataFrame

Para extrair uma coluna do DataFrame, utiliza-se a linha abaixo.

```
Coluna_Total = df['Ano', 'Total']
print(Coluna_Total)
```

Extrair múltiplas colunas do DataFrame

```
Varias_Colunas = df[['Ano', 'Jun', 'Jul', 'Ago']]
print(Varias_Colunas)
```

Checar o tipo de uma determinada coluna

```
print(type(df['Total']))  # <class 'pandas.core.series.Series'>
```

Criar nova coluna no DataFrame

Será criada uma nova coluna no DataFrame com o nome “JJA” resultante da soma dos valores dos meses de junho, julho e agosto.

```
df['JJA'] = df['Jun'] + df['Jul'] + df['Ago']
```

Remover uma coluna do DataFrame

Dica importante sobre remoção de coluna. Antes de continuar com essa tarefa, digite o comando abaixo:

```
print(df.shape)
```

Será retornada a tupla com a seguinte informação:

(18, 15)

A forma de um DataFrame é armazenada como uma tupla em que o elemento indexado 0 (zero) denota o número de linhas e o elemento indexado 1 (um) mostra o número de colunas. Portanto, os valores do eixo 0 e 1 denotam linhas e colunas, respectivamente. Por isso, o valor axis=1, ou seja, coluna.

Para remover a coluna rank, utiliza-se o comando abaixo.

```
df.drop('Rank', axis=1, inplace=True)
```

Uso do iloc

O iloc() seleciona os dados pela posição (índice da coluna), ou seja, o mais importante sobre o iloc() é que ele seleciona os dados utilizando índices numéricos.

Observação:

O iloc() é exclusive, isto é, em uma seleção de vários valores, o valor do último índice não é considerado.

Exemplo: O resultado será A B C e não A B C D justamente pela característica do iloc() de não considerar o último elemento.

```
x = pd.DataFrame(['A', 'B', 'C', 'D', 'E'])
# Índice           0    1    2    3    4


print(x.iloc[0:3])  # A B C
```

Exemplo 1: Retornar o primeiro valor da variável.

É selecionada a primeira linha e a primeira coluna.

```
x = df.iloc[0, 0]  # 2003. Primeiro valor da coluna ano.
```

Exemplo 2: Seleção de várias colunas.

A linha abaixo seleciona todas as linhas (:) e colunas com índice 0 (ano), 2 (Fev) e 10 (Out).

```
x = df.iloc[:, [0, 2, 10]]
```

Exemplo 3: Selecionar a última coluna do DataFrame.

```
x = df.iloc[:, -1]
```

Exemplo 4: Selecionar uma única linha.

```
# Forma 1:
x = df.iloc[0]

# Forma 2:
x = df.iloc[[0]]
```

Exemplo 5: Selecionar várias linhas.

```
x = df.iloc[[0, 5, 10]]
```

Exemplo 6: Selecionar as últimas linhas.

```
x = df.iloc[[-5, -4, -3, -2, -1]]
```

Exemplo 7: Selecionar parte do DataFrame.

São selecionadas as linhas com índice 0, 5 e 10 e as colunas com índice 0, 3 e 7.

```
x = df.iloc[[0, 5, 10], [0, 3, 7]]
```

Exemplo 8: A partir da coluna de interesse (“Ago”), identificar os índices com os menores (idxmin) e maiores (idxmax) valores. Será retornada a linha com as informações, não somente o índice.

```
x = df.iloc[[df['Ago'].idxmin(), df['Ago'].idxmax()]]
```

Exemplo 9: Selecionar uma linha específica (13) com todas as colunas (:).

```
x = df.iloc[13, :]
```

Exemplo 10: Média de cada coluna.

Selecionam-se todas as linhas de cada uma das colunas desde Jan até Dez (1:13), e depois calcula-se a média mensal e faz o arredondamento para inteiro.

```
x = df.iloc[:, 1:13].mean().round(0)
```

Uso do loc

O loc() seleciona os dados pelos rótulos da linha ou coluna. Diferentemente do iloc(), o loc() é inclusive.

Exemplo: O resultado será A B C D justamente pela característica do loc() de considerar o último elemento.

```
x = pd.DataFrame(['A', 'B', 'C', 'D', 'E'])
# Índice           0    1    2    3    4

print(x.loc[0:3])  # A B C D
```

Exemplo 1: Extrair linhas e colunas com base em rótulos da coluna ou linha.

É selecionada a linha índice 16 e a coluna “Mai”. o resultado será o valor 14.

```
x = df.loc[16, ['Mai']]
```

Exemplo 2: Selecionar um intervalo de linhas (3:5) e colunas de interesse (“Jul” e “Nov”).

```
x = df.loc[3:5, ['Jul', 'Nov' ]]
```

Exemplo 3: Selecionar linhas específicas e todas as suas colunas.

São selecionadas as linhas com índice 3, 5, 7 e 9 e todas as suas colunas. Note que esses valores possuem passo 2 conforme o comando abaixo, por isso, o intervalo de valores.

```
x = df.loc[3:9:2, :]
```

Exemplo 4: Seleciona múltiplos elementos no DataFrame.

```
# Define a coluna Ano como index para ser utilizado como consulta. O inplace diz
para realizar a alteração no próprio DataFrame.
df.set_index('Ano', inplace=True)
x = df.loc[ [2011, 2017], ['Nov',  'Dez'] ]
```

Exemplo 5: Média em uma coluna específica.

Aplica-se a média na coluna “Mai”. O valor retornado é 15 porque foi utilizado o arredondamento para o valor inteiro (round(0)).

```
x = df.mean().loc['Mai'].round(0)
```

Exemplo 6: Média de todos os anos em múltiplas colunas (meses).

```
x = df.mean().loc[['Mai', 'Ago', 'Nov']].round(0)
```

Exemplo 7: Uso de condicional para realizar o fatiamento no DataFrame.

A partir da coluna “Jul”, selecionam-se apenas os anos em que Jul > 25. O resultado mostrará as demais colunas, mas todas em função de “Jul”.

```
x = df.loc[df.Jul > 25]
```

Exemplo 8: Extrair as linhas onde Jul > 25 e (&) Jul < 31.

Observação: Caso seja necessário utilizar o operador “ou”, basta alterar “&” por “|” (pipe) que representa o operador “ou” ou “or”.

```
x = df.loc[(df.Jul > 25) & (df.Jul < 31)]
```

Outras possibilidades de manipular DataFrame

Exemplo 1: Selecionar linhas onde a coluna específica for maior que um dado limiar.

Seleciona todas as linhas quando a coluna “Ano” for maior que 2015.

```
x = df[ df['Ano'] > 2015 ]
```

Exemplo 2: Selecionar todas as linhas a partir de um intervalo de uma coluna específica.

Seleciona todas as linhas quando a coluna “Ano” for maior igual a 2015 e (&) menor igual a 2018.

Observação: É possível alterar a condição “&” ou “e” por “|” que representa “ou”.

```
x = df[ (df['Ano'] >= 2015) & (df['Ano'] <= 2018) ]
```

Função to_datetime()

A função to_datetime() converte um objeto Python para o formato datetime. Ele pode ter um inteiro, ponto flutuante, lista, Pandas Series ou Pandas DataFrame como argumento. O to_datetime() é muito poderoso quando o conjunto de dados têm valores de série temporal ou datas.

Converte a coluna “Ano” que antes era int64 para o formato datetime64[ns].

```
df['Ano'] = pd.to_datetime(df['Ano'])
```

Função resample()

Outra forma de calcular a média mensal. Inicialmente, converte-se a coluna “Ano” para o formato datetime64[ns] e depois realiza-se a média em cada uma das colunas.

```
df['Ano'] = pd.to_datetime(df['Ano'])
media_mensal = df.resample("Y", on="Ano").mean().round(0)
```

Função between()

Exemplo 1: Incluir os limites especificados. Serão considerados apenas valores maiores e iguais a 25 e menores e iguais a 30.

```
x = df[df['Jun'].between(25, 30]
```

Exemplo 2: Excluir os limites especificados. Serão considerados apenas valores acima de 25 e abaixo de 30. Isso foi possível por meio do “inclusive=False”.

```
x = df[df['Jun'].between(25, 30, inclusive=False)]
```

Enfatizar trechos do DataFrame

Para cada coluna dos meses de Jan a Dez (0:13), realça o valor mínimo (*highlight_min*) e o máximo (*highlight_max*), cada um com uma cor especificada, respectivamente. Para que a coluna "Ano" não fosse afetada, ela foi definida como *index*.

Observação: O resultado somente é visualizado no *Jupyter Notebook*.

```
df.set_index('Ano', inplace=True)
df.iloc[:, 0:13].style.highlight_min(color="blue").highlight_max(color="red")
```

O resultado será:

Ano	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
2003	3	12	5	17	14	27	31	26	11	8	4	11
2004	5	4	8	7	13	22	11	14	30	9	8	4
2005	3	4	6	21	12	18	31	31	10	14	4	4
2006	15	7	4	12	12	30	26	19	8	6	4	3
2007	2	4	13	10	17	30	27	26	29	18	4	7
2008	5	8	7	13	31	30	31	24	14	15	4	4
2009	5	7	6	10	15	12	26	21	12	3	8	3
2010	6	7	5	24	11	23	31	31	12	5	3	6
2011	15	13	6	20	15	19	31	31	23	4	4	3
2012	13	8	10	13	8	16	31	31	17	12	4	7
2013	9	20	5	8	25	17	31	17	9	7	4	2
2014	7	15	12	17	11	16	24	24	17	17	7	7
2015	19	7	6	12	10	29	26	18	15	21	6	5
2016	4	9	8	26	18	23	30	12	12	11	4	5
2017	9	17	7	21	16	30	31	22	29	17	5	4
2018	6	10	8	12	15	25	31	13	10	5	5	10
2019	11	4	8	13	14	26	16	22	24	9	6	5
2020	2	3	12	14	7	25	31	16	20	4	7	4

Função nlargest()

Retorna os três maiores valores de uma coluna (Set).

```
x = df[['Ano', 'Set']].nlargest(3, 'Set')
```

Função nmallest()

Retorna os três menores valores de uma coluna (Set).

```
x = df[['Ano', 'Set']].nsmallest(3, 'Set')
```

Calcular porcentagem de uma coluna

```
import pandas as pd



# Abertura do arquivo.
df = pd.read_csv('direcao_vento.txt')

#  Agrupa pela direção (frequência).
direcao_string = df.groupby('Direção').size()

# Calcula a porcentagem (0-100%) para cada classe de direção do vento.
direcao_porcentagem = df['Direção'].value_counts(normalize=True) * 100

# Cria um DataFrame para armazenar tudo.
direcao_vento = pd.DataFrame({'Dir String':direcao_string,
                              'Dir Porcentagem': direcao_porcentagem.round(1)})


print(direcao_vento)
```

Tratar dados ausentes

O arquivo “dados_ausentes.txt” será utilizado para explicar as possibilidades de tratamento de dados ausentes. O valor ausente escolhido para este arquivo é -999.

```
# Leitura do arquivo.
df = pd.read_csv('dados_ausentes.txt', na_values=[-999])
```

O arquivo possui a seguinte forma:

```
    Ano  Jan  Fev  Mar   Abr   Mai  Jun   Jul   Ago   Set  Out  Nov  Dez
0  2016    4    9  8.0  26.0  18.0   23  30.0   NaN   NaN   11  4.0    5
1  2017    9   17  NaN  21.0  16.0   30   NaN  22.0  29.0   17  5.0    4
2  2018    6   10  8.0   NaN   NaN   25   NaN  13.0  10.0    5  5.0   10
3  2019   11    4  8.0  13.0  14.0   26  16.0  22.0  24.0    9  6.0    5
4  2020    2    3  NaN  14.0   NaN   25   NaN  16.0  20.0    4  NaN    4
```

Exemplo 1: Eliminar todas as linhas com dados ausentes.

Basta ter apenas um dado ausente na linha que ela será eliminada.

```
x = df.dropna()
```

Exemplo 2: Eliminar todas as colunas que possuem valores ausentes. Independentemente da quantidade de valores ausentes.

```
x = df.dropna(axis=1)
```

Exemplo 3: Uso do parâmetro “thresh” para definir um limite de valores NaN (dados faltantes) aceitáveis. Neste exemplo, a coluna que tiver até duas ocorrências de dados faltantes serão mantidas, no caso contrário, serão excluídas. Por isso, o mês de julho foi excluído porque tem 3 ocorrências de dados faltantes. A explicação é a seguinte: nota-se que cada coluna tem 5 valores que corresponde a 100%, logo, 50% de 5 valores, é igual a 2.5 valores ou ocorrências, mas o Python entende apenas o valor 2 e não 2.5.

```
# Mantém 50% dos valores NaN em cada coluna.
x = df.dropna(axis=1, thresh=0.5*len(df))
```

Exemplo 4: Preenchimento de valores ausentes. Os valores NaN serão preenchidos com -999.

```
x = df.fillna(value='-999')
```

Exemplo 5: Os valores ausentes também podem ser substituídos por alguns valores calculados. Por exemplo, ao realizar a média como no exemplo abaixo, cada coluna será preenchida pela média da sua própria coluna. Exemplo, a coluna “Mai” tem os valores 18, 16 e 14 cuja média é igual a 16. Essa média será utilizada apenas na coluna “Mai” para preencher os dados ausentes. O mesmo raciocínio é feito para as demais colunas.

```
x = df.fillna(value=df.mean().round(0))
```

Exemplo 6: Saber se um elemento está faltando ou não por meio da função isnull()

```
x = df.isnull()


# Retorna o total de valores NaN do DataFrame.
print(df.isnull().sum().sum())  # 11
```

Exemplo 7: Contar o número de valores ausentes em cada coluna.

```
x = df.isna().sum()
```

Exemplo 8: Contar o número de valores ausentes em cada linha.

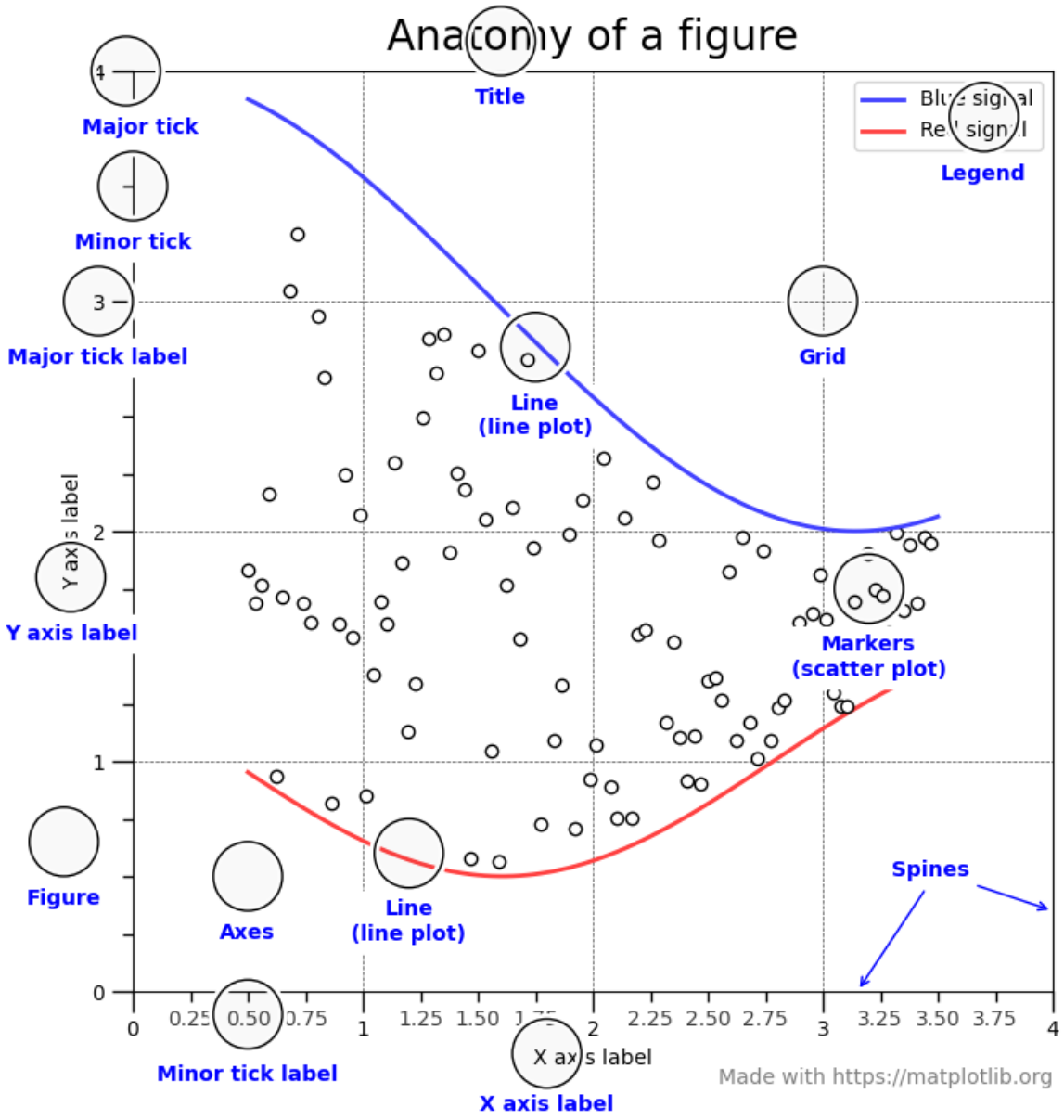
```
x = df.isna().sum(axis=1)
```

Checar se o DataFrame está vazio ou não

```
print(df.empty)  # False
```

Anatomia de um gráfico

É importante conhecer os elementos que compõem um gráfico, e principalmente, o que se deseja apresentar ou não. Essas informações podem ser visualizadas na figura abaixo.



Fonte:
https://matplotlib.org/stable/gallery/showcase/anatomy.html#sphx-glr-gallery-showcase-anatomy-py

Formatação do eixo:
https://matplotlib.org/stable/api/axes_api.html
https://matplotlib.org/stable/api/axes_api.html#axis-limits

Formatação do eixo x:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.axis.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xlim.html#matplotlib.axes.Axes.set_xlim

Formatação do eixo y:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.ylim.html#matplotlib.pyplot.ylim
https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_ylim.html#matplotlib.axes.Axes.set_ylim

Projeção de mapa

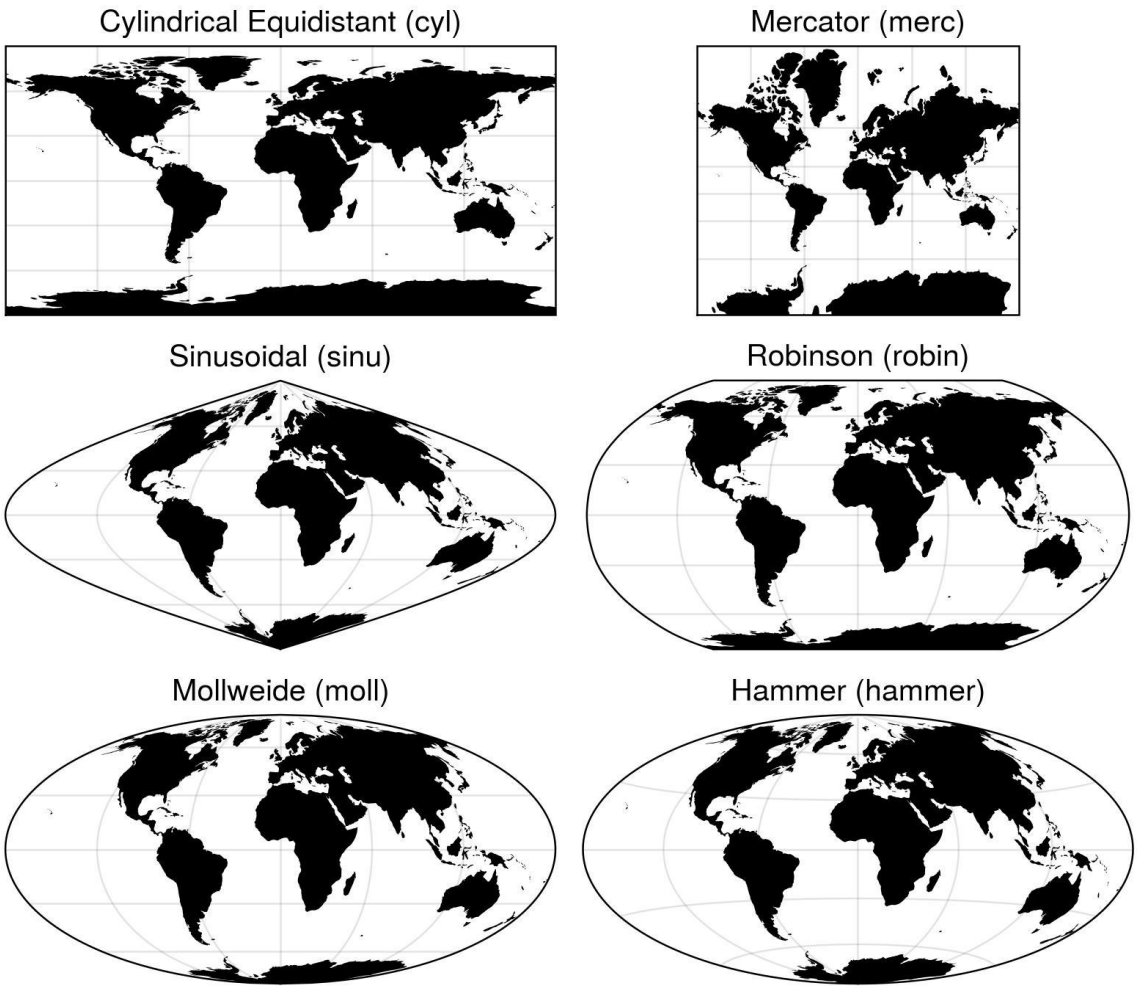
A figura abaixo mostra alguns exemplos possíveis de projeções. Os nomes entre parênteses nas figuras são as projeções que devem ser utilizadas no script.

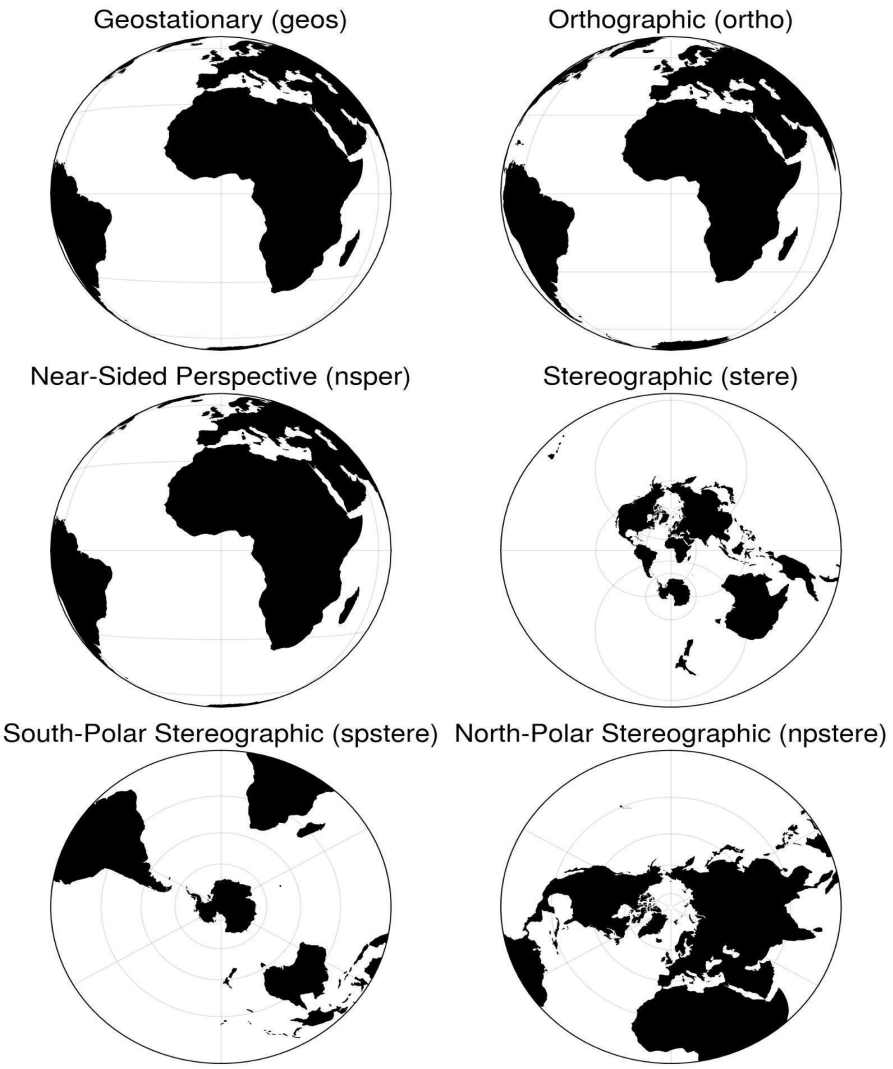
Para alterar o campo de visão das seguintes projeções: `geos`, `ortho`, `nsper`, `stere`, `spstere` e `npstere`, basta alterar a opção (proj_kw) abaixo dentro do script.

O exemplo a seguir, é apenas um trecho de um script genérico. Nesse caso, a visão será definida na longitude -120 (longitude oeste).

```
fig, ax = plot.subplots(proj='geos', proj_kw={'lon_0': -120})
```

Leitura recomendada:
https://basemaptutorial.readthedocs.io/en/latest/index.html
https://proplot.readthedocs.io/en/stable/projections.html
https://scitools.org.uk/cartopy/docs/latest/crs/projections.html
https://matplotlib.org/basemap/api/basemap_api.html#module-mpl_toolkits.basemap
https://proplot.readthedocs.io/en/stable/api/proplot.constructor.Proj.html#proplot.constructor.Proj





Mais opções de formatação do mapa como zoom, definição de linhas e cores, dentre outras possibilidades podem ser encontradas em:

https://proplot.readthedocs.io/en/latest/api/proplot.axes.GeoAxes.format.html

Exemplo de um pequeno trecho de código utilizando a formatação acima.

```
# Formatação do mapa.
ax.format(coast=True, borders=True, innerborders=True,
          labels=True, grid=False, latlines=10, lonlines=10,
          latlim=(-57, 31), lonlim=(-116, -33), small='9px', large='10px',
          title='01/07/2021', labelpad=10)
```

Seleção de cores

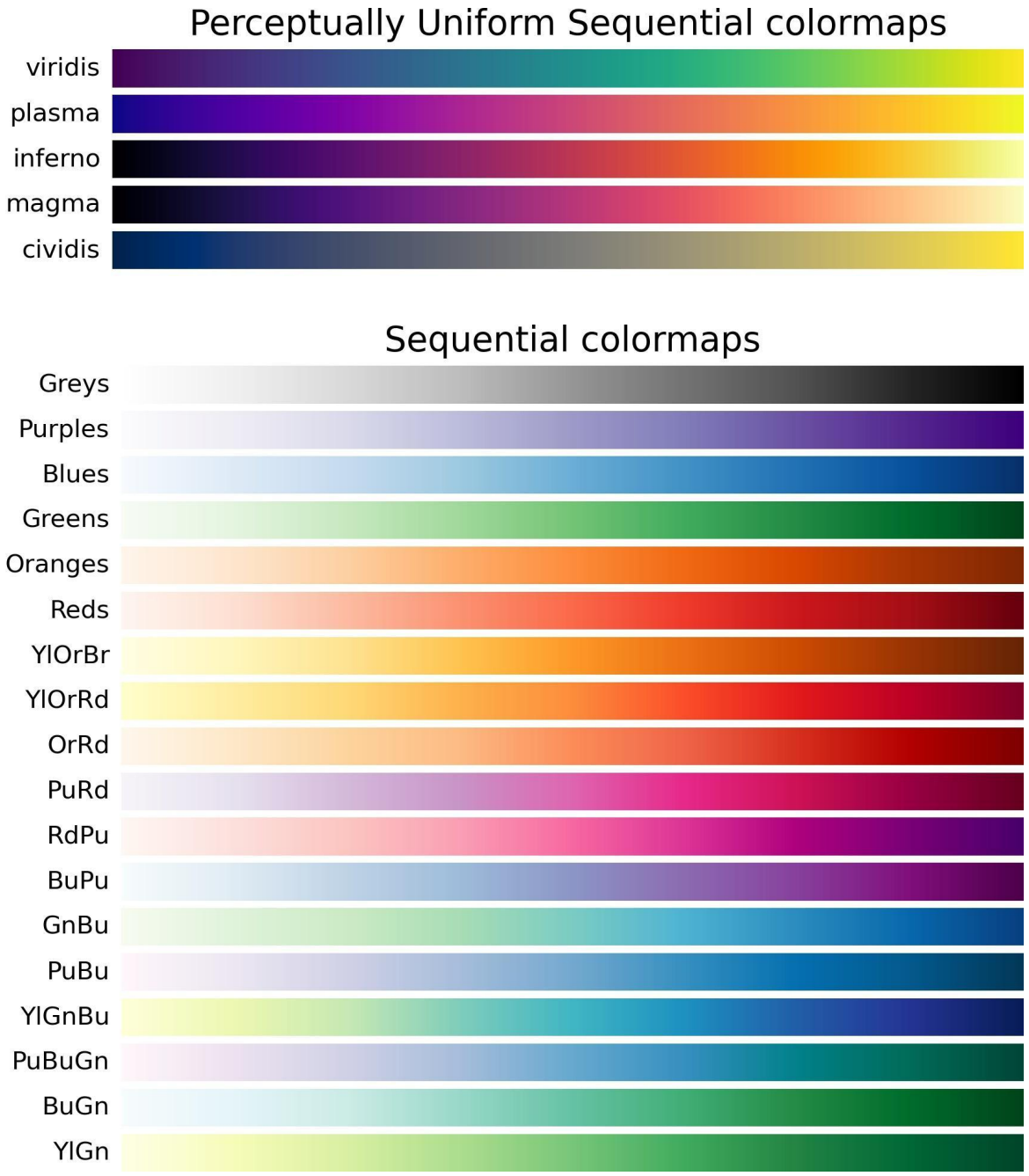
É possível selecionar as cores pelo nome de acordo com os links abaixo:

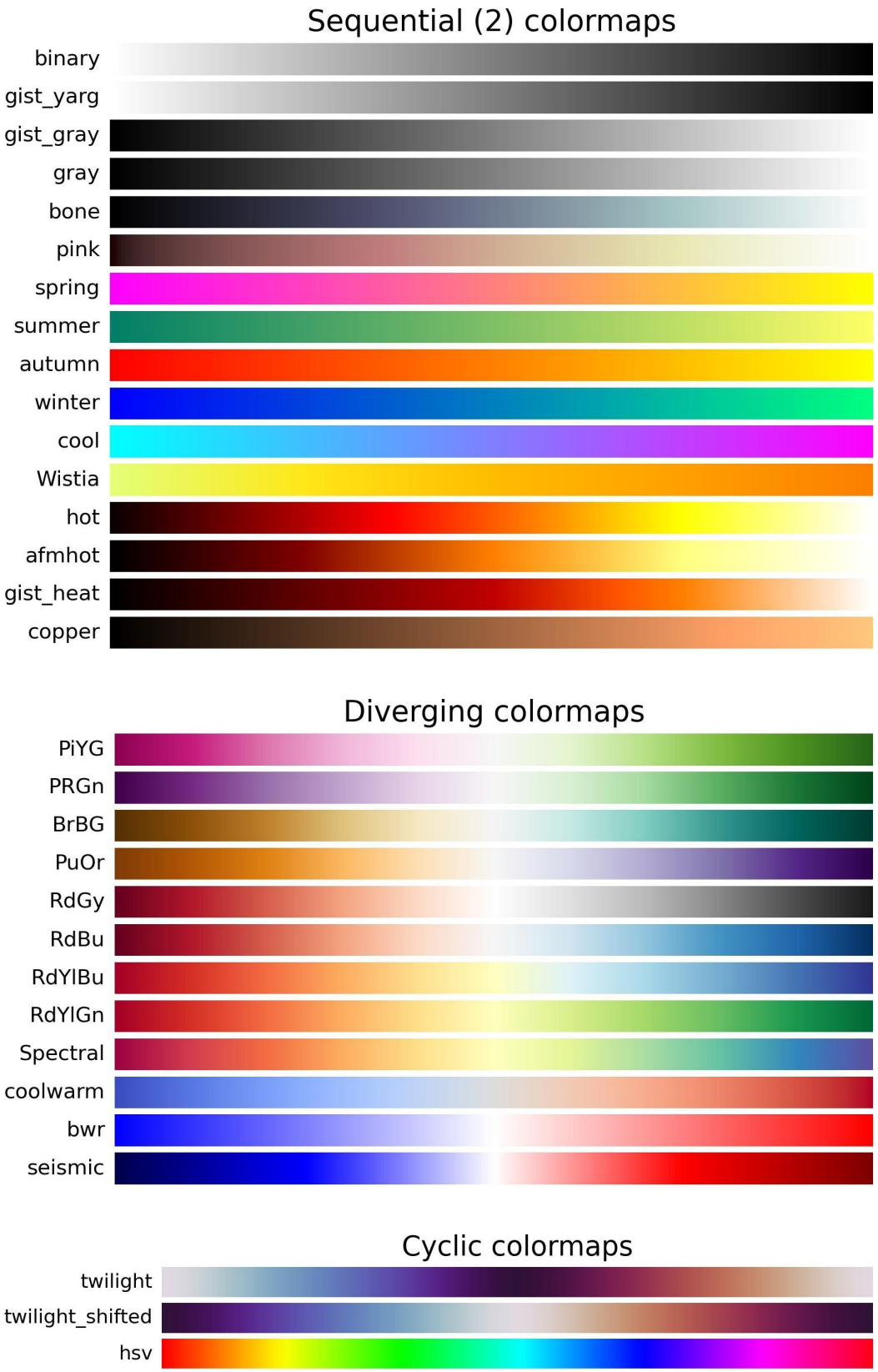
https://proplot.readthedocs.io/en/latest/colors.html
https://matplotlib.org/stable/gallery/color/named_colors.html

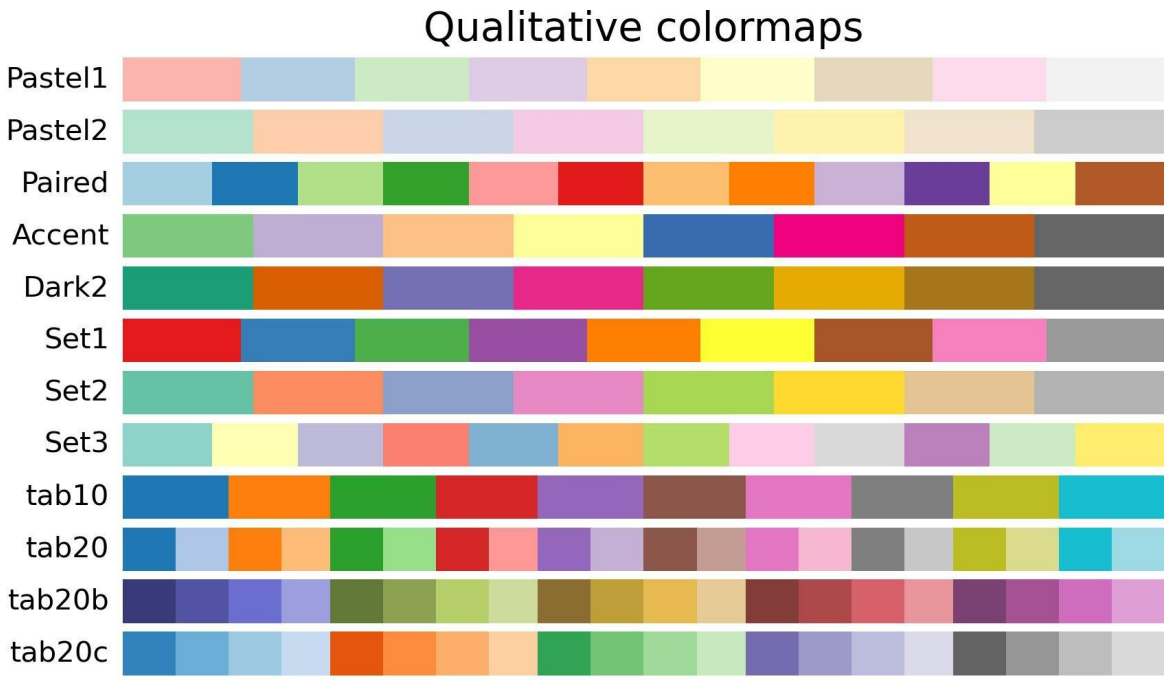
Como usar? color='k' ou c='k'.

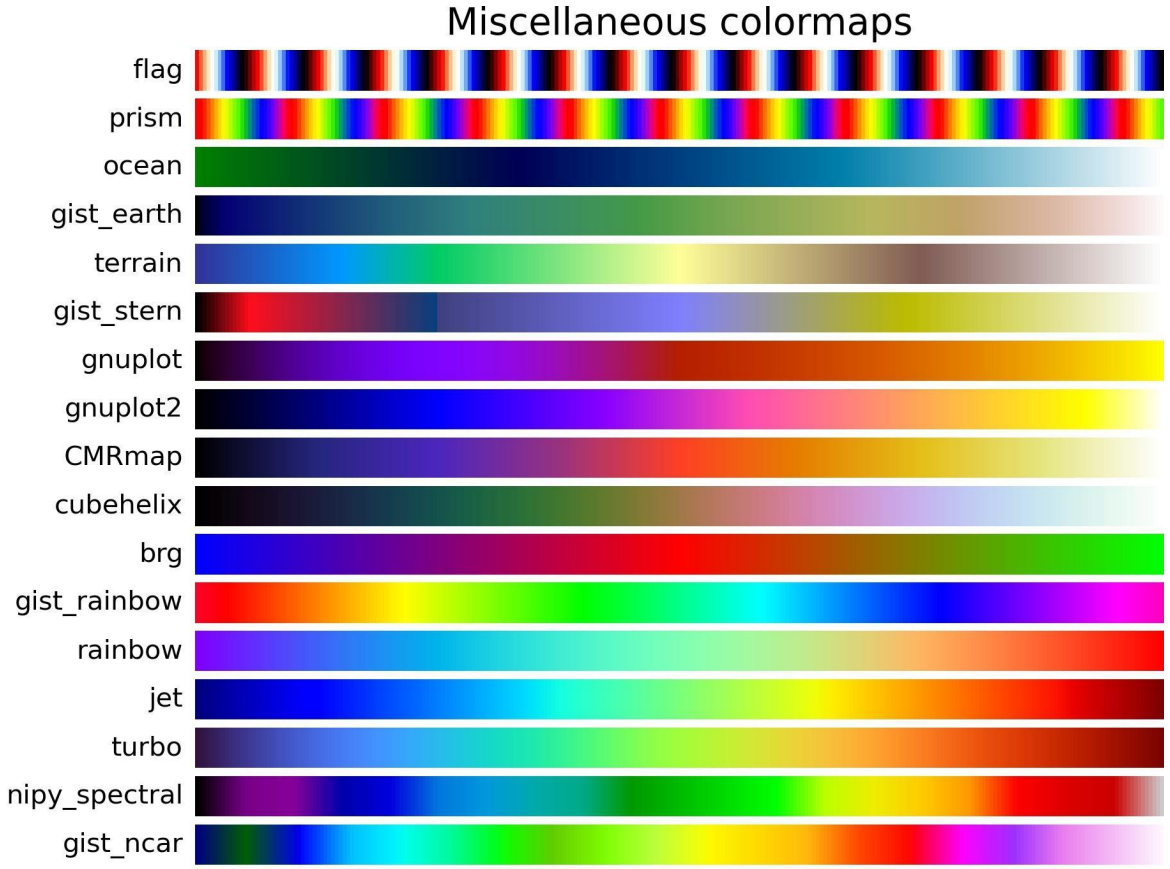
Ou, por meio de paletas:

https://matplotlib.org/stable/tutorials/colors/colormaps.html




Sequential (2) colormaps
binary
gist_yarg
gist_gray
gray
bone
pink
spring
summer
autumn
winter
cool
Wistia
hot
afmhot
gist_heat
copper
Diverging colormaps
PiYG
PRGn
BrBG
PuOr
RdGy
RdBu
RdYlBu
RdYlGn
Spectral
coolwarm
bwr
seismic
Cyclic colormaps
twilight
twilight_shifted
hsv


Qualitative colormaps
Pastel1
Pastel2
Paired
Accent
Dark2
Set1
Set2
Set3
tab10
tab20
tab20b
tab20c


Miscellaneous colormaps
flag
prism
ocean
gist_earth
terrain
gist_stern
gnuplot
gnuplot2
CMRmap
cubehelix
brg
gist_rainbow
rainbow
jet
turbo
nipy_spectral
gist_ncar

Funções em Python

Leitura recomendada:
https://numpy.org
https://docs.python.org/3/library/os.html
https://docs.python.org/3/library/functions.html

Uso da função linspace

Gera 11 elementos, começando em 0 e terminando em 5 com intervalo de 0.5.

```
import numpy as np
import math

x = np.linspace(0, 5, 11)
print(x)  # [0.  0.5  1.  1.5  2.  2.5  3.  3.5  4.  4.5  5.]
```

Gera 5 elementos, começando em 0 e terminando em 10 com intervalo de 2.5.

```
x = np.linspace(0, 10, 5)
print(x)  # [0.  2.5  5.  7.5  10.]
```

Uso da função random.random

Gera números aleatórios.

```
x = np.random.random(5)
print(x)  # [0.14640476 0.81601629 0.85883815 0.78358921 0.99644288]
```

Uso da função type

Exibe o tipo de dado utilizado.

```
x = [1, 2., 3, 4]
print(type(x))  # int64
```

Uso da função arange

Gera valores entre 0 e 19. O valor 20 não é gerado, é uma característica do Python. E como fazer para o 20 ser impresso? Basta colocar o valor 21 no lugar do 20.

```
x = np.arange(20)
print(x)  # [0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

Gera valores entre 0 e 100 com passo ou incremento 10. O valor 100 não é gerado. E como fazer para o 100 ser impresso? Basta colocar o valor 110 no lugar do 100.

```
x = np.arange(0, 100, 10)
print(x)  # [ 0 10 20 30 40 50 60 70 80 90]
```

Gera valores entre 1 e 12. O valor 13 não é gerado.

```
x = np.arange(1, 13)
print(x)  # [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

Uso da função sort

Ordena o arranjo de forma crescente.

```
x = [10, 4, 50, 1, 15]
x.sort()  # Primeiro, ordena a lista.
print(x)  # [1, 4, 10, 15, 50]
```

Ordena o arranjo de forma decrescente.

```
x = [10, 4, 50, 1, 15]
x.sort(reverse=True)
print(x)  # [1, 4, 10, 15, 50]
```

Uso da função dtype

Exibe o tipo do dado.

```
x = np.array([-10, 10, 20, 30])
print(x.dtype)  # int64
```

Uso da função math.ceil

Arredondamento para o inteiro mais próximo.

```
print(math.ceil(1.1))   # 2
print(math.ceil(5.3))   # 6
print(math.ceil(-5.6))  # -5
print(math.ceil(22.4))  # 23
print(math.ceil(10.0))  # 10
```

Uso da função math.floor

Arredondamento para o menor inteiro mais próximo.

```
print(math.floor(1.1))   # 1
print(math.floor(5.3))   # 5
print(math.floor(-5.6))  # -6
print(math.floor(22.4))  # 22
print(math.floor(10.0))  # 10
```

Uso da função len

Retorna o número de elementos.

```
x = [1, 2, 3, 4, 5]
print(len(x))  # 5

x = [[1, 2, 3, 4, 5],[6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
print(len(x))  # 3
```

Uso da função mean

Média dos valores.

```
x = [1, 2, 3, 4, 5]
y = np.mean(x)
print(y)  # 3.0
```

Uso da função round

Outra forma de realizar arredondamento utilizando Numpy.

```
import numpy as np

print(round(3.2))  # 3
print(round(7.5))  # 8
print(round(2.4))  # 2

#  Especificando o número de casas decimais.

print(round(3.25,1))  # 3.2
print(round(7.516,2))  # 7.52
print(round(2.458,2))  # 2.46
```

Uso da função std

Desvio padrão dos valores.

```
x = [[1, 2, 3, 4, 5]]
y = np.std(x)
print(y)  # 1.4142135623730951
```

Uso da função upper

Transforma toda a string em maiúscula.

```
x = 'egUa, mAno!'
x = x.upper()
print(x)  # EGUA, MANO!
```

Uso da função lower

Transforma toda a string em minúscula.

```
x = x.lower()
print(x)  # egua, mano!
```

Uso da função title

Altera apenas a primeira letra de cada palavra para maiúscula.

```
x = 'egUa, mAno!'
x = x.title()  # Egua, Mano!
print(x)
```

Uso da função split

Divide a string.

```
x = x.split()
print(x)  # ['egua,', 'mano!']
```

Interagir com o sistema operacional

Leitura recomendada:
https://docs.python.org/3/library/os.html

Visualizar os métodos do os

```
import os


print(dir(os))
```

Retornar o caminho do diretório corrente

```
x = os.getcwd()
print(x)
```

Remover um diretório

```
os.rmdir('nome_diretorio')
```

Criar um diretório

```
os.mkdir('nome_diretorio')
```

Listar o conteúdo do diretório corrente

```
x = os.listdir()
print(x)
```

Remover arquivo

```
os.remove('nome_arquivo')
```

Renomear arquivo

```
os.rename('nomeantigo','nomenovo')
```

Funções lambdas

Conhecidas por expressões Lambdas, ou simplesmente Lambdas, são funções sem nome, ou seja, funções anônimas.

Em funções Python existe a possibilidade de que não se tenha nenhuma ou várias entradas. Em Lambdas ocorre o mesmo.

Leitura recomendada:
https://docs.python.org/3.9/tutorial/controlflow.html?highlight=lambda#lambda-expressions

Representação genérica da função lambda:

```
n = lambda: x1, x2, ..., xn: <expressao>
```

Exemplo de uso da função lambda: Converte a temperatura de Kelvin para Celsius.

```
# Função lambda que converte de Kelvin para Celsius.
tc = lambda tk: tk - 273.15

# O valor de 300 K será convertido para Celsius.
print(tc(300))  # 26.850000000000023
```

Funções integradas

Map

Com o map é possível realizar o mapeamento de valores para função.

O map (mapeia uma função para um iterável) é uma função que recebe dois parâmetros: o primeiro, é a função, o segundo, um iterável, o resultado do map é do tipo map object. Os resultados do map são colocados em um map object que pode ser convertido para uma lista, tupla, dicionário, depende do interesse do usuário.

Observação: Após a primeira utilização da função map, o resultado é zerado e essa característica é interessante porque limpa a memória da máquina.

Leitura recomendada:
https://docs.python.org/3.9/library/functions.html#map

Exemplo de uso do map: Converte a temperatura de Kelvin para Celsius.

1) Utilizando a função integrada map com lambda (função sem nome).

```
# Lista de valores de temperatura em Kelvin.
temp_kelvin = [300, 280, 350, 315]

tc = map(lambda tk: tk-273.15, temp_kelvin)
# Para visualizar os valores tem que converter map para list.
print(list(tc))
```

2) Utilizando map com função.

```
# Definindo a função que converte de Kelvin para Celsius.
def kelvin_to_celsius(tk):
    return tk-273.15

tc = map(kelvin_to_celsius, temp_kelvin)
print(list(tc))
```

Filter

Serve para filtrar dados de uma determinada coleção.

Assim como a função map(), o filter() recebe dois parâmetros, isto é, uma função e um iterável.

Leitura recomendada:
https://docs.python.org/3.9/library/functions.html#filter

Qual a diferença entre map e filter?

1) O map recebe dois parâmetros, uma função e um iterável e retorna um objeto mapeando a função para cada elemento do iterável. O map retorna outros valores que não sejam booleanos. A função será aplicada nos dados e retornará um valor.
2) O filter recebe dois parâmetros, uma função e um iterável e retorna um objeto filtrando apenas os elementos de acordo com a função. O filter retorna valores booleanos (True ou False), isso faz com que o dado seja ou não selecionado.

Observação: Assim como na função map(), após serem utilizados os dados de filter(), eles são excluídos da memória.

Se colocar mais um print, a lista estará vazia, o valor fica na memória para ser utilizado apenas uma vez.

Exemplo de uso do filter: Filtra valores específicos de temperatura.

1) A variável “valor” representa cada valor de temperatura da lista "temp_celsius". A comparação "valor > media" gera True ou False. Caso seja True, compara com os valores da variável "temp_celsius" com a "media", e depois envia o resultado numérico para função lambda (função sem nome).

```
import statistics

# Lista de valores de temperatura em Celsius.
temp_celsius = [26.8, 33.2, 15.0, 35.4]

# Calculando a média dos dados utilizando a função mean().
media = statistics.mean(temp_celsius)
print(media)  # 27.6

# Compara os valores de temperatura com a "media", e caso o valor seja
# maior que a "media", armazena o resultado em "valor".
tc_filtrada = filter(lambda valor: valor > media,temp_celsius)
print(list(tc_filtrada))  # [33.2, 35.4]
```

Zip

O zip() cria um iterável (Zip Object) que agrega elementos de cada um dos iteráveis passados como entrada em pares.

Leitura recomendada:
https://docs.python.org/3.9/library/functions.html#zip

Exemplo de uso do zip:

1) São fornecidas duas listas, uma com o nome dos biomas e outra, com a quantidade de focos de queimadas. O objetivo consiste em visualizar o nome dos biomas e os seus respectivos valores.

```
bioma = ['Amazônia', 'Cerrado', 'Pantanal']
numero_focos_queimadas = [6803, 5663, 1532]

# bioma =                  ['Amazônia', 'Cerrado', 'Pantanal']
# numero_focos_queimadas = [6803      , 5663     , 1532]
# Índice                     0          1          2

ret = zip(bioma, numero_focos_queimadas)
print(list(ret))  # [('Amazônia', 6803), ('Cerrado', 5663), ('Pantanal', 1532)]
```

2) São fornecidas as quantidades de focos de queimadas (para dois períodos distintos) para três biomas brasileiros. O objetivo consiste em retornar o maior valor entre dois períodos para cada bioma. Por exemplo, para o bioma Amazônia, o maior valor entre 6803 e 5803 é 6803. O mesmo raciocínio é aplicado para os demais biomas.

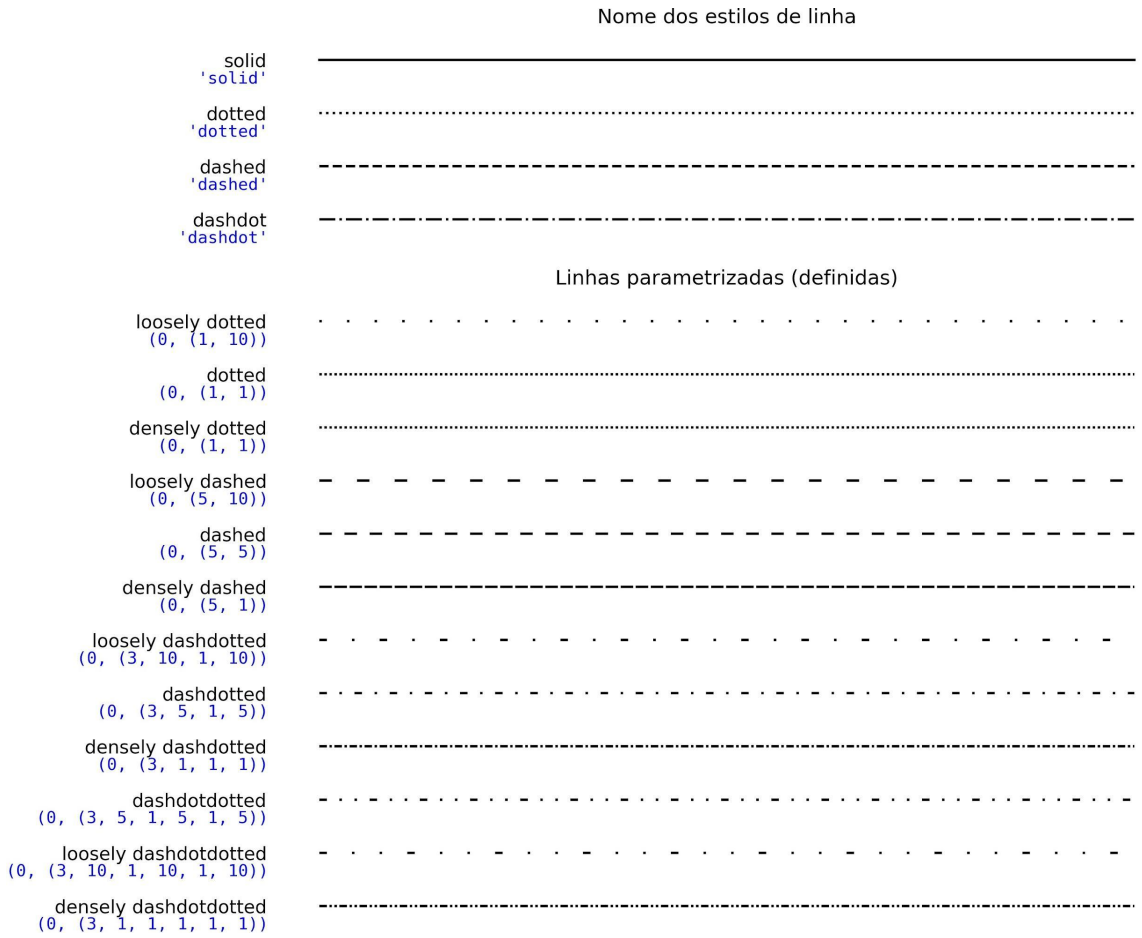
```
bioma = ['Amazônia', 'Cerrado', 'Pantanal']
nfocos1 = [6803, 5663, 1532]  # Ano 2019
nfocos2 = [5803, 4663, 1084]  # Ano 2020

ret = {dados[0]: max(dados[1], dados[2]) for dados in zip(bioma,nfocos1,nfocos2)}
print(ret)
```

Estilos de linha

Leitura recomendada:
https://matplotlib.org/stable/gallery/lines_bars_and_markers/linestyles.html

Como usar? Pode-se utilizar o nome `linestyle='-'` ou `ls='-'`.

- `linestyle='-' ou linestyle=solid'`
- `linestyle=':' ou linestyle=dotted'`
- `linestyle='--' ou linestyle='dashed'`
- `linestyle='-.' ou linestyle='dashdot'`

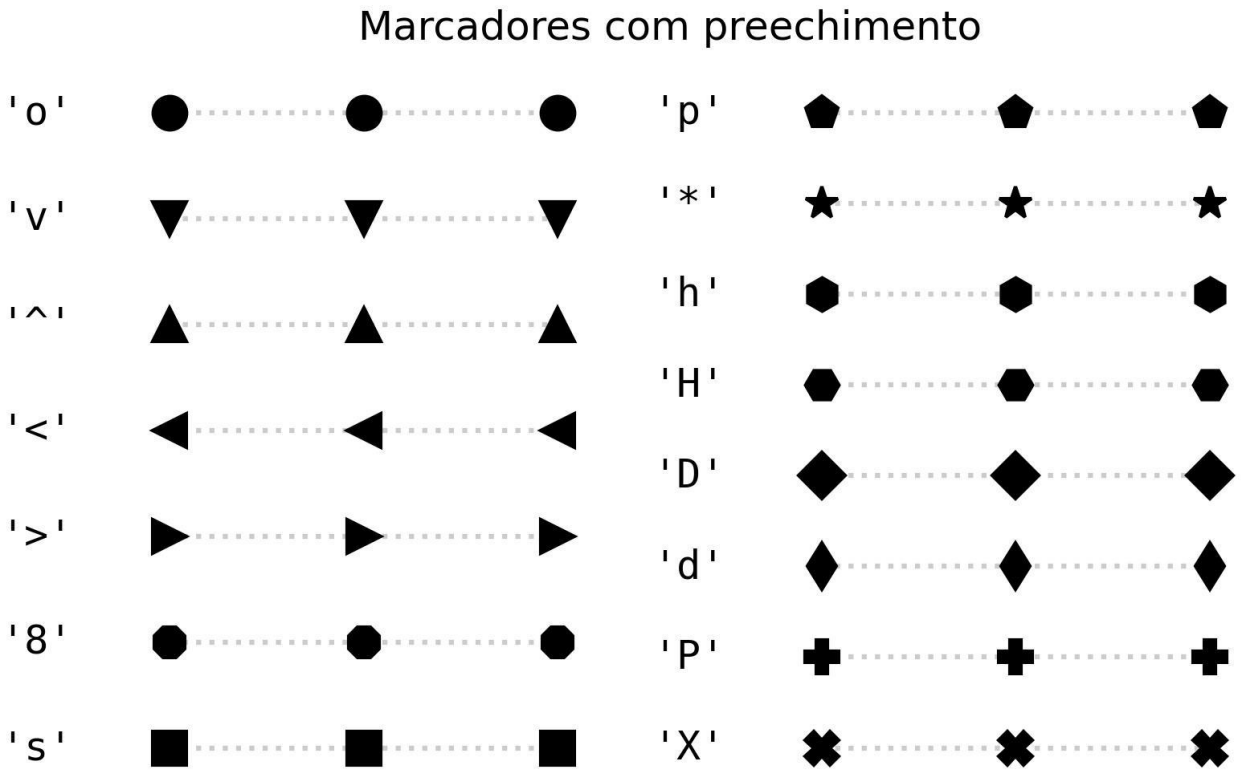


Fonte: https://matplotlib.org/stable/gallery/lines_bars_and_markers/linestyles.html

Estilo de marcadores

Como usar? `marker='o'`

Fonte: https://matplotlib.org/stable/gallery/lines_bars_and_markers/marker_reference.html



Fonte: https://matplotlib.org/stable/gallery/lines_bars_and_markers/marker_reference.html

Leitura recomendada:
https://matplotlib.org/stable/api/markers_api.html

https://matplotlib.org/stable/gallery/lines_bars_and_markers/marker_reference.html

- Tamanho do marcador da linha

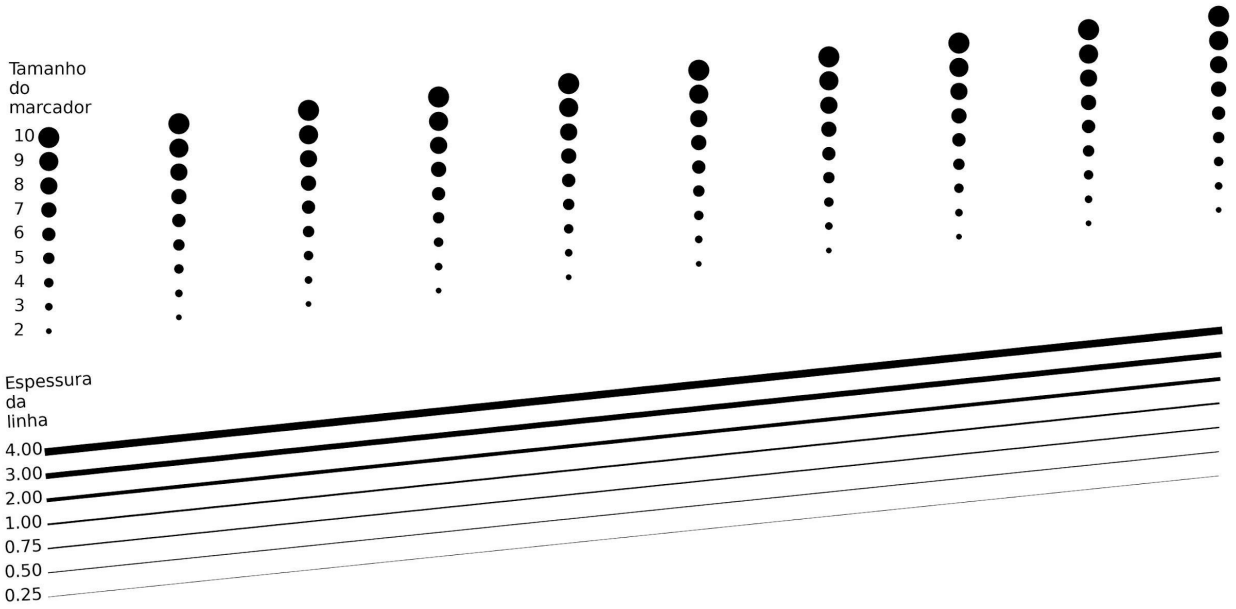
Como usar? `markersize=12` ou `ms=12`.

O valor zero não mostra nenhum marcador, e quanto maior esse valor, maior será o marcador.

Espessura da linha

Como usar? `linewidth=1` ou `lw=1`.

O valor zero não mostra nenhuma linha, e quanto maior esse valor, mais espessa será a linha.



Fonte: https://www.southampton.ac.uk/~feeg1001/notebooks/Matplotlib.html

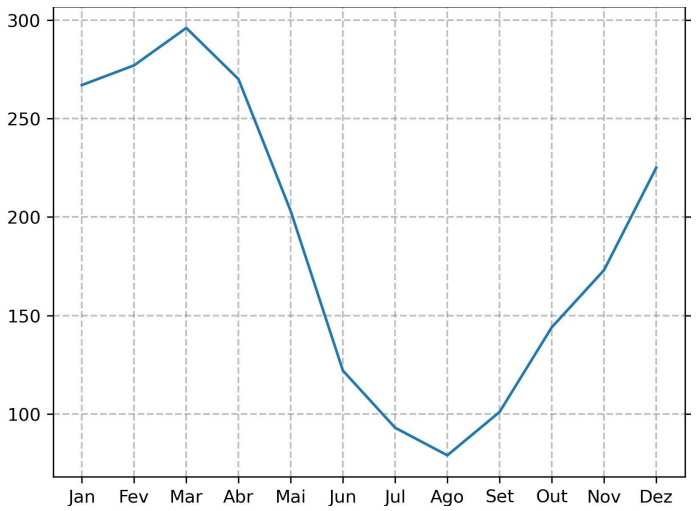
Mostrar a linha de grade do gráfico

É possível visualizar no gráfico as linhas de grade horizontal e vertical como também é possível personalizá-las.

Exemplo.

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('precipitacao.txt', sep='\t',
                 names=['Mês', 'Climatologia', '2019', '2020'])

x = df['Mês']
y = df['Climatologia']

plt.plot(x, y)

# Visualiza as grades horizontal e vertical.
# alpha = transparência das linhas, lw = espessura das linhas e
# color = cor das linhas.
# Opções: which='major' ou 'minor', axis='x' ou 'y' ou 'both'.
plt.grid(alpha=0.5, ls='--', lw=1, color='gray')

plt.tick_params(axis='y', right=True)

plt.savefig('ex01.jpg', transparent=True, dpi=300,
            bbox_inches='tight', pad_inches=0)
```

O resultado será:



Galeria de gráficos e mapas

Leitura recomendada:
https://matplotlib.org/stable/gallery/index.html
https://proplot.readthedocs.io/en/stable/index.html#

Download de todos os scripts:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos

Tipos de gráficos

Nesta seção diferentes gráficos serão apresentados, bem como os scripts que os geraram. Os comentários foram adicionados no corpo do próprio script e sendo necessário, informações adicionais para melhor o seu entendimento serão incluídos.

Diagrama de Taylor

Instalar a biblioteca:

```
pip install SkillMetrics
```

Leitura recomendada:
https://pypi.org/project/SkillMetrics/
https://github.com/PeterRochford/SkillMetrics/wiki
https://github.com/PeterRochford/SkillMetrics/wiki/Taylor-Diagram-Options
https://cdat.llnl.gov/Jupyter-notebooks/vcs/Taylor_Diagrams/Taylor_Diagrams.html

Link para o script:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/diagrama_taylor

O resultado será:

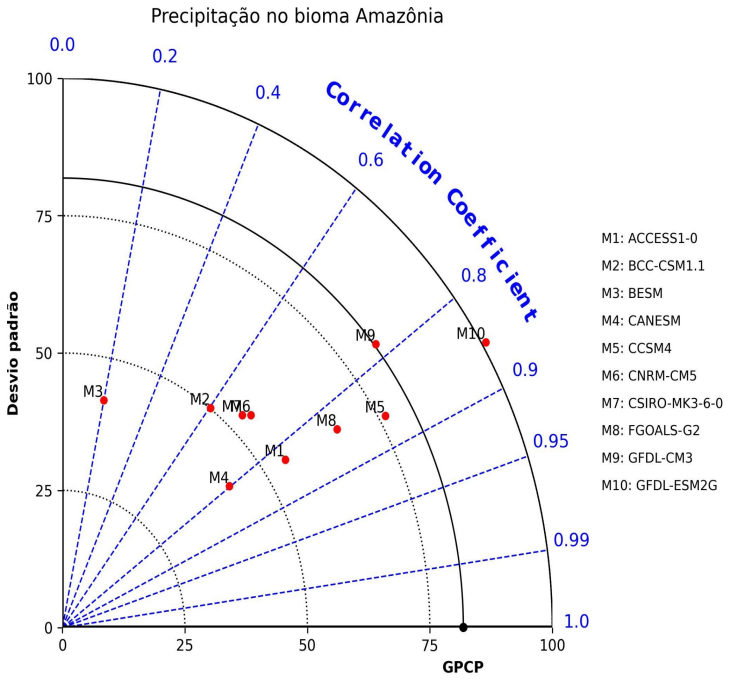


Gráfico de barra

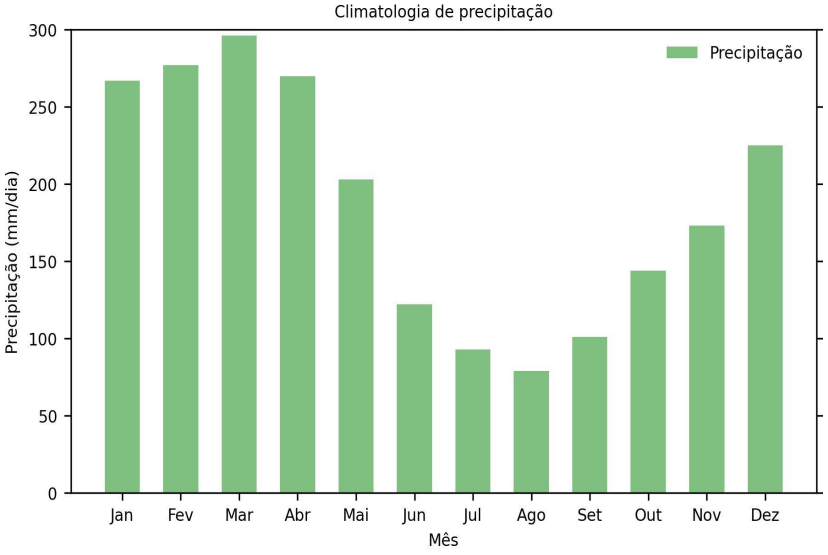
Leitura recomendada:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.colorbar.html
https://matplotlib.org/stable/gallery/ticks_and_spines/multiple_yaxis_with_spines.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.legend.html

Todos os scripts podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/grafico_barras

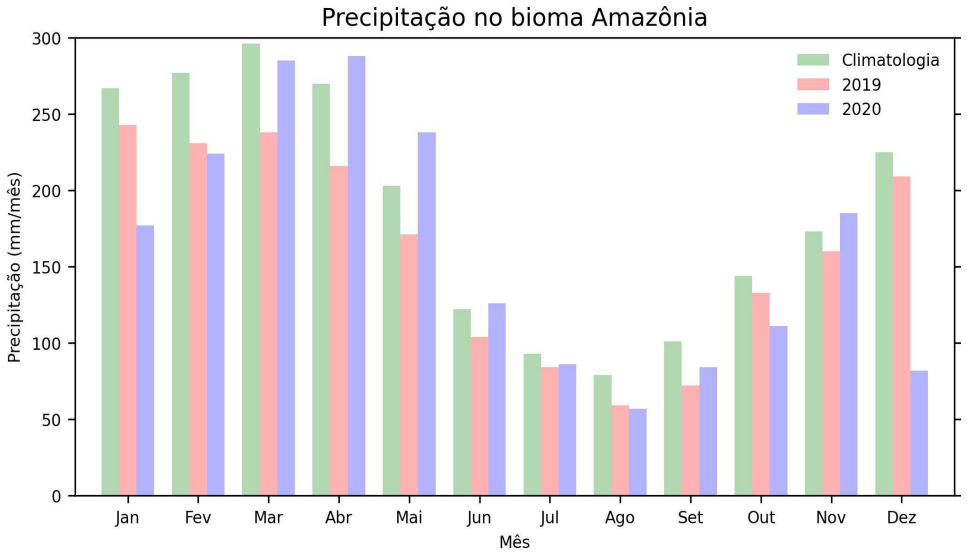
Exemplo 1: Script ex01.py.

O resultado será:



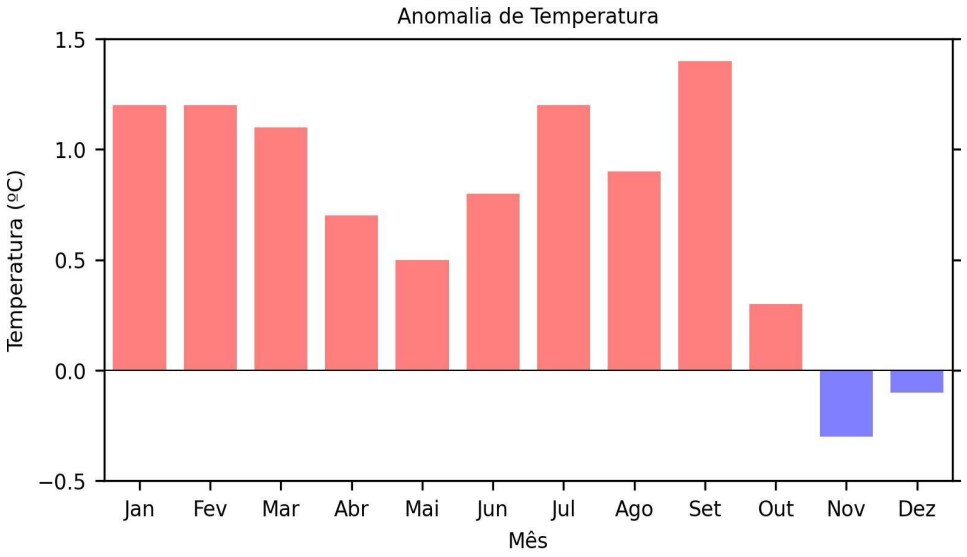
Exemplo 2: Script ex02.py.

O resultado será:



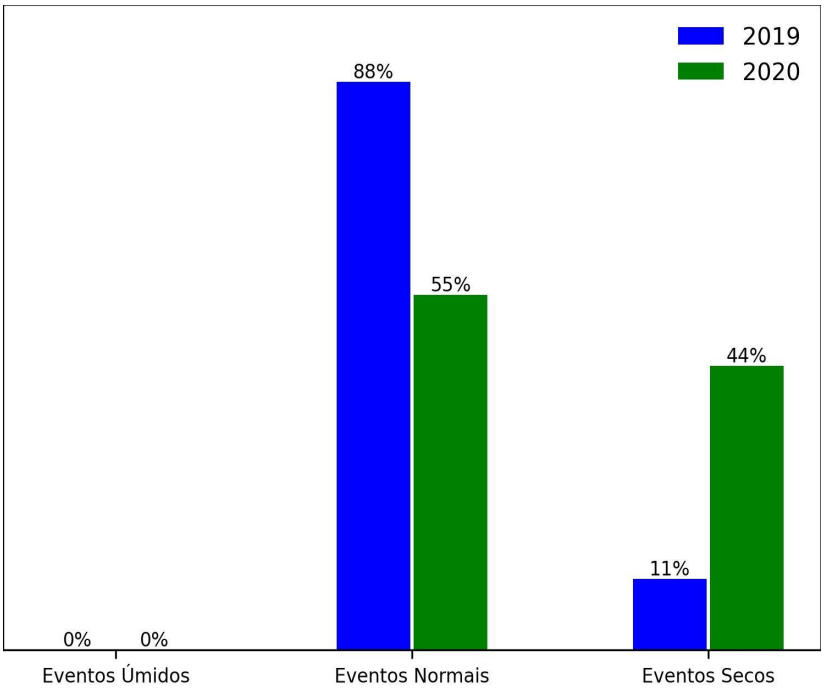
Exemplo 3: Script ex03.py.

O resultado será:



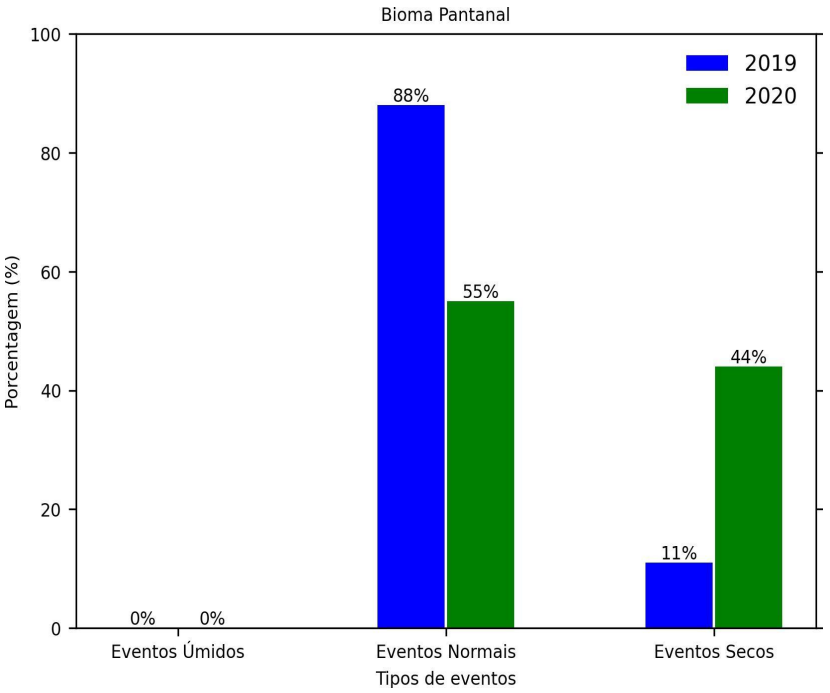
Exemplo 4: Script ex04.py.

O resultado será:


2019
2020
88%
55%
44%
11%
0%
0%
Eventos Úmidos
Eventos Normais
Eventos Secos

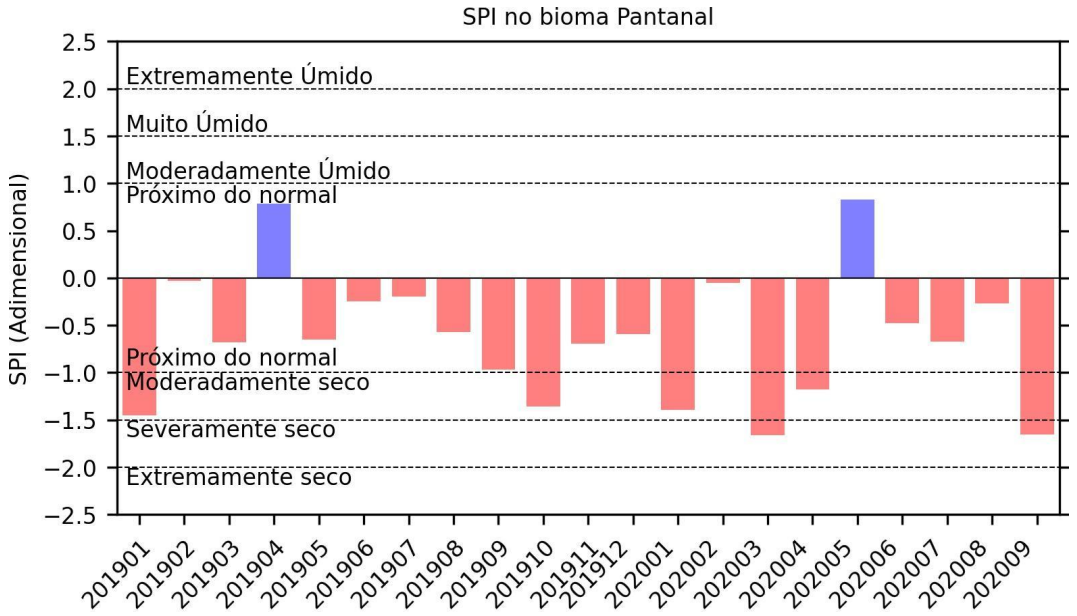
Exemplo 5: Script ex05.py.

O resultado será:



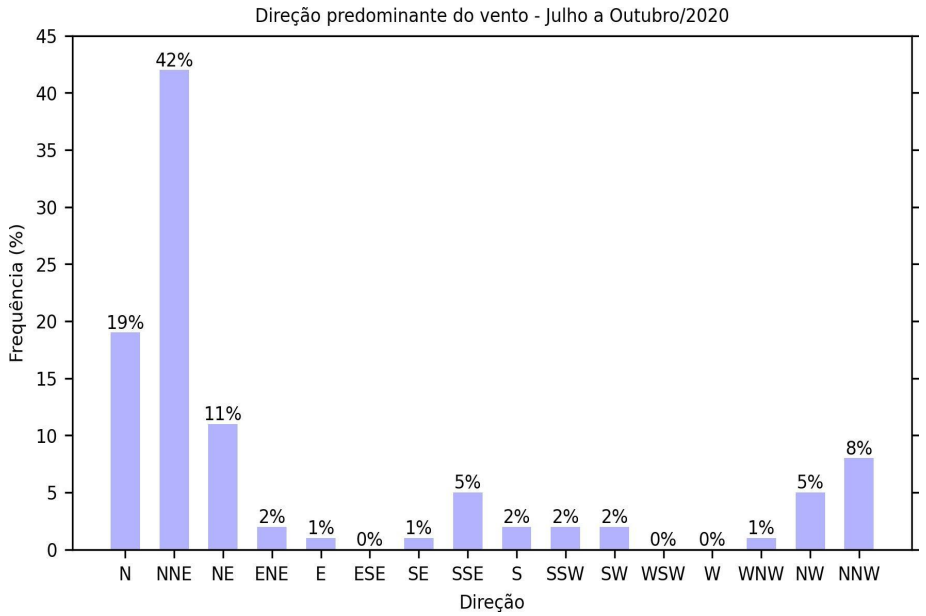
Exemplo 6: Script ex06.py.

O resultado será:



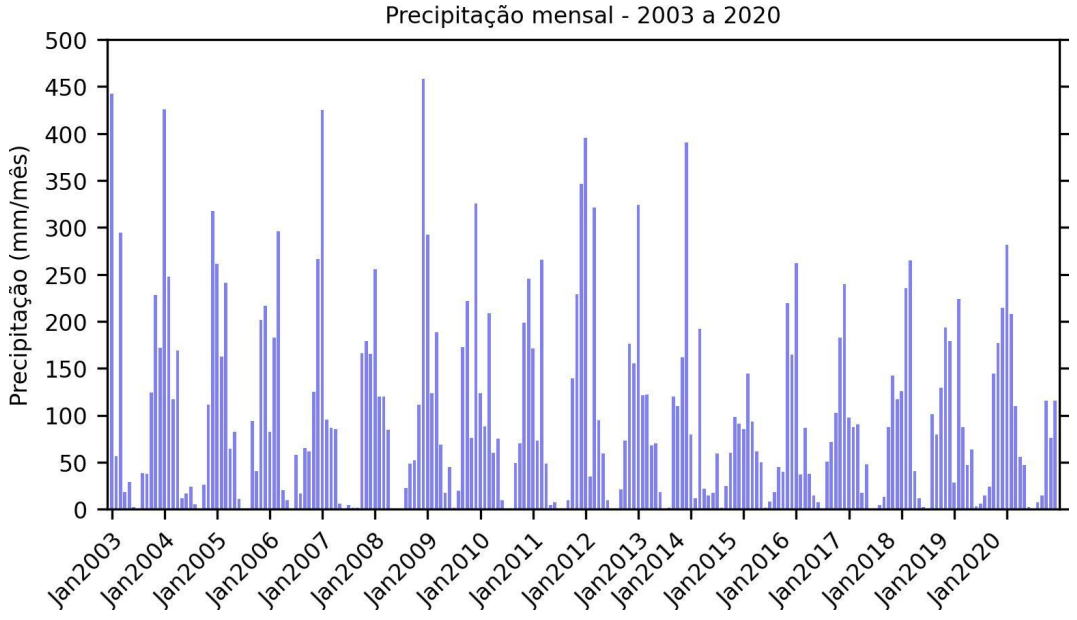
Exemplo 7: Script ex07.py.

O resultado será:



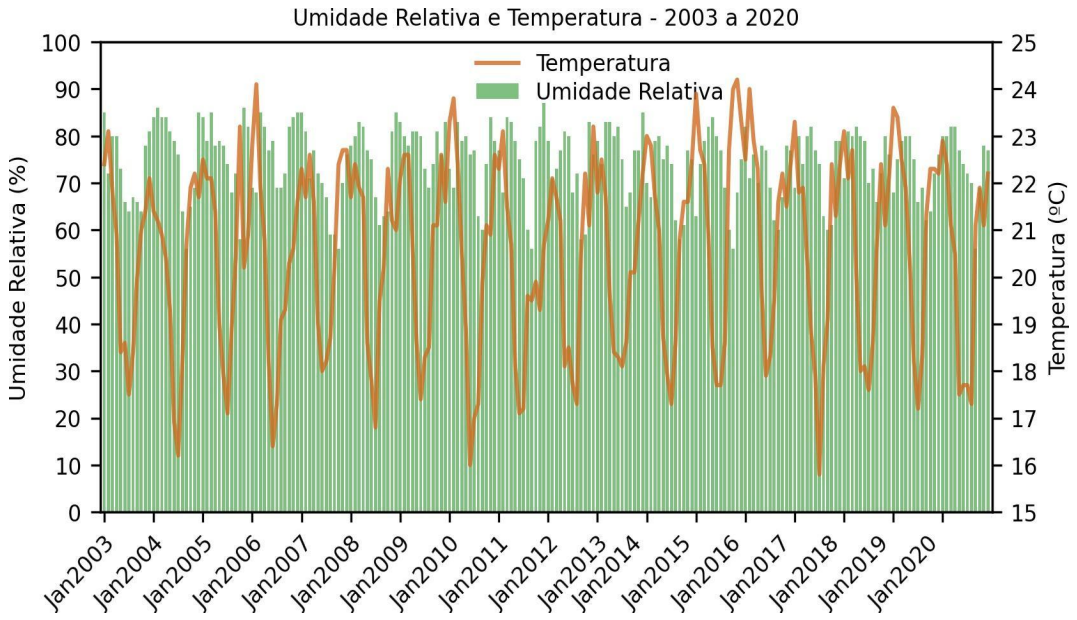
Exemplo 8: Script ex08.py.

O resultado será:



Exemplo 9: Script ex09.py.

O resultado será:



Exemplo 10: Script ex10.py.

O resultado será:

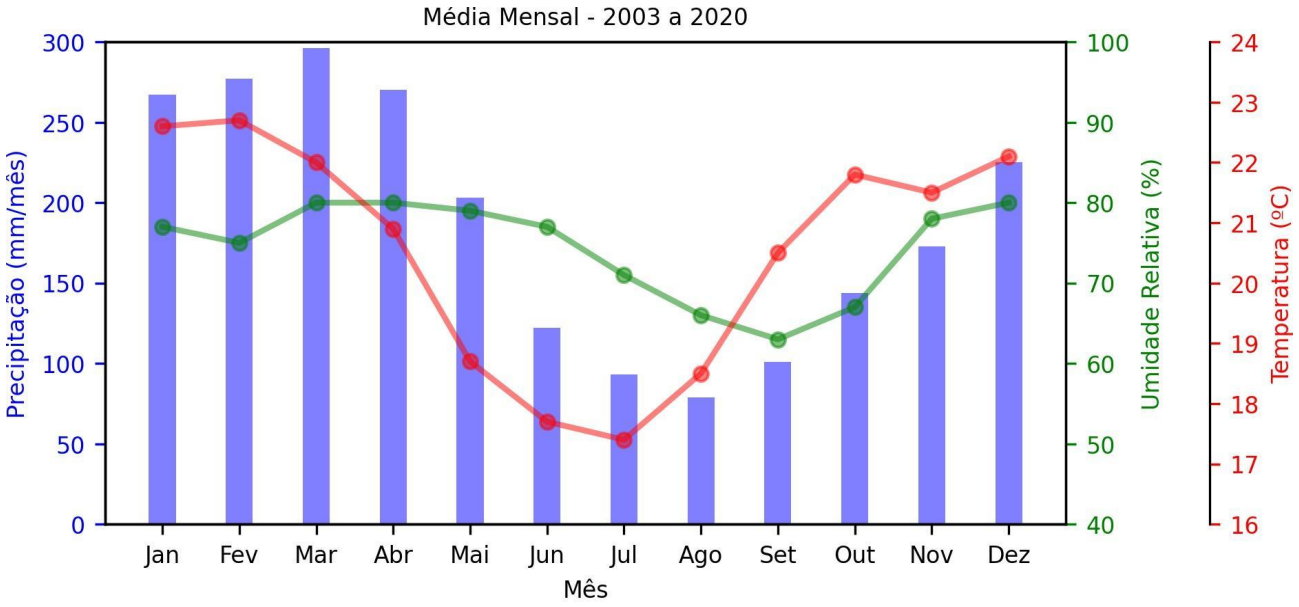


Gráfico estatístico

Histograma

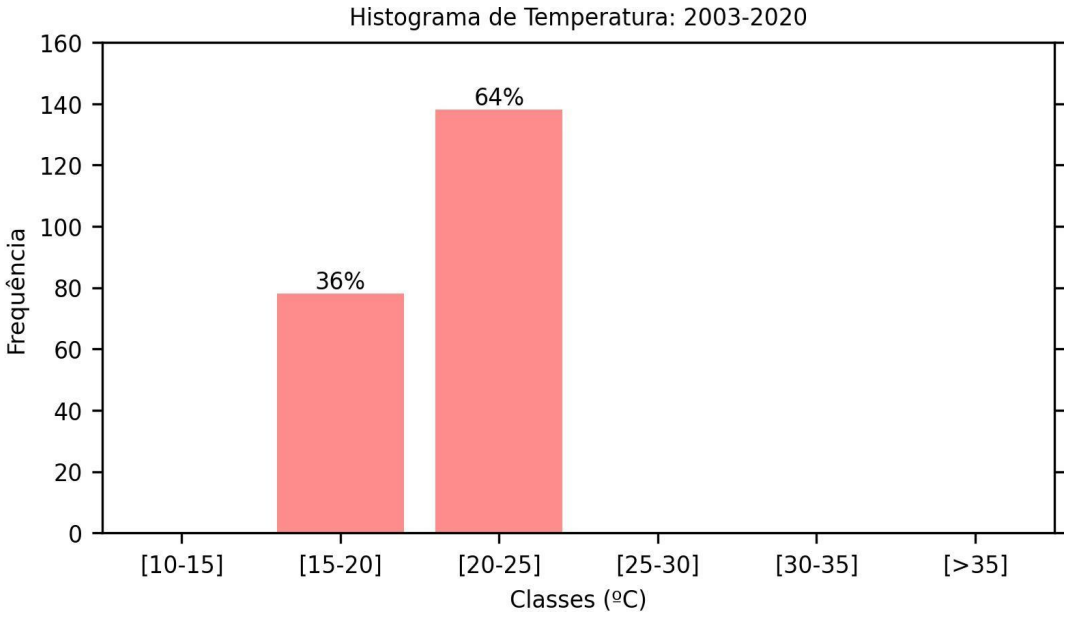
Leitura recomendada:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
https://matplotlib.org/stable/gallery/statistics/hist.html#sphx-glr-gallery-statistics-hist-py
https://matplotlib.org/stable/gallery/statistics/histogram_histtypes.html#sphx-glr-gallery-statistics-histogram-histtypes-py
https://matplotlib.org/stable/gallery/statistics/histogram_features.html#sphx-glr-gallery-statistics-histogram-features-py

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/estatisticos/histograma

Exemplo 1: Script ex01.py.

O resultado será:



Boxplot

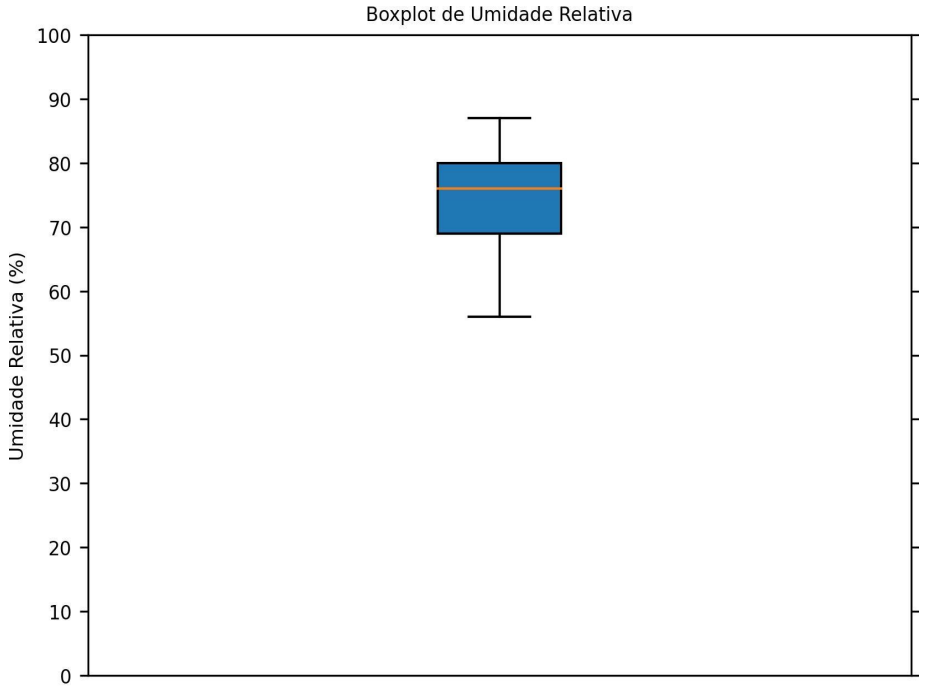
Leitura recomendada:
https://matplotlib.org/stable/gallery/statistics/boxplot_color.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html
https://matplotlib.org/stable/gallery/statistics/boxplot_color.html#sphx-glr-gallery-statistics-boxplot-color-py

Todos os scripts podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/estatisticos/boxplot

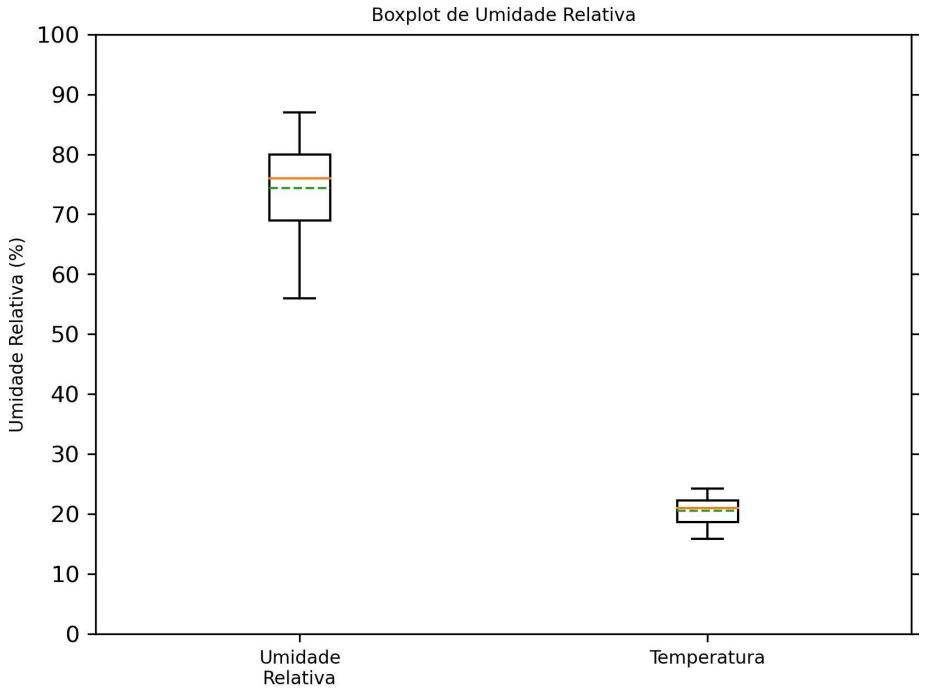
Exemplo 1: Script ex01.py.

O resultado será:



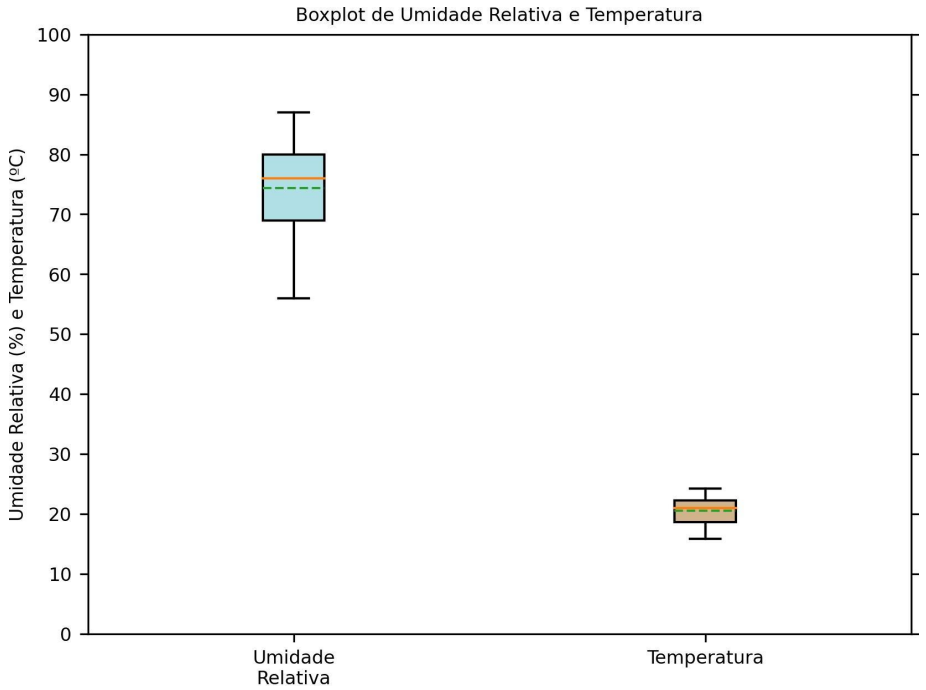
Exemplo 2: Script ex02.py.

O resultado será:



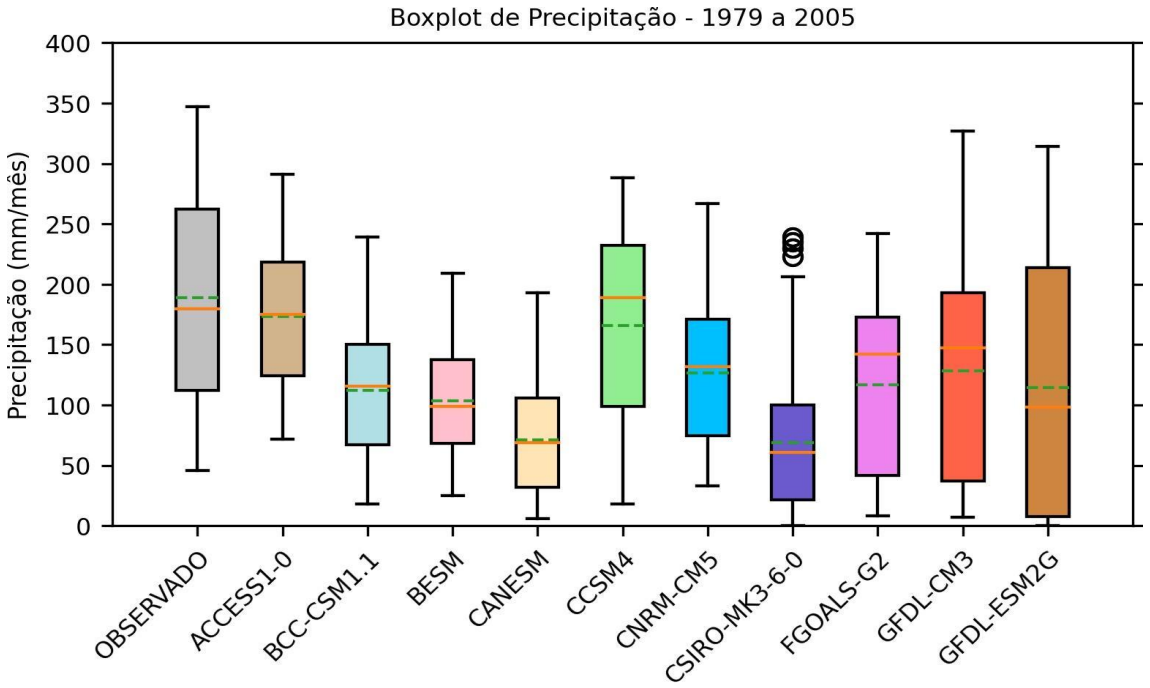
Exemplo 3: Script ex03.py.

O resultado será:



Exemplo 4: Script ex04.py.

O resultado será:



Regressão Linear

Leitura recomendada:
https://scipy-lectures.org/packages/scikit-learn/index.html
https://scipy-lectures.org/packages/scikit-learn/index.html#introducing-the-scikit-learn-estimator-object

Não esquecer de instalar a biblioteca abaixo:

```
pip install yellowbrick
```

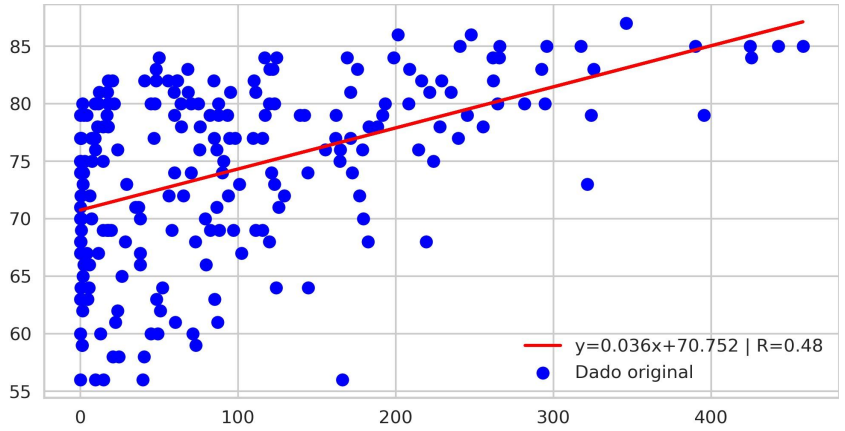
Todos os scripts podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/estatisticos/regressao

Exemplo 1: Script ex01.py.

O resultado será:

Primeiro gráfico:



Segundo gráfico:

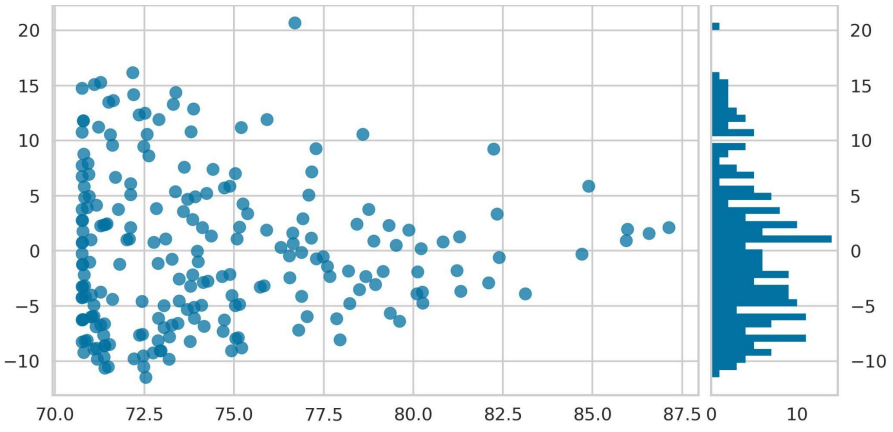


Gráfico espaciais

Para esta seção de gráficos espaciais é altamente recomendado criar um ambiente virtual como descrito no sumário “Instalação do ambiente virtual” para gerar as figuras abaixo. Nesta seção será utilizada a biblioteca proplot para gerar os mapas por se tratar de um pacote mais simples para geração de mapas do que o matplotlib. Além disso, é necessário instalar os pacotes abaixo:

```
1. pip install proplot
2. python -m pip install xarray
3. pip install cartopy
```

Campo escalar

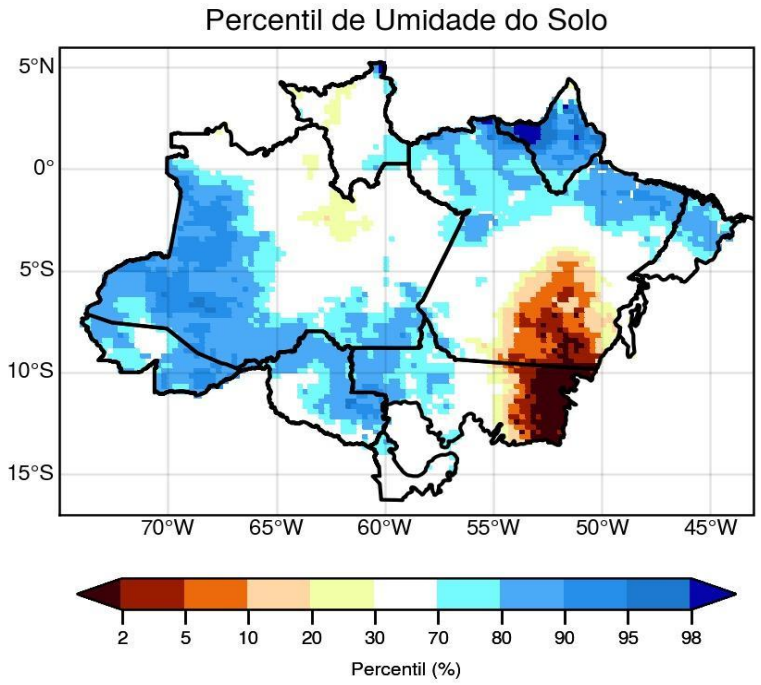
Leitura recomendada:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pcolormesh.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.contourf.html

Todos os scripts podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/grafico_espacial/campos_escalares

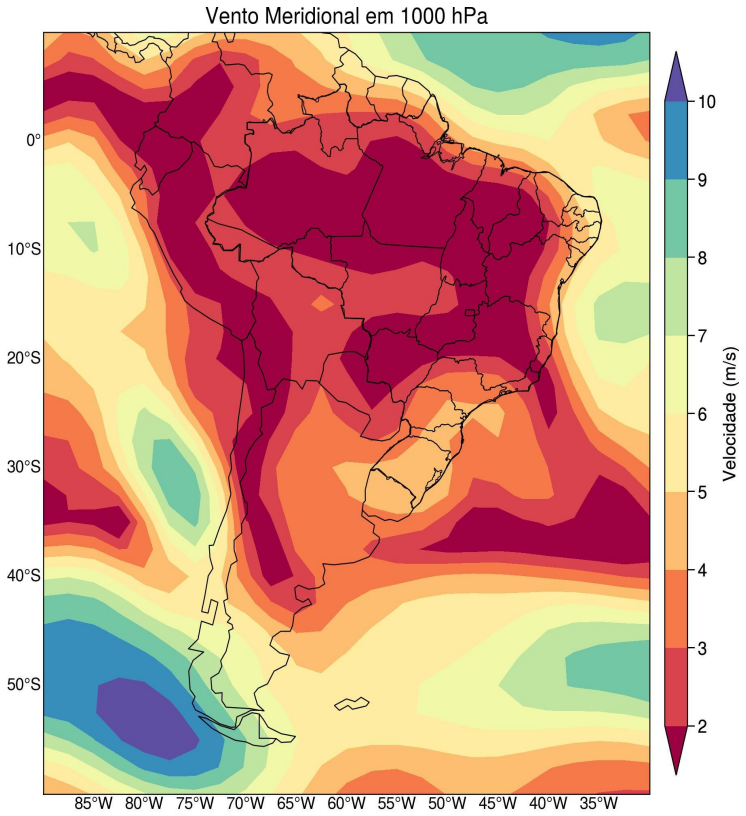
Exemplo 1: Script ex01.py.

O resultado será:



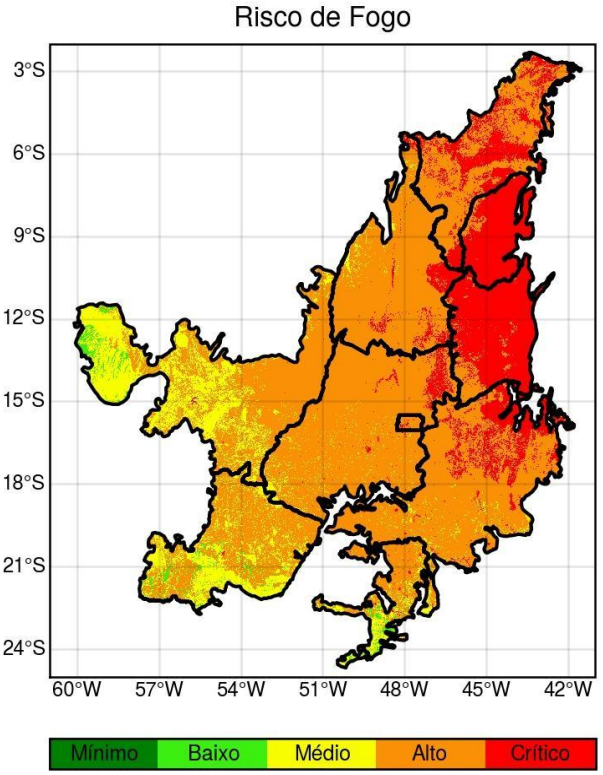
Exemplo 2: Script ex02.py.

O resultado será:



Exemplo 3: Script ex03.py.

O resultado será:



Campo vetorial

Leitura recomendada:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.barbs.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.quiverkey.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.quiver.html

Gráfico de barbela do vento

Todos os scripts podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/grafico_espacial/campos_vetoriais

Exemplo 1: Script ex01.py.

O resultado será:

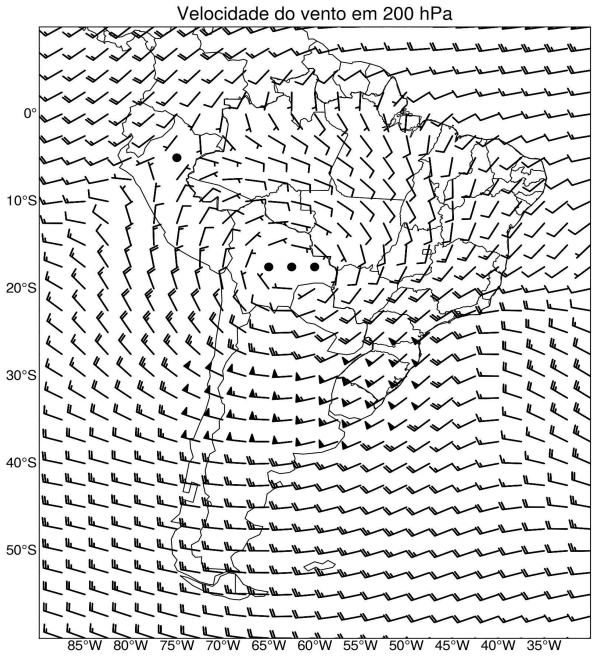
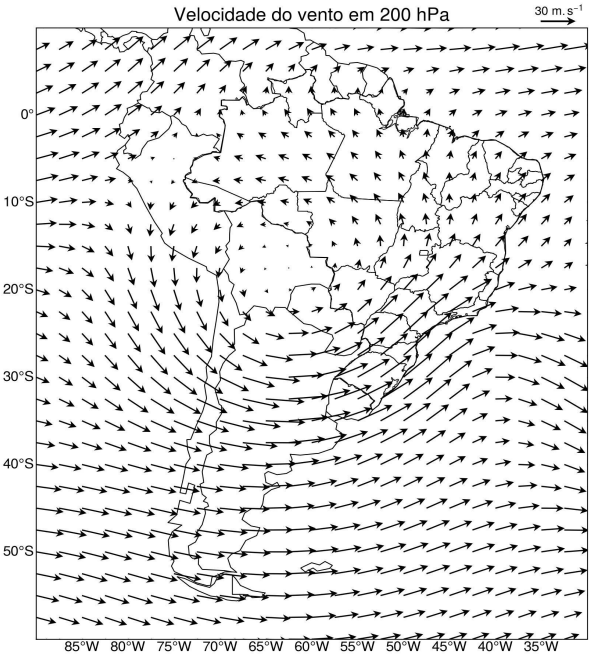


Gráfico de vetor do vento

Exemplo 2: Script ex02.py.

O resultado será:



Exemplo 3: Script ex03.py.

O resultado será:

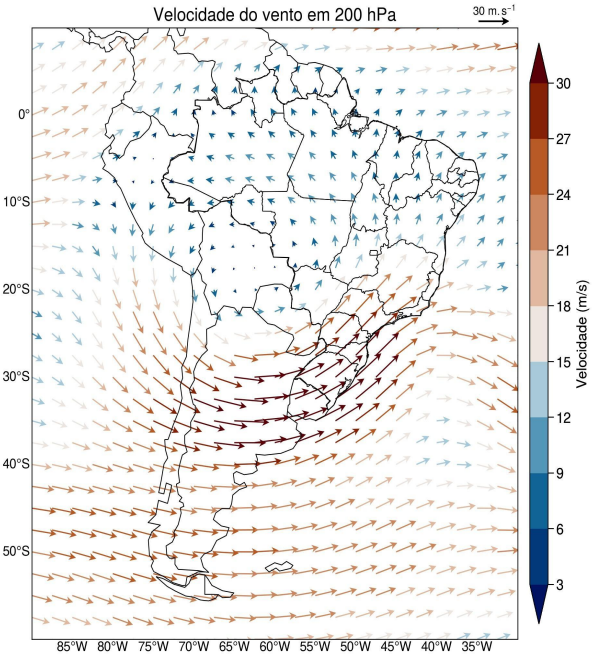


Gráfico de dispersão

Leitura recomendada:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html
https://matplotlib.org/stable/gallery/index.html?highlight=scatter%20plot

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/grafico_dispersao

Exemplo 1: Script ex01.py.

O resultado será:

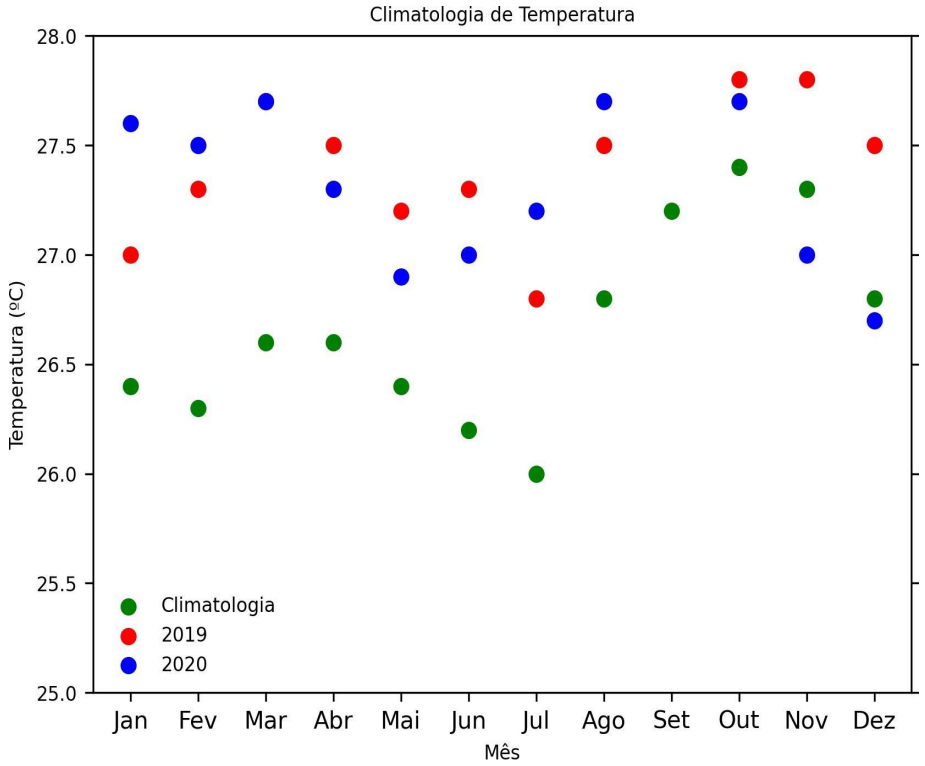


Gráfico de linha

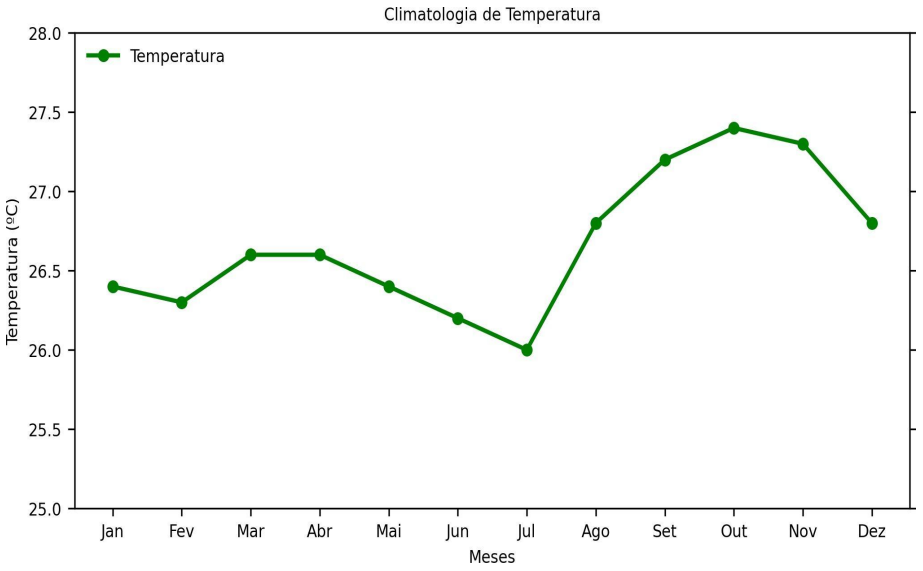
Leitura recomendada:
https://matplotlib.org/stable/api/axes_api.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Todos os scripts podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/grafico_linha

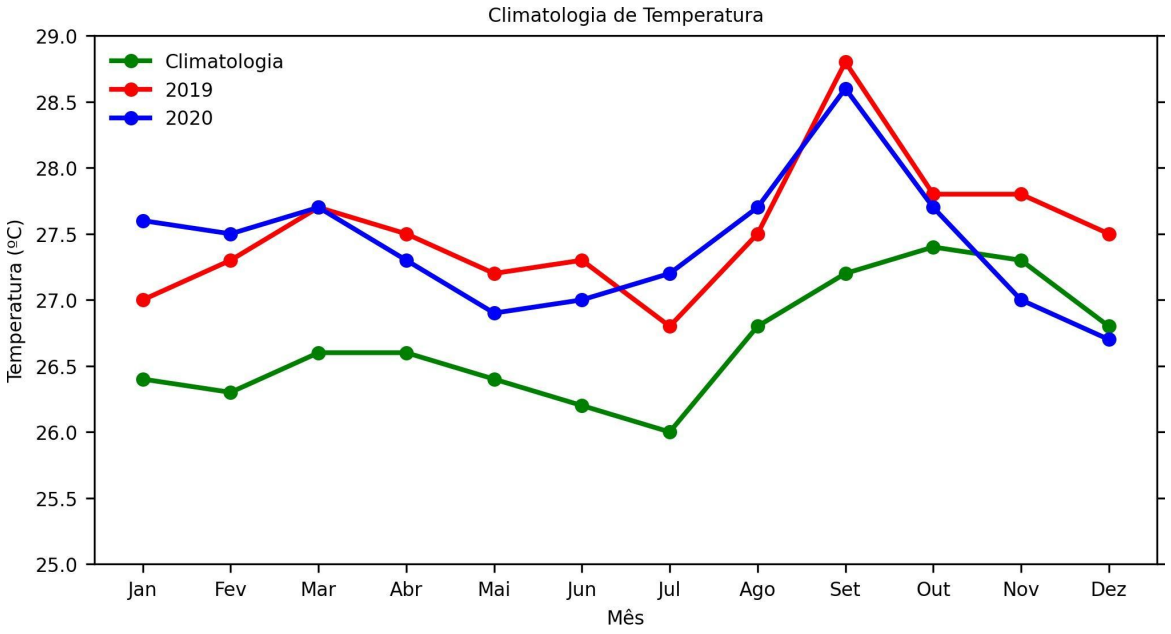
Exemplo 1: Script ex01.py.

O resultado será:



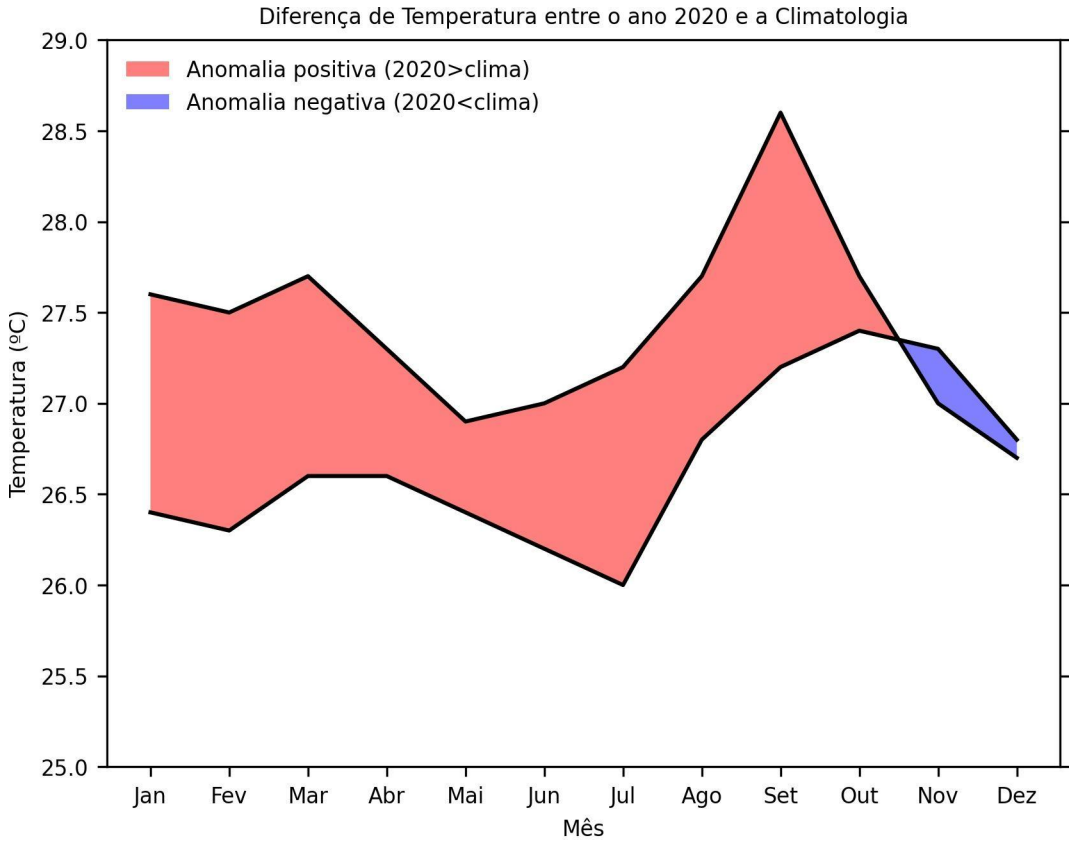
Exemplo 2: Script ex02.py.

O resultado será:



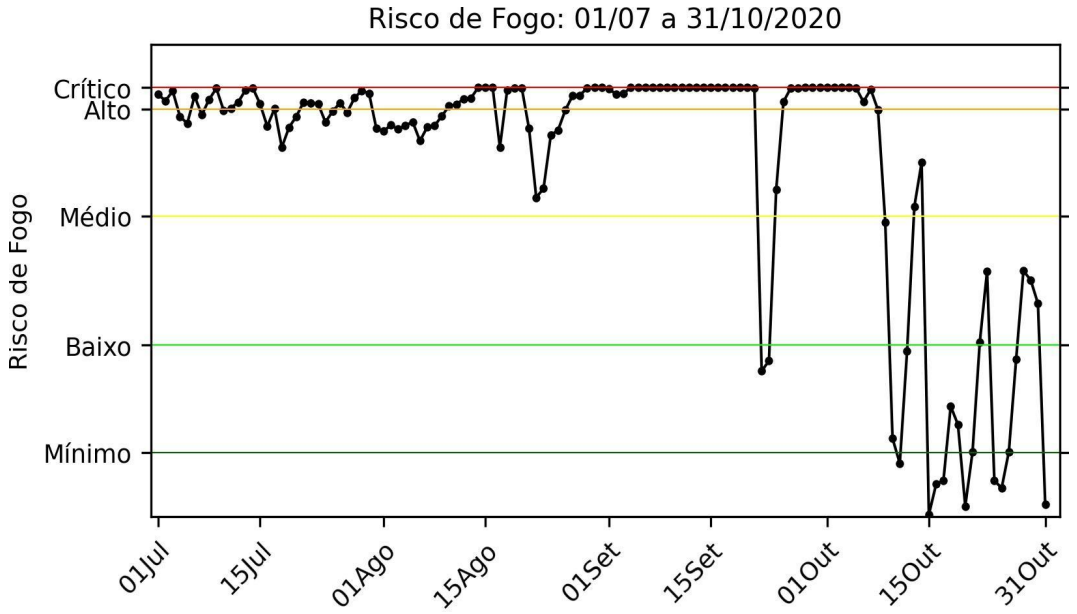
Exemplo 3: Script ex03.py.

O resultado será:



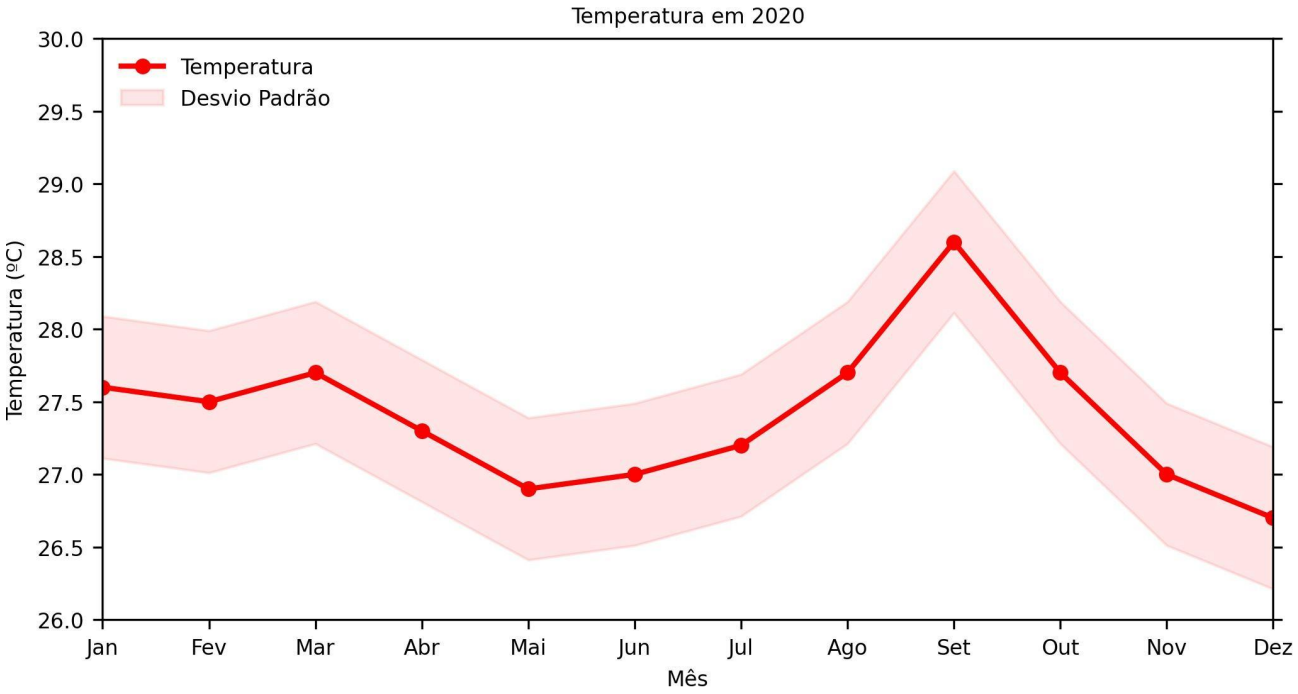
Exemplo 4: Script ex04.py.

O resultado será:



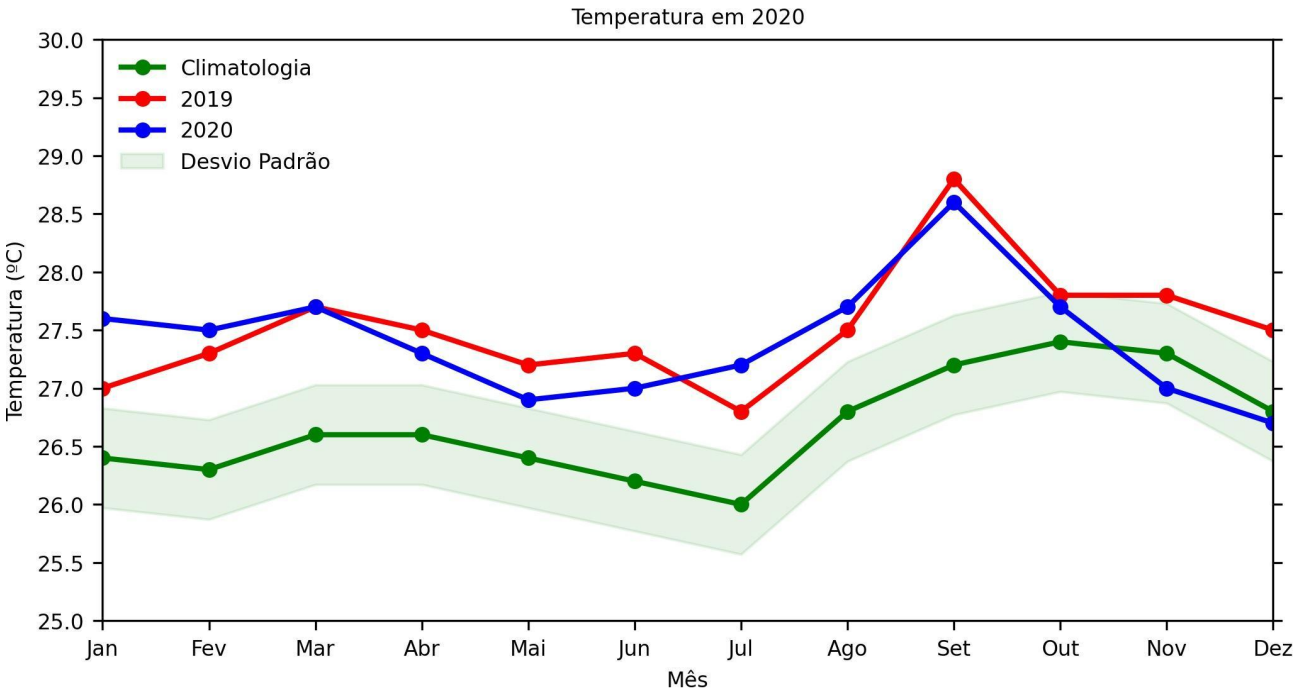
Exemplo 5: Script ex05.py.

O resultado será:



Exemplo 6: Script ex06.py.

O resultado será:



Exemplo 7: Script ex07.py.

O resultado será:

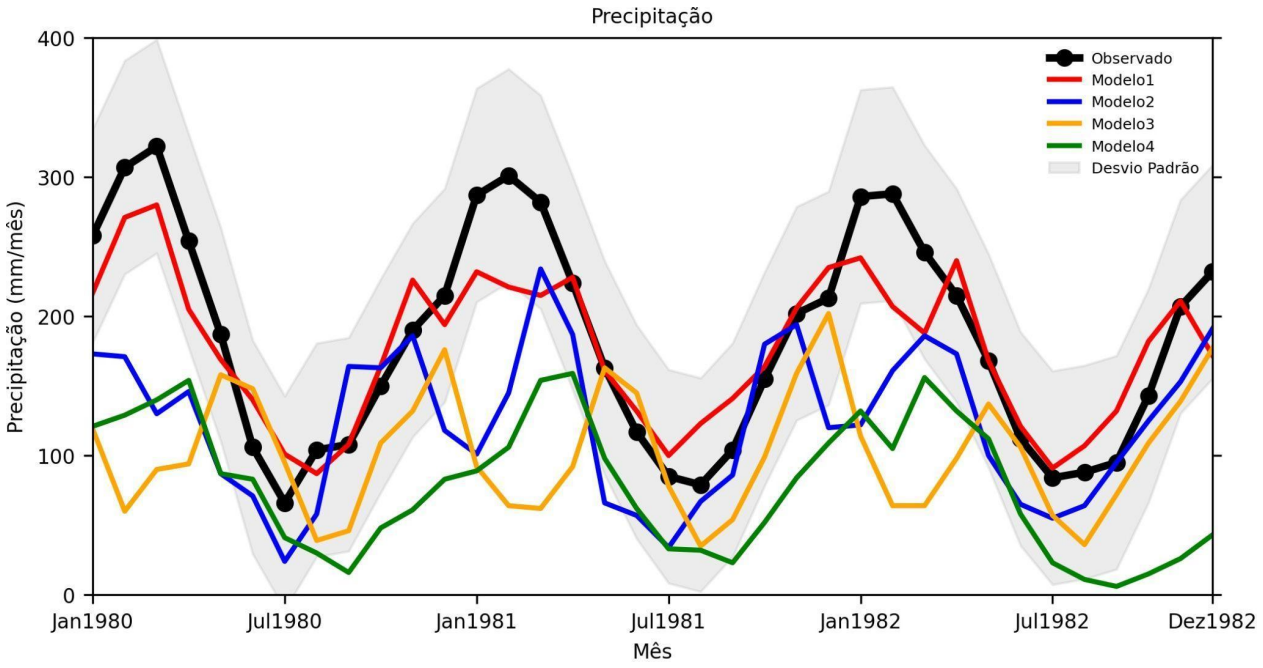


Gráfico de pizza

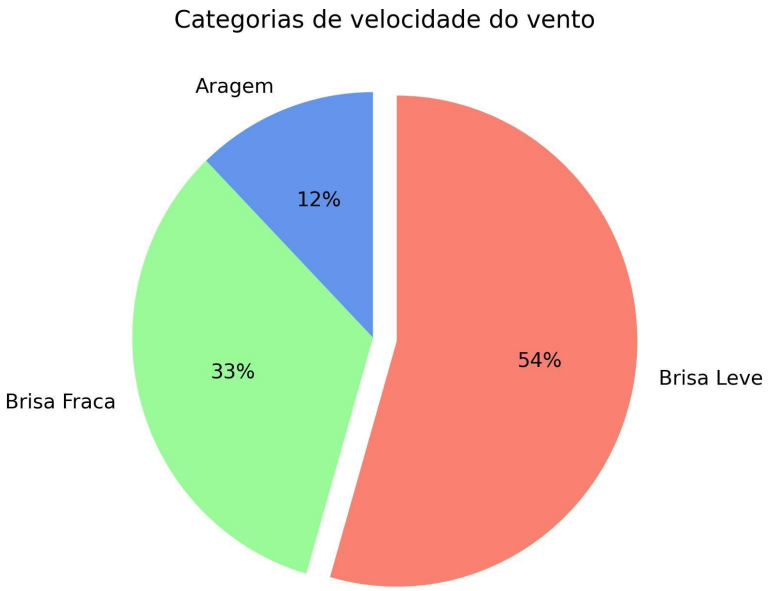
Leitura recomendada:
https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.pie.html
https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_features.html#sphx-glr-gallery-pie-and-polar-charts-pie-features-py
https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-gallery-pie-and-polar-charts-pie-and-donut-labels-py

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/grafico_pizza

Exemplo 1: Script ex01.py.

O resultado será:



Heatmap

Leitura recomendada:
https://seaborn.pydata.org/generated/seaborn.heatmap.html
https://matplotlib.org/stable/gallery/images_contours_and_fields/image_annotated_heatmap.html

Não esquecer de instalar a biblioteca seaborn:

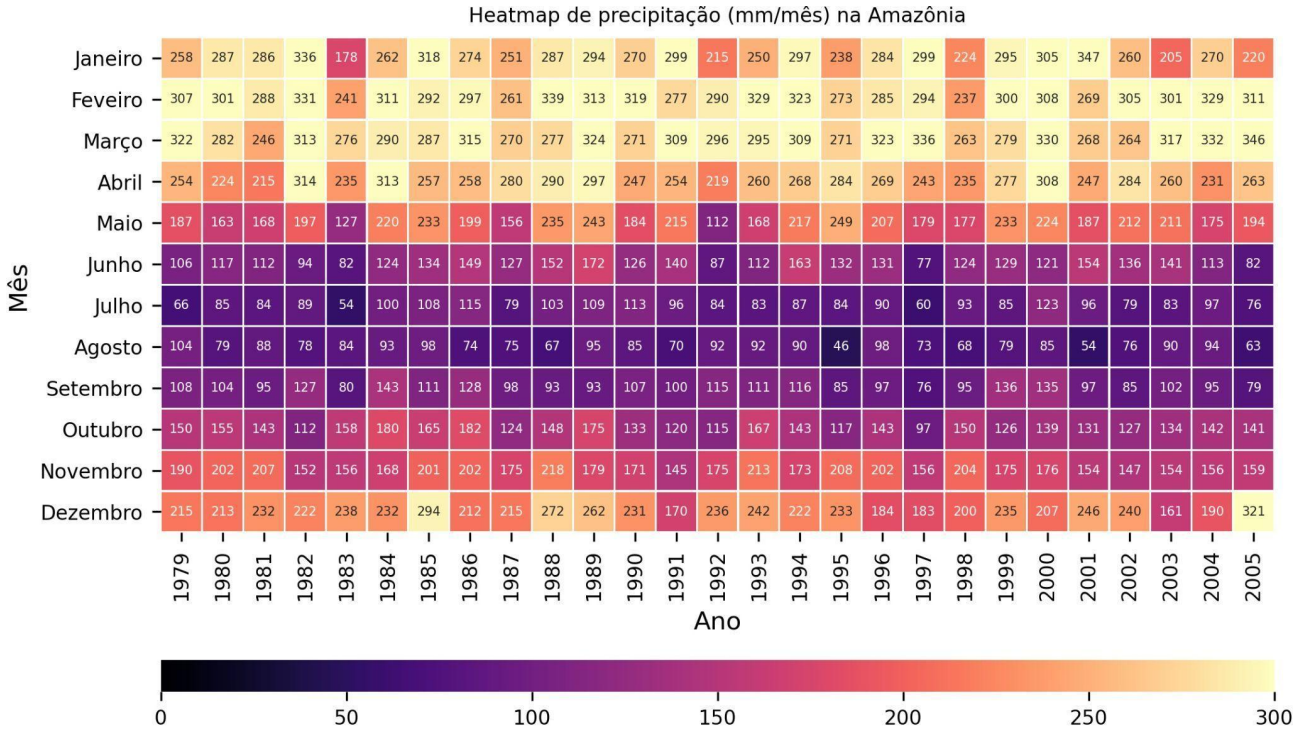
```
pip install seaborn
```

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/heatmap

Exemplo 1: Script ex01.py.

O resultado será:



Rosa dos ventos

Leitura recomenda:
https://windrose.readthedocs.io/en/latest/usage.html
https://windrose.readthedocs.io/en/latest/api.html#windrose.plot_windrose

Editar o arquivo windrose.py que se encontra no endereço abaixo para que a figura seja gerada conforme se visualiza.

`/home/<usuario>/<programa>/envs/<ambiente>/lib/python3.9/site-packages/windrose`

Onde: **<usuario>** é o nome do usuário da máquina, **<programa>** depende da sua instalação, pode ser miniconda3 ou anaconda3 e **<ambiente>** é o nome do ambiente que se está trabalhando.

Não esquecer de instalar a biblioteca abaixo:

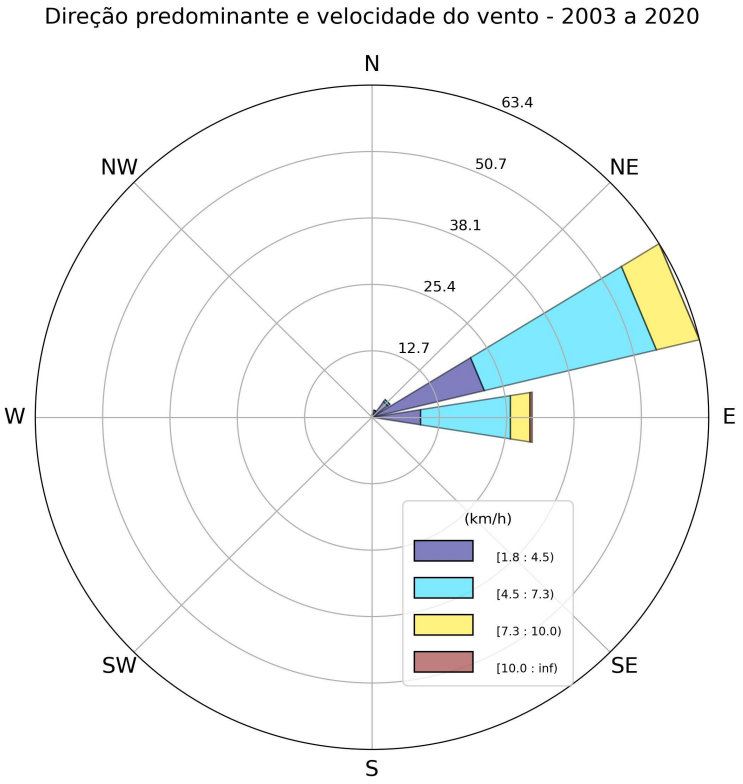
```
pip install windrose
```

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/rosa_ventos

Exemplo 1: Script ex01.py.

O resultado será:



Uso de shapefile para mascarar dados

A dica foi disponibilizada pelo Prof. Dr. Enrique Vieira do curso de Ciências Atmosféricas da UNIFEI/Itajubá.

No exemplo abaixo, o arquivo de precipitação (0.1º latitude x 0.1º de longitude) apresenta dados sobre o Brasil (figura abaixo). O objetivo consiste em utilizar o shapefile do Bioma Amazônia (contorno em vermelho) para mostrar a precipitação somente sobre este bioma.

As dimensões longitude (linha 39, ds.longitude) e latitude (linha 40, ds.latitude) são do arquivo NetCDF MERGE_CPTEC_20220131.nc que tem como nome de variável a prec (linha 36, ds.prec). Verifique antes se o nome das dimensões do arquivo são essas.

Não esquecer de instalar as bibliotecas abaixo:

```
pip install rasterio
pip install pyproj
pip install geopandas
pip install salem
```

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/blob/main/mascara_dados

Exemplo 1: Script ex01.py.

A figura abaixo mostra o dado real de precipitação sobre o Brasil.

O resultado será:

Como criar painéis

Leitura recomendada:
https://matplotlib.org/stable/api/axes_api.html#subplots
https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot2grid.html
https://towardsdatascience.com/what-are-the-plt-and-ax-in-matplotlib-exactly-d2cf4bf164a9
https://stackoverflow.com/questions/34162443/why-do-many-examples-use-fig-ax-plt-subplots-in-matplotlib-pyplot-python
https://matplotlib.org/stable/gallery/lines_bars_and_markers/scatter_hist.html#sphx-glr-gallery-lines-bars-and-markers-scatter-hist-py
https://matplotlib.org/stable/gallery/lines_bars_and_markers/psd_demo.html#sphx-glr-gallery-lines-bars-and-markers-psd-demo-py
https://matplotlib.org/stable/gallery/subplots_axes_and_figures/align_labels_demo.html#sphx-glr-gallery-subplots-axes-and-figures-align-labels-demo-py
https://stackoverflow.com/questions/12372380/matplotlib-repositioning-a-subplot-in-a-grid-of-subplots

Para a criação de painéis se utiliza o método subplots do matplotlib.

O plt.subplots() é uma função que retorna uma tupla que contém uma figura e os eixos dos objetos. Ao utilizar:

```
fig, ax = plt.subplots()
```

Descompacta-se a tupla em duas variáveis, isto é, `fig` e `ax`. A variável fig é útil para alterar os atributos da figura ou para salvá-la em um arquivo no computador. Além disso, todos os objetos de eixo (os objetos que possuem métodos de plotagem) são ajustados de forma mais fácil utilizando a forma acima que é mais conciso que:

```
fig = plt.figure()
ax = fig.add_subplot(111)
```

Para melhor entendimento, a figura abaixo foi feita por meio do matplotlib (plt), e alguns pontos precisam ser mencionados:

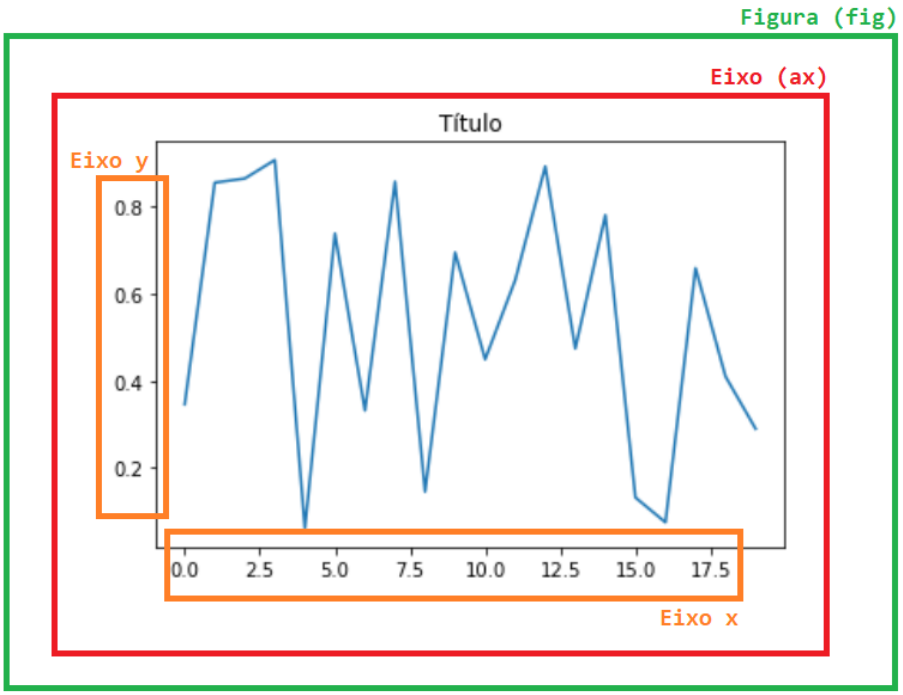
1. Ao gerar um gráfico com o alias plt, cria-se o objeto Figura (fig) destacado em verde.
2. Um objeto eixo (ax) destacado em vermelho é criado implicitamente com o gráfico de linha.
3. E por fim, todos os elementos do gráfico de linha, isto é, os eixos x e y (destacados em laranja) são renderizados dentro do objeto eixo (ax).

Em outras palavras:

O objeto Figura (fig) é o local para desenhar/inserir qualquer elemento. A partir disso, desenha-se um gráfico em uma célula que neste contexto é representado pelo objeto eixo (ax). No caso da geração de um único gráfico, não há necessidade de geração desta célula, simplesmente, o plot pode ser feito diretamente por meio da chamada plt.plot(...).

1. Código Python que gerou a figura abaixo:

```
import matplotlib.pyplot as plt
import numpy as np

plt.plot(np.random.rand(20))
plt.title('Título')
plt.show()
```



2. O código abaixo gera o mesmo gráfico, porém utilizando os objetos fig e ax.

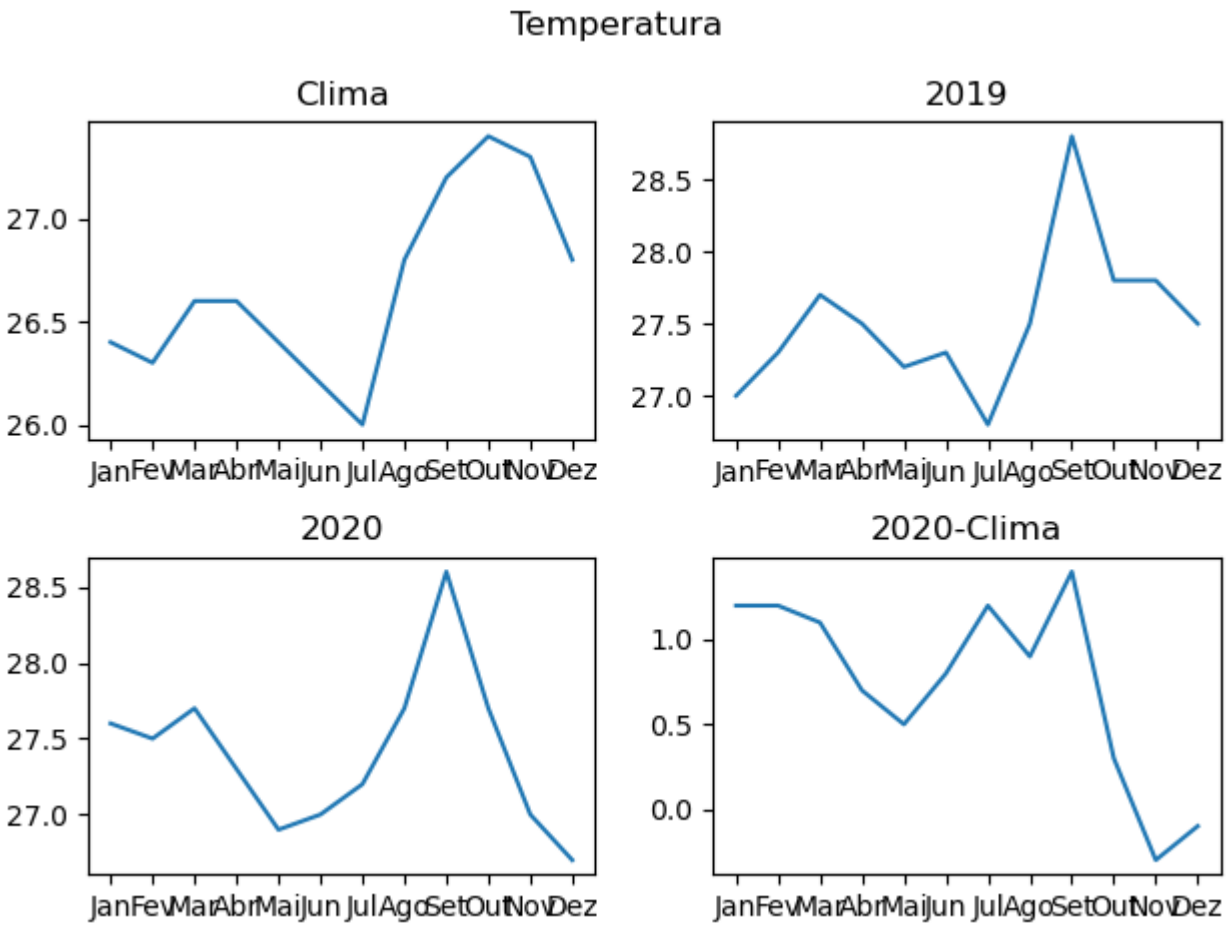
```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()
ax.plot(np.random.rand(20))
ax.set_title('Título')
plt.show()
```

Quando há necessidade de plotar apenas um gráfico, não é necessário “desenhar” esta célula. Este método deve ser utilizado quando se deseja desenhar vários gráficos em um único gráfico (painel de figuras).

3. Exemplo de um painel com 4 figuras (2 linhas x 2 colunas):

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('temperatura.txt',
                 sep='\t',
                 names=['Mês', 'Climatologia', '2019', '2020'])

mes = df['Mês']
pclima = df['Climatologia']
p2019 = df['2019']
p2020 = df['2020']

nlin = 2
ncol = 2

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nlin, ncol)

ax1.plot(mes, pclima)
ax2.plot(mes, p2019)
ax3.plot(mes, p2020)
ax4.plot(mes, p2020-pclima)

ax1.title.set_text('Clima')
ax2.title.set_text('2019')
ax3.title.set_text('2020')
ax4.title.set_text('2020-Clima')

fig.suptitle('Temperatura')

fig.tight_layout()

plt.show()
```

O resultado será:



Nota-se que os gráficos são desenhados utilizando a posição linha/coluna por meio da instância ax. Por outro lado, o título da figura (Título principal) utiliza a instância fig porque ele não faz parte de nenhuma das 4 células criadas.

O método:

```
plt.tight_layout()
```

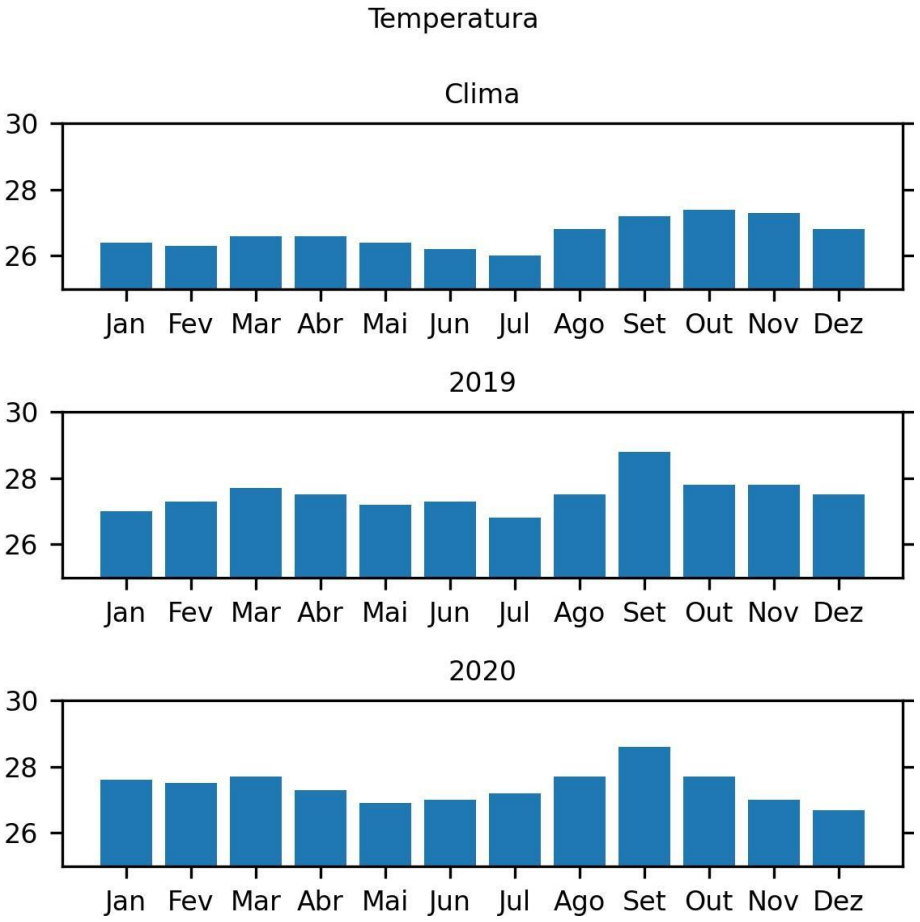
Ajusta automaticamente os espaços entre os gráficos para que todos tenham o mesmo espaçamento horizontal e vertical.

Todos os scripts podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/painel

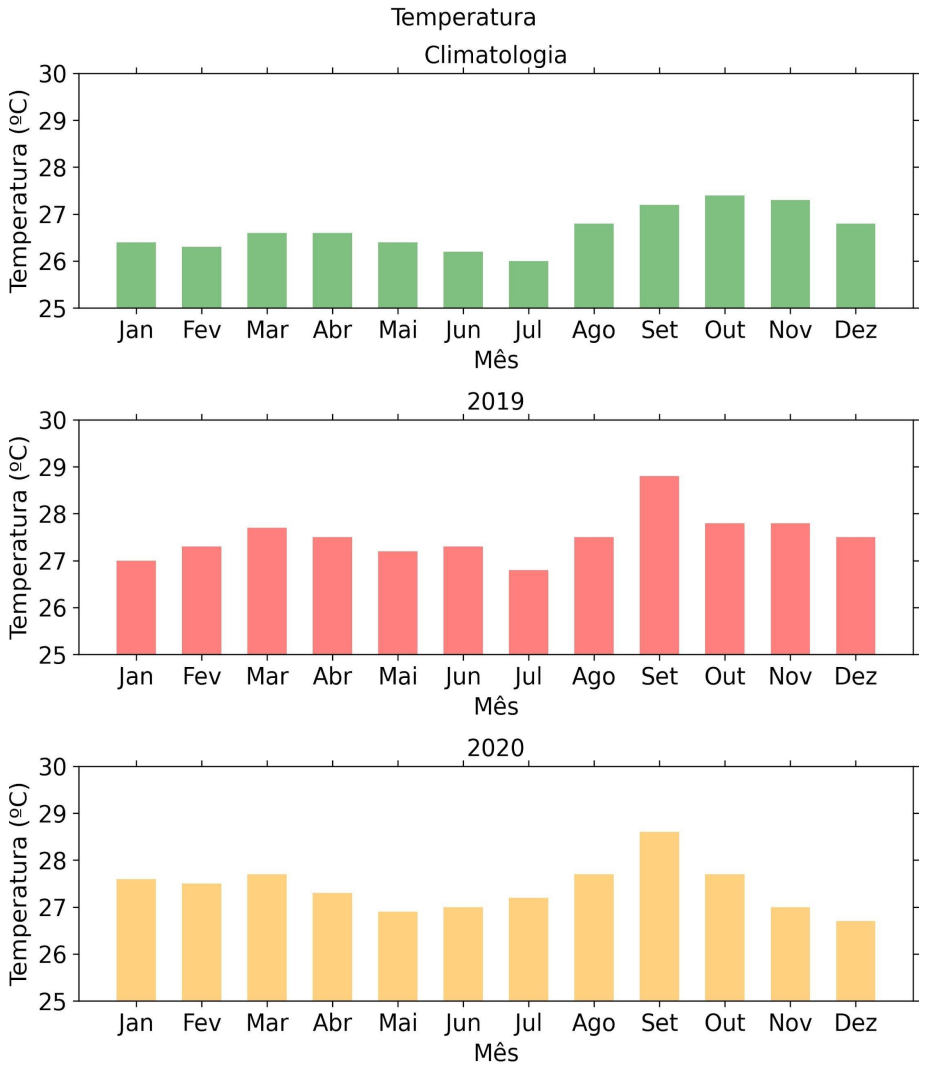
Exemplo 1: Script ex01.py.

O resultado será:



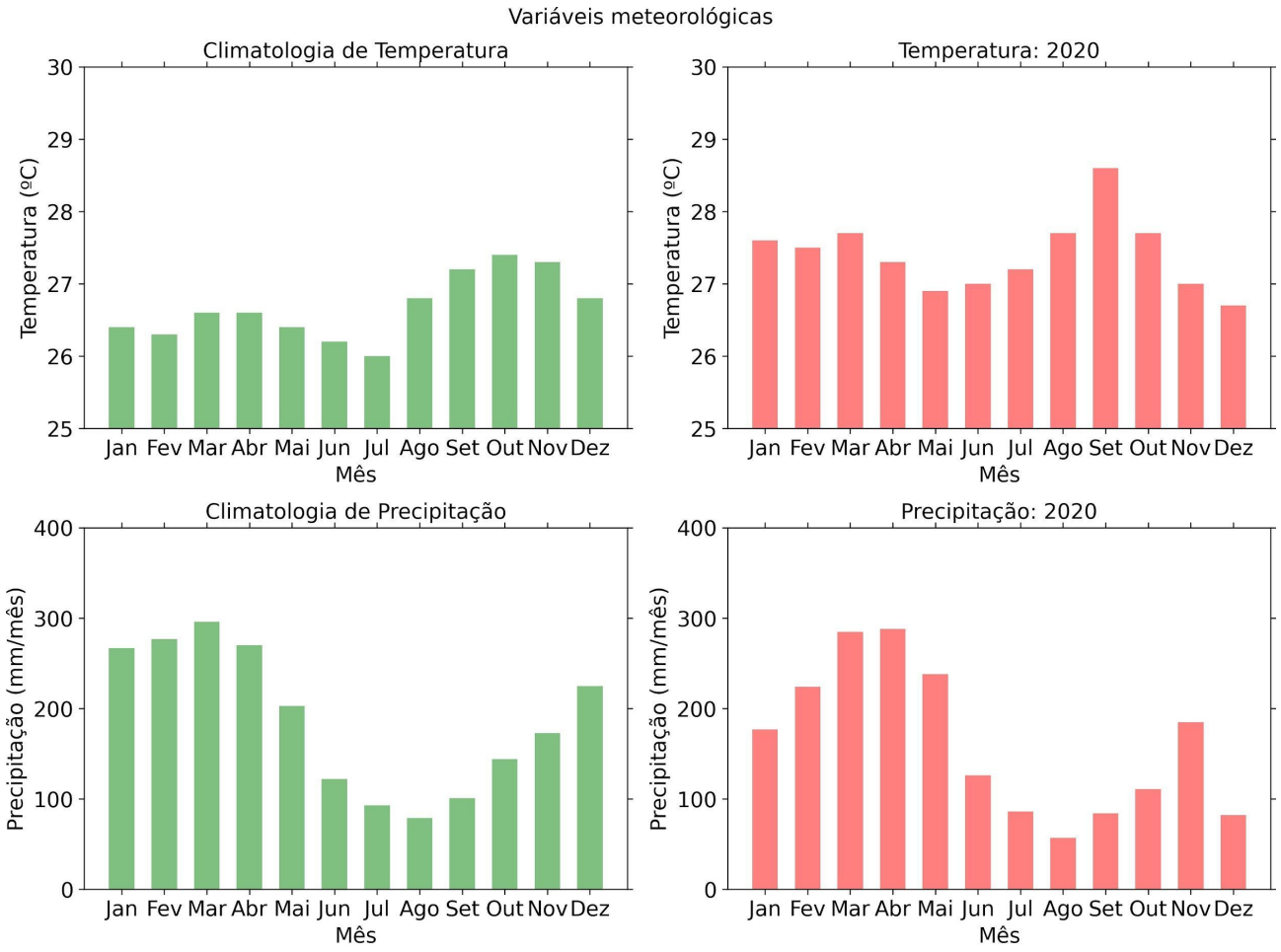
Exemplo 2: Script ex02.py.

O resultado será:



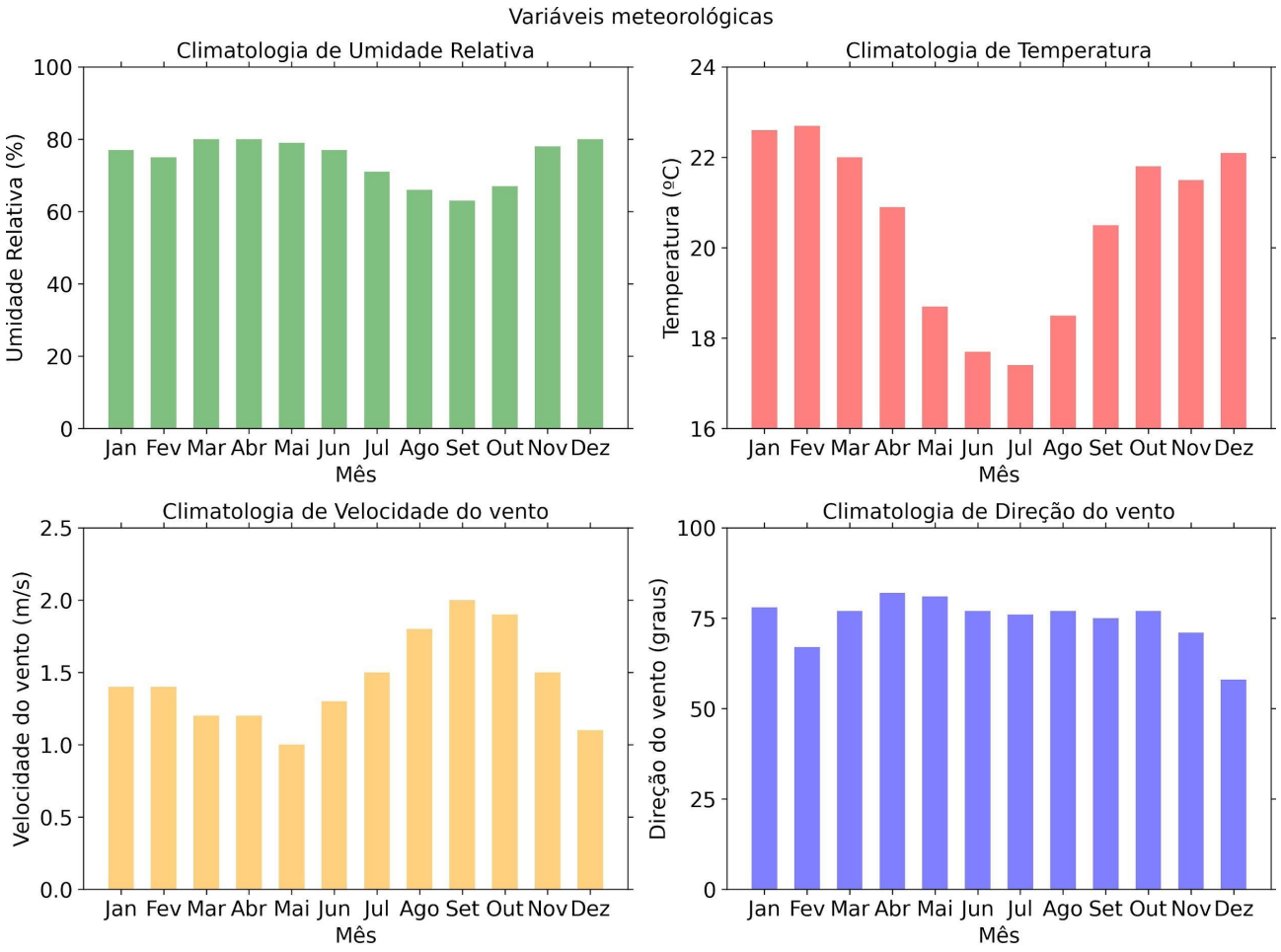
Exemplo 3: Script ex03.py.

O resultado será:



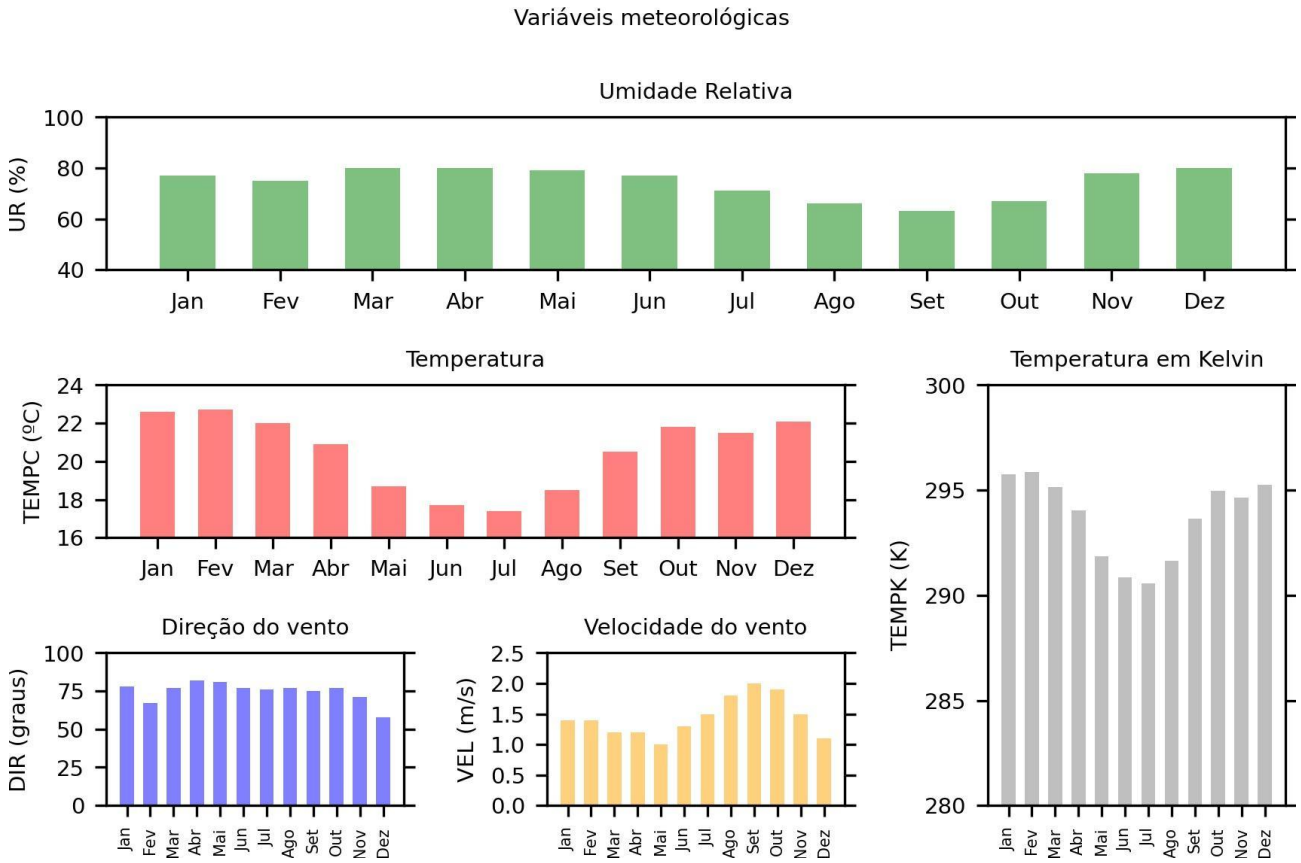
Exemplo 4: Script ex04.py.

O resultado será:



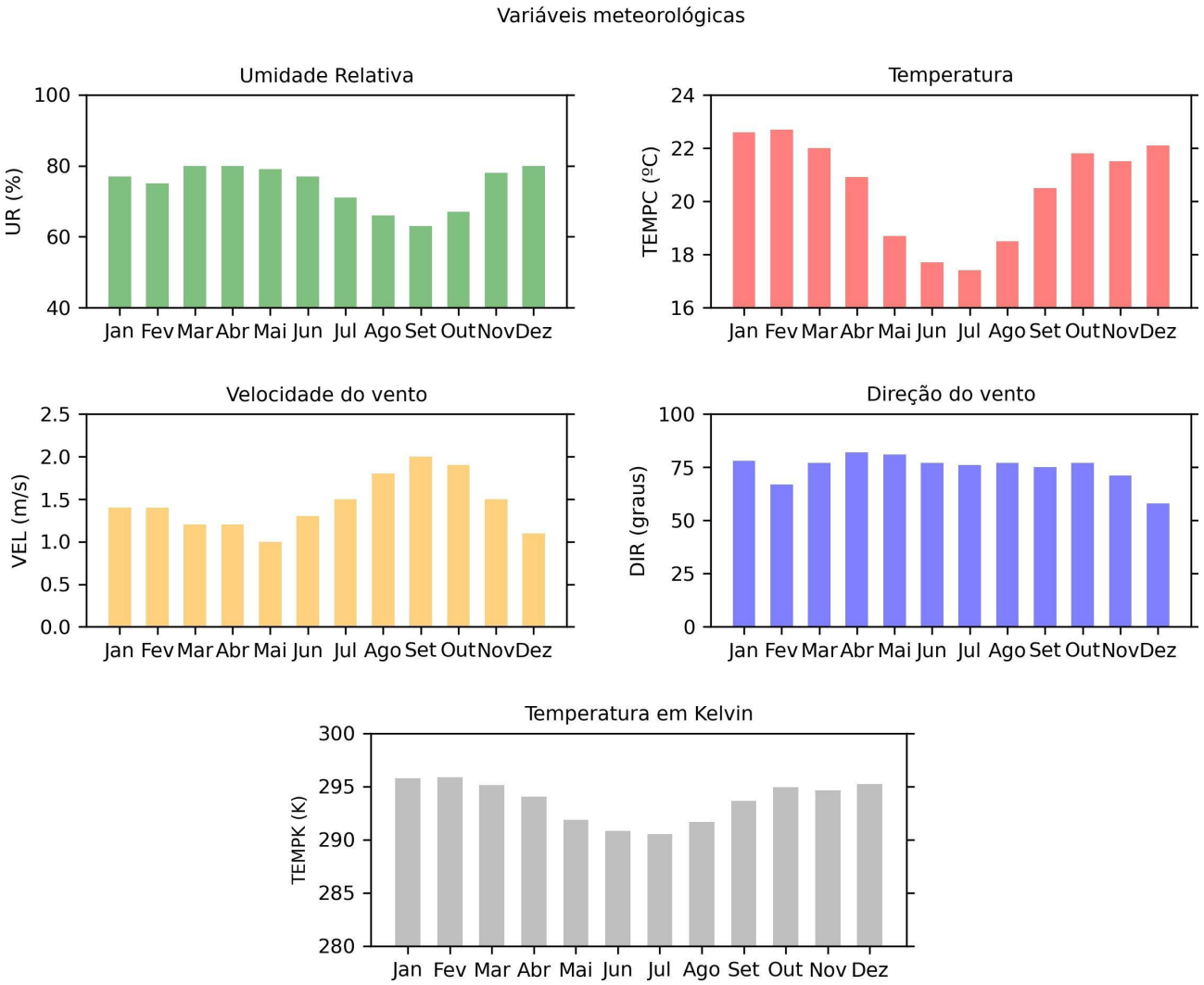
Exemplo 5: Script ex05.py.

O resultado será:



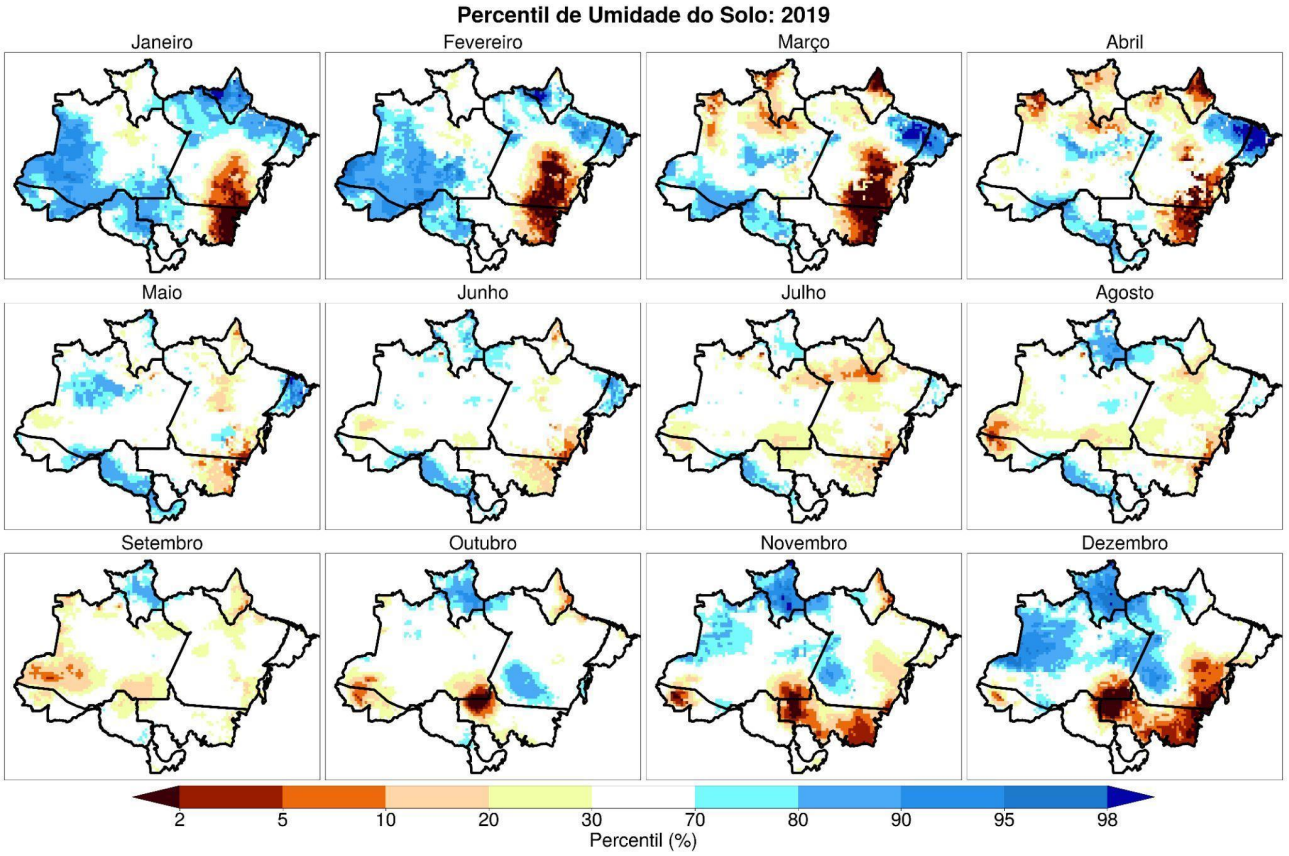
Exemplo 6: Script ex06.py.

O resultado será:



Exemplo 7: Script ex07.py.

O resultado será:



Um pouco de estatística

Leitura recomendada:
https://docs.scipy.org/doc/scipy/reference/stats.html

Função stats.describe

Calcula várias estatísticas descritivas da matriz utilizada.

Leitura recomendada:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.describe.html#scipy.stats.describe

Neste exemplo, utiliza-se apenas a série temporal de precipitação para gerar uma estatística básica sobre a série temporal.

```
import pandas as pd
from scipy import stats

# Abertura do arquivo utilizando o separador TAB e adiciona o
# título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

x = df['PREC']  # Importa a precipitação.

y = stats.describe(x)  # Calcula a estatística básica.

print(y)
```

Neste exemplo, utilizam-se todas as variáveis do arquivo para gerar a estatística.

```
import pandas as pd
from scipy import stats

# Abertura do arquivo utilizando o separador TAB e adiciona o
# título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

ur = df['UR']
temp = df['Temp']
prec = df['PREC']
vel = df['VelVento']
dir = df['DirVento']

lista_variaveis = [ur, temp, prec, vel, dir]

[print("Resultado: ", stats.describe(variavel)) for variavel in lista_variaveis]
```

Função stats.mode

Retorna uma matriz do valor modal (mais comum) na matriz passada.

Leitura recomendada:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mode.html#scipy.stats.mode

```
import pandas as pd
from scipy import stats

# Abertura do arquivo utilizando o separador TAB e adiciona o
# título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

dir = df['DirVento']

y = stats.mode(dir)

print(y)
```

Função scipy.stats.tmean

Calcula a média aritmética dos valores fornecidos ignorando os valores fora dos limites fornecidos.

Leitura recomendada:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.tmean.html#scipy.stats.tmean

O exemplo abaixo exclui os valores do limite inferior e superior e calcula a média.

```
import pandas as pd
from scipy import stats

# Abertura do arquivo utilizando o separador TAB e adiciona
# o título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

ur = df['UR']

x = stats.tmean(ur)

print(x)  # 74.32870370370371

# Exclui os valores abaixo de 60% e acima de 80%.
y = stats.tmean(ur, limits=(60, 80))

print(y)  # 72.8407643312102
```

Por outro lado, o exemplo abaixo exclui os valores iguais e inferiores e iguais e superiores do limite fornecido e calcula a média.

```
import pandas as pd
from scipy import stats

# Abertura do arquivo utilizando o separador TAB e adiciona
# o título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

ur = df['UR']

x = stats.tmean(ur)

print(x)  # 74.32870370370371

# Exclui os valores iguais e abaixo de 60% e iguais e acima de 80%.
y = stats.tmean(ur, limits=(60, 80), inclusive=(False, False))

print(y)  # 72.30075187969925
```

Função scipy.stats.tmin

Encontra o valor mínimo de uma matriz ao longo do eixo especificado, mas considerando apenas os valores maiores que um limite inferior especificado.

Leitura recomendada:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.tmin.html#scipy.stats.tmin

```
import pandas as pd
from scipy import stats

# Abertura do arquivo utilizando o separador TAB e adiciona o
# título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

ur = df['UR']

x = stats.tmin(ur)

print(x)  # 56

# Retorna o valor mínimo dado um limite inferior.
y = stats.tmin(ur, lowerlimit=70)

print(y)  # 70

# Retorna o valor mínimo dado um limite inferior sendo que ele é excluído.
z = stats.tmin(ur, lowerlimit=70, inclusive=False)

print(z)  # 71
```

O mesmo pode ser feito para obter o valor máximo:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.tmax.html#scipy.stats.tmax

E o desvio padrão:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.tstd.html#scipy.stats.tstd

Função scipy.stats.pearsonr

Calcula o coeficiente de correlação de Pearson e o p-value.

Leitura recomendada:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html#scipy.stats.pearsonr

```
import pandas as pd
from scipy import stats

# Abertura do arquivo utilizando o separador TAB e adiciona o
# título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

x = df['Temp']
y = df['PREC']

r = stats.pearsonr(x, y)

          #          r                  p-value
print(r)  # (0.5522538017058155, 1.2107247393256543e-18)
```

Função scipy.stats.spearmanr

Calcula o coeficiente de correlação de Spearman e o p-value.

Leitura recomendada:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html#scipy.stats.spearmanr

```
import pandas as pd
from scipy import stats

# Abertura do arquivo utilizando o separador TAB e adiciona
# o título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

x = df['Temp']
y = df['PREC']

r = stats.spearmanr(x, y)

print(r)

# SpearmanrResult(correlation=0.677086961910298, pvalue=2.5283781966531542e-30)
```

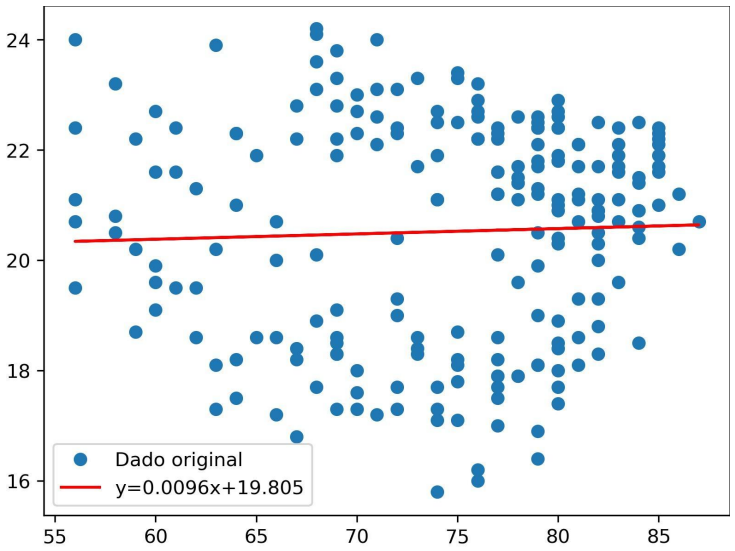
Função scipy.stats.linregress

Calcula a regressão linear de mínimos quadrados para dois conjuntos de medições.

Leitura recomendada:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html#scipy.stats.linregress

```
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt

# Abertura do arquivo utilizando o separador TAB e adiciona
# o título como primeira linha.
df = pd.read_csv('variaveis_meteorologicas.txt',
                 sep='\t',
                 names=['Data','UR','Temp','PREC','VelVento','DirVento'])

x = df['UR']
y = df['Temp']

res = stats.linregress(x, y)  # Calcula a regressão.

print(res)

# LinregressResult(slope=0.009648198915792867, intercept=19.804621140772895,
# rvalue=0.036961733120948005, pvalue=0.5890198667365112,
# stderr=0.017831594203681106, intercept_stderr=1.3324861298480515)

r2 = res.rvalue**2  # Calcula o R**2.

# Gera o gráfico.
plt.plot(x, y, 'o', label='Dado original')
plt.plot(x,
         res.intercept + res.slope*x,
         'r',
         label=f'y={res.slope:.4f}x+{res.intercept:.3f}')
plt.legend()
# Salva a figura no formato ".jpg" com dpi=300 e remove espaços excedentes.
plt.savefig('ex01.jpg', transparent=True, dpi=300, bbox_inches='tight',
            pad_inches=0)
```

O resultado será:



Anomalia

Neste exemplo, é calculada a anomalia de Temperatura da Superfície do Mar na região do Niño 3.4. Agradeço ao Willy Hagi por ceder os códigos para realizar este cálculo.

Instalar a biblioteca:

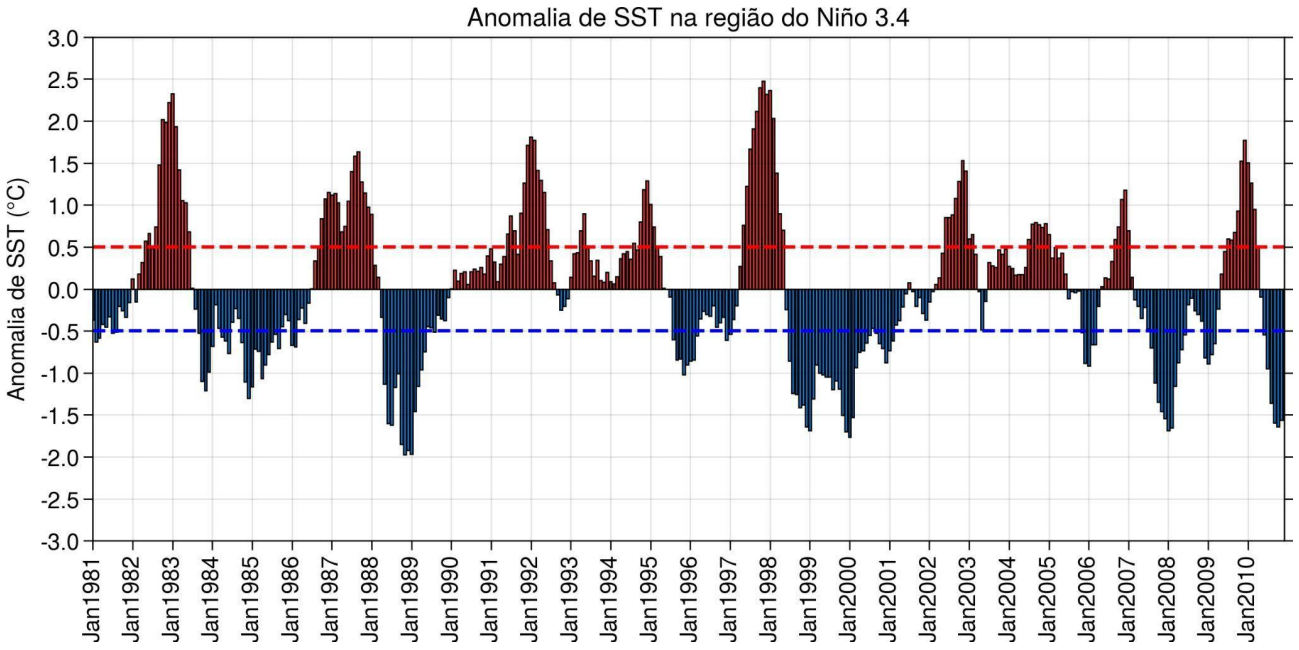
conda install netcdf4

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/estatisticos/anomalia

Exemplo 1: Script ex01.py.

O resultado será:



Correlação

Neste exemplo calcula-se a correlação entre a anomalia espacial de TSM e a anomalia da série temporal de TSM na região do Niño 3.4. Agradeço ao Willy Hagi por ceder o código para realizar este cálculo.

Instalar a biblioteca:

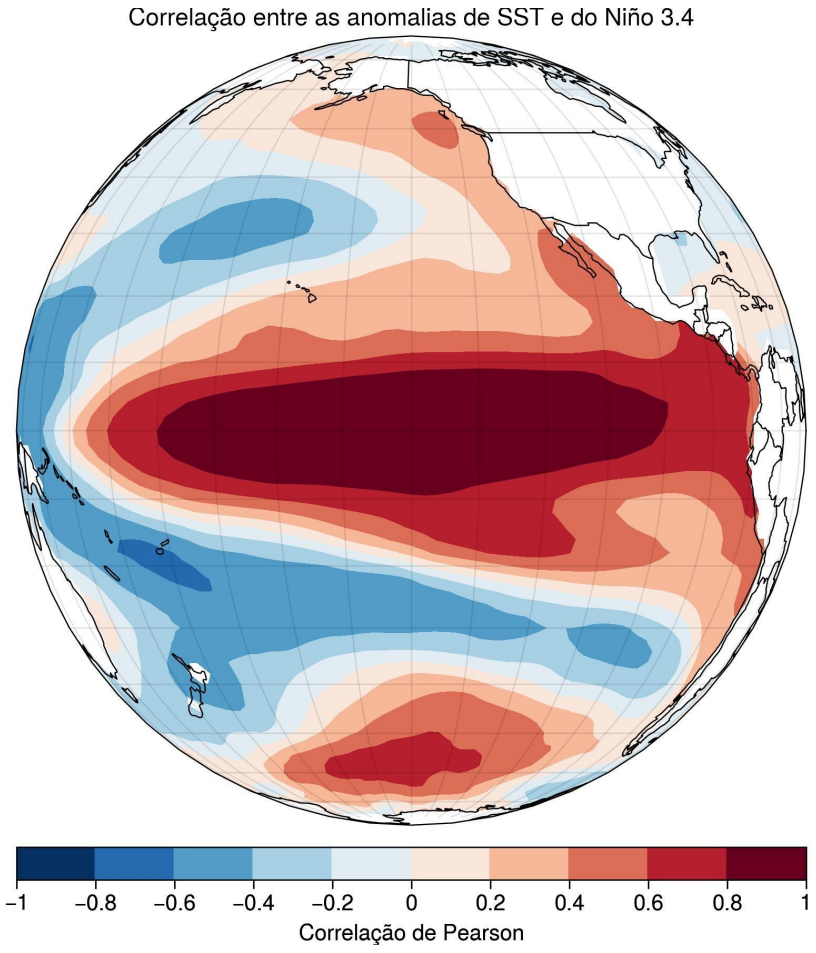
pip install esmtools

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/graficos/estatisticos/correlacao

Exemplo 1: Script ex01.py.

O resultado será:



Índices Climáticos com o climate_indices

A partir da biblioteca climate_indices é possível calcular os seguintes índices:

1. **pdsi** (Palmer Drought Severity Index - PDSI)
2. **percentage_of_normal** (Esta função encontra a percentagem dos valores normais (média de cada calendário mês ou dia ao longo de um período de calibração especificado de anos) para um determinado escala de passos de tempo)
3. **pet** (potential evapotranspiration (PET) using Thornthwaite's equation)
4. **scpdsi** (self-calibrated Palmer Drought Severity Index - SPEI)
5. **spei** (Standardised Precipitation-Evapotranspiration Index - SPEI)
6. **spi** (Standardized Precipitation Index - SPI)

Instalação da biblioteca climate_indices

```
conda install -n <Nome_Ambiente_Virtual> -c nesii -c conda-forge esmpy

pip install climate-indices
```

Onde: `<Nome_Ambiente_Virtual>` é o nome do ambiente virtual em uso.

Cálculo do SPI

Para o cálculo do SPI a unidade da variável precipitação tem que ser somente mm e não mm/day. Para resolver isso, utiliza-se o `nco` por meio do comando abaixo.

```
ncatted -a units,precip,o,c,"mm" input.nc output.nc
```

Onde:

`units`: é o atributo unidade da variável `precip` do arquivo `input.nc` que se deseja alterar, ou seja, se altera esse atributo que está em `mm/day` para somente `mm`. Sem isso, será retornado erro de execução. Essa unidade (`mm/day`) foi verificada por meio do comando `ncdump -h input.nc`.

Leitura recomendada:
https://icclim.readthedocs.io/en/latest
https://pypi.org/project/climate-indices
https://github.com/monocongo/climate_indices
https://climate-indices.readthedocs.io/en/latest
https://github.com/monocongo/climate_indices/issues/326
https://readthedocs.org/projects/climate-indices/downloads/pdf/latest

Sobre escrita de dado no formato NetCDF:
http://xarray.pydata.org/en/stable/io.html
http://xarray.pydata.org/en/stable/generated/xarray.Dataset.to_netcdf.html
http://xarray.pydata.org/en/stable/generated/xarray.Variable.html

```
from climate_indices import indices


# Imprime a ajuda de todos os índices.
print(help(indices))
```

Exemplo de cálculo do SPI

Neste exemplo, calcula-se o SPI de comprimento 3 meses e gera o NetCDF espacial como também a série temporal (média de todos os valores de latitude e longitude).

O script pode ser acessado pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/indices_climaticos

Exemplo 1: Script ex01.py.

Serão gerados dois arquivos no formato NetCDF: 1) spi_espacial.nc e 2) spi_serie.nc.

Índices Climáticos com o xclim

O xclim é uma biblioteca para cálculo de índices climáticos baseado em *Xarray*. Essa biblioteca fornece atualmente mais de 50 índices relacionados à temperatura média, mínima e máxima diária, precipitação diária, fluxo de água e concentração de gelo marinho. Mais informações podem ser obtidas no link abaixo:

https://xclim.readthedocs.io/en/stable/index.html

Descrição dos índices:
https://xclim.readthedocs.io/en/stable/indicators.html
https://xclim.readthedocs.io/en/stable/notebooks/cli.html

Leitura recomendada:
https://xclim.readthedocs.io/en/stable/notebooks/example.html#Threshold-indices
https://xclim.readthedocs.io/en/stable/indicators_api.html

Instalação da biblioteca xclim

Via pip: `pip install xclim`

Via conda: `conda install -c conda-forge xclim`

Para testar a instalação, execute o Python e digite as linhas abaixo:

```
import xclim



dir(xclim)
help(xclim)
```

Serão mostrados os módulos (atmos, land e sealce) e o help, como utilizar os índices.

Para visualizar os demais módulos, as possibilidades são:

- atmos
- land
- sealce

Ao calcular o índice, existe uma opção para a variável freq e os valores assumidos podem ser:

- `YS`: annual starting in January
- `YS-JUL`: annual starting in July
- `MS`: monthly
- `QS-DEC`: seasonal starting in December

O valor padrão é: `freq = 'YS'`.

Mais opções podem ser encontradas em:

https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#timeseries-offset-aliases

Módulo atmos

Leitura recomendada:
https://xclim.readthedocs.io/en/stable/indicators.html#atmos-atmosphere

Obter ajuda sobre um índice.

`help(xclim.atmos.<Nome_Indice>)`

Exemplo: `help(xclim.atmos.cold_spell_days)`

Como usar o índice?

`x = xclim.atmos.<indice_interesse>(parametros)`

Exemplo: `x = xclim.atmos.cold_spell_days(parametros)`

Opções de índices (`<Nome_Indice>`) do módulo atmos:

1. biologically_effective_degree_days
2. calm_days
3. cold_and_dry_days
4. cold_and_wet_days
5. cold_spell_days
6. cold_spell_duration_index
7. cold_spell_frequency
8. consecutive_frost_days
9. cool_night_index
10. cooling_degree_days
11. corn_heat_units
12. daily_freezethaw_cycles
13. daily_pr_intensity
14. daily_temperature_range
15. daily_temperature_range_variability
16. days_over_precip_doy_thresh
17. days_over_precip_thresh
18. days_with_snow
19. degree_days_exceedance_date
20. drought_code
21. dry_days
22. dry_spell_frequency
23. dry_spell_total_length
24. extreme_temperature_range
25. fire_season
26. fire_weather_indexes
27. first_day_above
28. first_day_below
29. first_snowfall
30. fraction_over_precip_doy_thresh
31. fraction_over_precip_thresh
32. freezethaw_spell_frequency
33. freezethaw_spell_max_length
34. freezethaw_spell_mean_length
35. freezing_degree_days

36. freshet_start
37. frost_days
38. frost_free_season_end
39. frost_free_season_length
40. frost_free_season_start
41. frost_season_length
42. growing_degree_days
43. growing_season_end
44. growing_season_length
45. growing_season_start
46. heat_index
47. heat_wave_frequency
48. heat_wave_index
49. heat_wave_max_length
50. heat_wave_total_length
51. heating_degree_days
52. high_precip_low_temp
53. hot_spell_frequency
54. hot_spell_max_length
55. huglin_index
56. humidex
57. ice_days
58. jetstream_metric_woollings
59. last_snowfall
60. last_spring_frost
61. latitude_temperature_index
62. liquid_precip_accumulation
63. liquid_precip_ratio
64. max_1day_precipitation_amount
65. max_daily_temperature_range
66. max_n_day_precipitation_amount
67. max_pr_intensity
68. maximum_consecutive_dry_days
69. maximum_consecutive_frost_free_days
70. maximum_consecutive_warm_days
71. maximum_consecutive_wet_days
72. potential_evapotranspiration
73. precip_accumulation
74. rain_approximation
75. rain_on_frozen_ground_days
76. relative_humidity
77. relative_humidity_from_dewpoint
78. rprctot
79. saturation_vapor_pressure
80. snowfall_approximation
81. solid_precip_accumulation
82. specific_humidity
83. tg
84. tg10p
85. tg90p
86. tg_days_above
87. tg_days_below
88. tg_max

89. tg_mean
90. tg_min
91. thawing_degree_days
92. tn10p
93. tn90p
94. tn_days_above
95. tn_days_below
96. tn_max
97. tn_mean
98. tn_min
99. tropical_nights
100. tx10p
101. tx90p
102. tx_days_above
103. tx_days_below
104. tx_max
105. tx_mean
106. tx_min
107. tx_tn_days_above
108. warm_and_dry_days
109. warm_and_wet_days
110. warm_spell_duration_index
111. water_budget
112. wet_precip_accumulation
113. wetdays
114. wetdays_prop
115. wind_chill_index
116. wind_speed_from_vector
117. wind_vector_from_speed
118. windy_days

Módulo land

Leitura recomendada:
https://xclim.readthedocs.io/en/stable/indicators.html#land-land-surface

Obter ajuda sobre um índice.

```
help(xclim.land.<Nome_Indice>)
```

Exemplo: `help(xclim.land.base_flow_index)`

Como usar o índice?

```
x = xclim.land.<Nome_Indice>(parametros)
```

Exemplo: `x = xclim.land.base_flow_index(parametros)`

Opções de índices (`<Nome_Indice>`) do módulo land:

1. base_flow_index
2. blowing_snow
3. continuous_snow_cover_end
4. continuous_snow_cover_start

5. doy_qmax
6. doy_qmin
7. fit
8. freq_analysis
9. rb_flashiness_index
10. snd_max_doy
11. snow_cover_duration
12. snow_depth
13. snow_melt_we_max
14. snw_max
15. snw_max_doy
16. stats
17. winter_storm

Módulo sealce

Leitura recomendada:
https://xclim.readthedocs.io/en/stable/indicators.html#seaice-sea-ice

Obter ajuda sobre um índice.

```
help(xclim.seaIce.<Nome_Indice>)
```

Exemplo: `help(xclim.seaIce.sea_ice_area)`

Como usar o índice?

```
x = xclim.seasIceland.<Nome_Indice>(parametros)
```

Exemplo: `x = xclim.seaIce.sea_ice_area(parametros)`

Opções de índices (`<Nome_Indice>`) do módulo sealce:

1. sea_ice_area
2. sea_ice_extent

Algumas aplicações dos índices

Os arquivos **GPCP.serie.nc** e **tmed.serie.nc** encontram-se no link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/dados/netcdf

Exemplo 1: Uso do índice maximum_consecutive_dry_days

Leitura recomendada:
https://xclim.readthedocs.io/en/stable/indicators_api.html#xclim.indicators.atmos.maximum_consecutive_dry_days
https://xclim.readthedocs.io/en/stable/notebooks/example.html#Threshold-indices

Este índice calcula o **maior número de dias consecutivos em que a precipitação é menor que um dado limiar**. A unidade do arquivo deve estar em `mm/day`, caso contrário será retornado erro abaixo. Neste exemplo, utiliza-se o limiar de 3 mm/day.

```
DimensionalityError: Cannot convert from 'millimeter / day' ([length] /
[time]) to 'kilogram / meter ** 2' ([mass] / [length] ** 2)
```

```
import xclim
import xarray as xr

# Abertura do arquivo com o xarray.
ds = xr.open_dataset('GPCP.serie.nc')

# Dado 1D => time=31, lat=1, lon=1
prec = ds['precip'][:,0,0]

# Solução para o erro:
# prec.attrs['units'] = 'mm/day'

num_dias = xclim.atmos.maximum_consecutive_dry_days(prec,
                                                    thresh='3 mm/day',
                                                    freq='MS')

# Total de dias: 4.
print(f'Total de dias: {int(num_dias)}.')
```

Exemplo 2: Uso do índice dry_days

Leitura recomendada:
https://xclim.readthedocs.io/en/stable/indicators_api.html#xclim.indicators.atmos.dry_days
https://xclim.readthedocs.io/en/stable/notebooks/example.html#Threshold-indices

Este índice calcula o número de dias em que a **precipitação é menor que um dado limiar**. A unidade do arquivo deve estar em `mm/day`, caso contrário será retornado erro abaixo. Neste exemplo, utiliza-se o limiar de 3 mm/day.

```
DimensionalityError: Cannot convert from 'millimeter / day' ([length] /
[time]) to 'kilogram / meter ** 2' ([mass] / [length] ** 2)
```

```
import xclim
import xarray as xr

#Abertura do arquivo com o xarray.
ds = xr.open_dataset('GPCP.serie.nc')

# Dado 1D => time=31, lat=1, lon=1
prec = ds['precip'][:,0,0]

# Solução para o erro:
# prec.attrs['units'] = 'mm/day'

num_dias = xclim.atmos.dry_days(prec,
                                thresh = '3.0 mm/day',
                                freq = 'MS')

# Total de dias: 8.
print(f'Total de dias: {int(num_dias)}.')
```

Exemplo 3: Uso do índice maximum_consecutive_warm_days

Leitura recomendada:
https://xclim.readthedocs.io/en/stable/indicators_api.html#xclim.indicators.atmos.maximum_consecutive_warm_days
https://xclim.readthedocs.io/en/stable/notebooks/example.html#Threshold-indices

Este índice calcula o maior número de dias consecutivos em que a temperatura é maior que um dado limiar. A unidade do arquivo deve estar em `C` ou `degC`, caso contrário será retornado erro. Neste exemplo, utiliza-se o limiar de 30 ºC.

```
import xclim
import xarray as xr

# Abertura do arquivo com o xarray.
ds = xr.open_dataset('tmed_serie_Temp_Kelvin.nc')

# Dado 1D => time=31, lat=1, lon=1
temp = ds['TEMP2m'][:,0,0] # Temperatura em Kelvin.

print(temp)  # units: K

tc = temp - 273.15  # Converte de Kelvin para Celsius.
tc.attrs["units"] = "C"  # Altera o atributo "units" para Celsius.

print(tc)  # units: C

num_dias = xclim.atmos.maximum_consecutive_warm_days(tc,
                                                     thresh='30 C',
                                                     freq='MS')

print(f'Total de dias: {int(num_dias)}.')  # Total de dias: 7.
```

Como interpretar erros

Nesta seção serão mostrados alguns erros que ocorrem e como resolvê-los. O Python retorna o tipo de erro e a linha em que ele ocorreu.

Leitura recomendada:
https://docs.python.org/3.9/library/exceptions.html

Erro de sintaxe

Ocorre quando algo foi escrito de forma que o Python não reconhece.

Exemplo 1:

```
print("------------------------------")
printf('Identificando erros no script.')
```

Resultado do erro:

```
------------------------------
Traceback (most recent call last):
  File "ex01.py", line 2, in <module>
    printf('Identificando erros no script.')
NameError: name 'printf' is not defined
```

Neste caso, o erro se refere a digitação incorreta (printf) da função (print) que retorna NameError. Além disso, a linha que ocorreu o erro é mostrada. A solução consiste em remover o f do printf.

Solução do erro:

```
print("------------------------------")
print('Identificando erros no script.')
```

Exemplo 2:

```
File "ex01.py", line 4
    None=3.14
    ^
SyntaxError: cannot assign to None
```

Neste caso, `None` é uma palavra reservada do Python e não pode ser utilizada como variável.

Erro NameError

Ocorre quando uma função ou variável não foi definida.

Exemplo 1:

```
print(Temperatura)  # Imprime a variável Temperatura.
```

Resultado do erro:

```
Traceback (most recent call last):
  File "ex01.py", line 6, in <module>
    print(Temperatura)
NameError: name 'Temperatura' is not defined
```

Neste caso, a variável temperatura não foi declarada.

Exemplo 2:

```
calcula_umidade()  # Chama a função calcula_umidade().
```

Resultado do erro:

```
Traceback (most recent call last):
  File "ex01.py", line 8, in <module>
    calcula_umidade()
NameError: name 'calcula_umidade' is not defined
```

A função foi usada sem ser declarada antes.

Erro TypeError

Ocorre quando tenta-se aplicar uma função, operação ou ação em um tipo errado.

```
print(len(5))
```

Erro retornado:

```
Traceback (most recent call last):
  File "ex01.py", line 10, in <module>
    print(len(5))
TypeError: object of type 'int' has no len()
```

A função `len` deve ser utilizada apenas com iteráveis e não apenas com um valor. O exemplo abaixo mostra o uso correto.

```
x = ['1', '2', '3']
print(len(x))  # 3
```

Erro IndexError

Ocorre quando tenta-se acessar um elemento de uma lista ou outro tipo de dado indexado utilizando um índice inválido.

```
a = ['1', '2', '3']
#     0    1    2
print(a[5])  # Tenta imprimir o índice 5 da lista "a".
```

Resultado do erro:

```
Traceback (most recent call last):
  File "ex01.py", line 18, in <module>
    print(a[5])
IndexError: list index out of range
```

Erro KeyError

Ocorre quando tenta-se acessar um dicionário com uma chave inválida.

```
x = {'Temperatura':'25', 'Umidade Relativa':'80'}  # Dicionário.
```

Resultado do erro:

```
Traceback (most recent call last):
  File "ex01.py", line 21, in <module>
    print(x['Vento'])
KeyError: 'Vento'
```

Erro AttributeError

Ocorre quando uma variável não tem atributo ou função.

```
a = ('1', '2', '3')  # Definindo uma tupla.
print(a.sort())
```

Resultado do erro:

```
Traceback (most recent call last):
  File "ex01.py", line 24, in <module>
    print(a.sort())
AttributeError: 'tuple' object has no attribute 'sort'
```

Erro IndentationError

Ocorre quando uma variável não respeita os espaços de indentação.

Exemplo 1:

```
print('Teste')
```

Resultado do erro:

```
File "ex01.py", line 26
    print('Teste')
IndentationError: unexpected indent
```

A solução consiste em recuar a função print() para a primeira coluna, uma vez que, ela inicia na segunda coluna.

Exemplo 2:

```
for i in range(4):
i = i + 1
```

Resultado do erro:

```
File "ex01.py", line 29
    i = i + 1
    ^
IndentationError: expected an indented block
```

A solução consiste em recuar o trecho “i = i + 1” quatro espaços para evitar o erro. A solução encontra-se abaixo.

```
for i in range(4):
    i = i + 1
```

Baixar dados do ECMWF utilizando processamento paralelo

É possível utilizar o Python para baixar vários arquivos por meio de várias requisições ao servidor do ECMWF utilizando paralelismos. Isso ajuda, porque gasta-se menos tempo para realizar o download dos dados. Inicialmente, será mostrado como criar uma conta porque ela é necessária para realizar o download.

1. Criar uma conta em:

https://cds.climate.copernicus.eu/#!/home

2. Como utilizar a API do ERA5?

Acesse o link: https://cds.climate.copernicus.eu/api-how-to

E crie o arquivo abaixo:

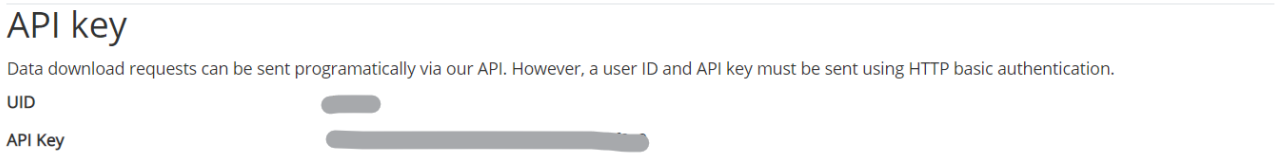
```
.cdsapirc
```

No home (`/home/seu_usuario`) da sua distribuição Linux. O ponto na frente significa arquivo oculto do sistema.

Esse arquivo tem que ter as seguintes linhas:

```
url: https://cds.climate.copernicus.eu/api/v2
key: <a ser preenchido no passo seguinte>
```

Ao logar na sua conta, no canto superior direito, clique no nome do seu usuário. No fim da página, copie os valores da linha API KEY (hachuradas em cinza, copie e cole no arquivo "`.cdsapirc`" no trecho onde tem "`key:`".



3. Instale o cliente "CDS API" digitando o comando abaixo:

```
pip install cdsapi
```

Caso apareça a mensagem abaixo:

```
You are using pip version 9.0.1, however version 19.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Atualize o pip com o comando:

```
sudo pip install --upgrade pip
```

Caso apareça o erro abaixo:

```
ModuleNotFoundError: No module named 'cdsapi'
```

Basta instalar o `pip3`:

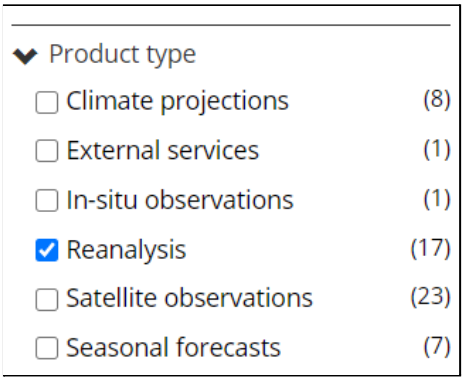
```
pip3 install cdsapi
```

4. Baixando dados do ERA5:

Acessar o link abaixo:

https://cds.climate.copernicus.eu/cdsapp#!/search?type=dataset

E selecionar Reanalysis, como mostrado na figura abaixo:



O exemplo consiste em acessar o link abaixo:

https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels-monthly-means?tab=form

Em *"Product type"*, selecione *"Monthly averaged reanalysis"* e defina os parâmetros de interesse. No fim da página tem a opção *"Show API requeste"*. Ao expandir essa seleção, as informações mostradas referem-se ao script Python para baixar os dados.

O script abaixo deve ser utilizado como modelo para baixar os dados. É necessário realizar algumas adaptações nele porque dependendo da base de dados, ele pode ser horário ou mensal, a superfície ou nível de pressão.

O exemplo utilizou o site:

https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels-monthly-means?tab=form

Link para os dados:

https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-pressure-levels-monthly-means?tab=form

Para realizar o download das variáveis abaixo:

- 10m_u_component_of_wind
- 10m_v_component_of_wind
- 2m_temperature

Para um determinado domínio espacial o `script.py` abaixo faz uma requisição de dados desde 1980 até 2020 de 10 em 10 anos (`max_workers=10`) no script abaixo, altere conforme as suas necessidades), isso ajuda muito para que o processo de download não demore tanto.

```
from concurrent.futures import ThreadPoolExecutor

import cdsapi

# para executar: python3 script.py

# Tem uma dimensão chamada expver nos arquivos NetCDF, isso acaba
# dificultando manipular os dados, a solução foi remover esta
# dimensão com o nco:
# ncwa -a expver ERA5_Superficie_2020.nc ERA5_Superficie_2020_Novo.nc

# A variável area representa a área de interesse com a seguinte ordem:
# Latitude Norte/Longitude Oeste/Latitude Sul/Longitude Leste.

months = ['01', '02', '03', '04', '05', '06',
          '07', '08', '09', '10', '11', '12']

# Nome do arquivo e local onde ele será armazenado.
fname = '../input/mensal_superficie/ERA5_Superficie_{}.nc'.format

with ThreadPoolExecutor(max_workers=10) as e:
    for i in range(1980, 2021):

        year = str(i)

        print(year, '--------------------------')

        c = cdsapi.Client()

        params = {
```

```
        'product_type'   : 'monthly_averaged_reanalysis',
        'format'         : 'netcdf',
        'variable'       : [
                             '10m_u_component_of_wind',
                             '10m_v_component_of_wind',
                             'surface_pressure', '2m_temperature',
                           ],
        'year'           : year,
        'month'          : months,
        'time'           : '00:00',
        'area'           : "10/-90/-40/-20",
    }
    e.submit(c.retrieve, 'reanalysis-era5-single-levels-monthly-means',
                         params,
                         fname(year))
```

Integração entre Python e CDO

A integração do Python com CDO fornece várias possibilidades de processamento.

Leitura recomendada:
https://pypi.org/project/cdo/
https://github.com/Try2Code/cdo-bindings
https://code.mpimet.mpg.de/projects/cdo/wiki/Cdo%7Brbpy%7D
https://code.mpimet.mpg.de/projects/cdo/wiki/Cdo#Documentation
https://code.mpimet.mpg.de/attachments/download/18824/cdo-bindings.pdf

Abaixo são listados alguns exemplos obtidos da página oficial do CDO via tradução para o português:

- **Acesso direto aos dados via numpy/narray:** Com o uso do python-netcdf4 ou scipy será possível ter acesso direto aos valores dos seus campos. Python e Ruby oferecem um rico conjunto de bibliotecas científicas para trabalhar com essas matrizes de dados, por exemplo, gerar um gráfico.
- **Tratamento automático de tempfile:** Ao lidar com dados temporários, você nunca terá que fazer as coisas manualmente. Sem limpeza, sem criação manual e sem renomear arquivos. Para salvar o que se deseja, crie um nome de arquivo de saída e o resto é feito automaticamente.
- **Paralelização flexível:** No caso de um trabalho que utiliza um grande volume de dados e demora bastante, ele pode ser feito em paralelo. Em geral, os shells não oferecem paralelismo suficiente para ter controle refinado. Tanto o Python quanto Ruby oferecem o que você precisa: a possibilidade de executar 1000 chamadas de rotina com apenas 12 processos ou threads simultâneos.
- **Processamento condicional:** Evite o reprocessamento uma vez que os arquivos de saída já se encontram no disco. Isso ajuda a acelerar seu script de forma substancial. Este comportamento pode ser ativado globalmente ou apenas para uma única chamada. Por exemplo, ao analisar os dados de saída de experimentos em execução, é possível executar o mesmo script indefinidamente e sempre obterá os resultados mais recentes.
- **Crie novos operadores:** A geração dos códigos em Python e Ruby são bibliotecas de código aberto que podem ser estendidas no tempo de execução. Caso seja necessário criar um novo operador, basta escrevê-lo em Python ou Ruby.

Alguns exemplos de uso de Python com o CDO:

1. Listar as possibilidades de uso do CDO.

```
from cdo import *

cdo = Cdo()
print(dir(cdo))  # Lista os métodos (operadores) disponíveis.
```

2. Qual a versão do CDO instalada? A partir do comando acima, será acessado o método version.

```
from cdo import *

cdo = Cdo()
print(cdo.version())  # 1.9.9
```

3. Obtendo informações sobre o arquivo.

```
from cdo import *

cdo = Cdo()

cdo.debug = True  # Mostra na tela o que está sendo feito.

print(dir(cdo))  # Lista os métodos (operadores) disponíveis.

cdo.infov(input='input.nc')  # Obtém informações sobre o arquivo.
```

4. Manipulando o arquivo.

```
from cdo import *

cdo = Cdo()

cdo.debug = True  # Mostra na tela o que está sendo feito.

cdo.timmin(input='input.nc', output='out.nc')
```

5. Interpolando um arquivo.

O arquivo `chuva.nc` será interpolado para o mesmo domínio espacial do arquivo `grade.nc` e será gerado o novo arquivo `out.nc`.

```
from cdo import *

cdo = Cdo()

cdo.debug = True  # Mostra na tela o que está sendo feito.

cdo.remapbil('grade.nc', input='chuva.nc', output='out.nc')
```

6. Utilizando opções para manipular o arquivo.

```
from cdo import *

cdo = Cdo()

cdo.debug = True  # Mostra na tela o que está sendo feito.

cdo.seltimestep('3/5',
                input='input.nc',
                output='out.nc',
                options='-r -f nc4')
```

7. Encadeamento de operadores.

```
from cdo import *

cdo = Cdo()

cdo.debug = True  # Mostra na tela o que está sendo feito.

cdo.sellonlatbox('-66,-51,-10,-2',
                 input='-mulc,20 -selmon,3 ' + 'input.nc',
                 output='out.nc',
                 options='-f nc4')
```

8. Criando arquivos temporários

   Quando o output é omitido são criados arquivos temporários e são removidos automaticamente quando o script finaliza. Portanto, a limpeza manual não é mais necessária, a menos que seja encontrada uma falha. Neste caso, remove-se o arquivo temporário por meio do comando abaixo:

   ```
   cdo.cleanTempDir()
   ```

   Exemplo 1 : O arquivo é armazenado na saída padrão, isto é, no "`/tmp`". Após a execução do script o arquivo será removido do computador.

```
from cdo import *

cdo = Cdo()

cdo.debug = True  # Mostra na tela o que está sendo feito.

outfile = cdo.timmin(input='input.nc', options='-f nc')

cdo.infon(input=outfile)
```

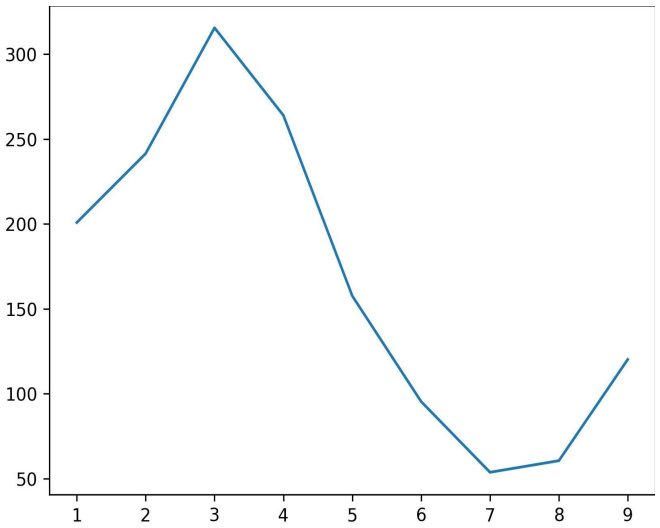
Exemplo 2: Definindo um diretório para armazenar os dados temporários. Após a execução do script o arquivo será removido do computador.

```
from cdo import *

cdo = Cdo()

cdo.debug = True  # Mostra na tela o que está sendo feito.

cdo = Cdo(tempdir='<nome_diretorio_arquivos_temporarios>')

outfile = cdo.timmin(input='.input.nc', options='-f nc')

cdo.infon(input=outfile)  # Visualiza as informações do arquivo.
```

9. Criando um gráfico combinando Python e CDO.

```
from cdo import *
import matplotlib.pyplot as plt

cdo = Cdo()

cdo.debug = True  # Mostra na tela o que está sendo feito.

x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
y = cdo.fldmean(input='prec.2020.nc', returnCdf=True).variables['prec'][:]

plt.plot(x,y[:,0,0])
plt.show()
```

O resultado será:



Outras aplicações utilizando Python

Esta seção apresenta alguns exemplos usualmente utilizados em ciências atmosféricas, com bibliotecas que profissionais da área que sejam iniciantes em Python podem não conhecer. Portanto, metodologias para consulta/download de dados observados (i.e., estações observacionais, satélites, radiossondas, etc.), de modelos atmosféricos globais e regionais, métodos estatísticos aplicados e etc serão descritas a seguir.

Módulo *siphon*

Desenvolvido pela UNIDATA (https://www.unidata.ucar.edu), o objetivo do módulo *siphon* é acessar a base de dados em servidores remotos baseados, na maioria das vezes, no protocolo *Thematic Real-time Environmental Distributed Data Services - THREDDS*. Estes servidores são adotados em diversos institutos de pesquisa científica dos EUA para distribuição de dados ambientais multidimensionais, como o NCAR, NASA, NOAA e NCEP. Estes servidores costumam reunir petabytes de dados como observações, saídas de modelos de previsão numérica de tempo ou de clima, imagens de satélites, bóias marítimas, radar meteorológico, etc. O site oficial do módulo apresenta vários exemplos de seu uso para diversos tipos de dados que pode ser acessado pelo link abaixo:

https://unidata.github.io/siphon/latest/examples/index.html

Todos os scripts a seguir podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/outras_aplicacoes_python

Exemplo 1: Script ex01.py.

Não esquecer de instalar a biblioteca abaixo:

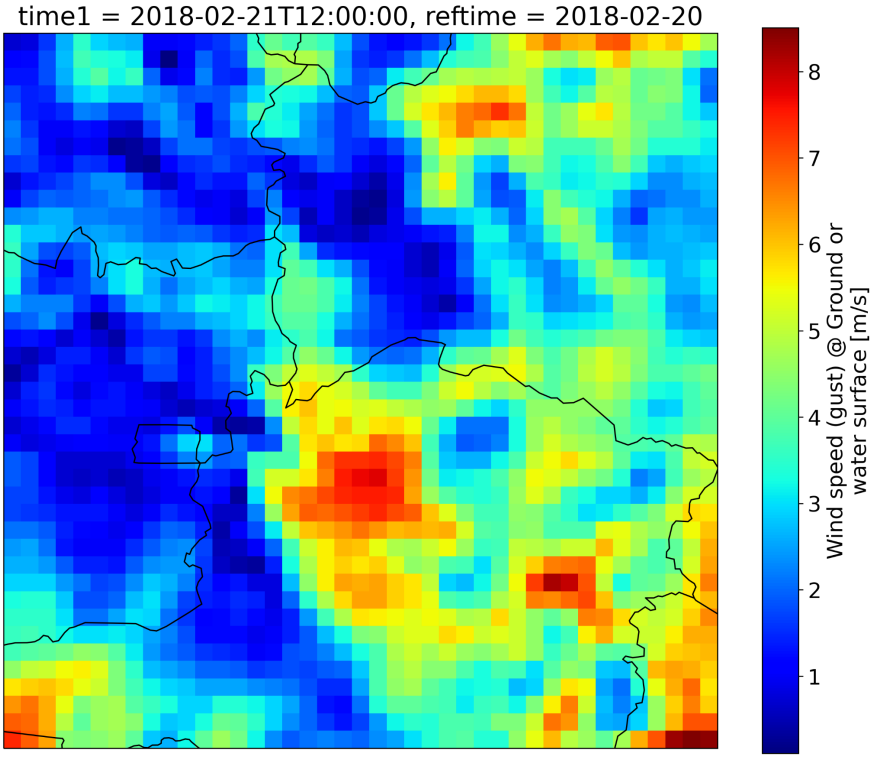
```
pip install siphon
```

Este script faz uma consulta personalizada na base histórica do modelo GFS na resolução de 0,25° (aproximadamente 25 km) usando o *siphon*, salvando o dado no formato NetCDF. Já o código ex02.py faz a leitura do dado salvo no código anterior para confecção de uma figura, cujo resultado é exemplificado abaixo. A principal vantagem deste método é realizar o download de variáveis, datas e coordenadas (no tempo e de espaço) que seja de interesse do usuário, evitando baixar o arquivo completo, visto que atualmente cada arquivo possui aproximadamente 500MB.

Exemplo 2: Script ex02.py.

Este script utiliza o arquivo gerado pelo script ex01.py.

O resultado será:



Obtenção de radiossondas globais usando siphon

A Universidade de Wyoming disponibiliza uma extensa base de radiossondas lançadas em diversas localidades do globo em seu site oficial há vários anos. A partir do link abaixo é possível realizar o acesso:

https://weather.uwyo.edu/upperair/sounding.html

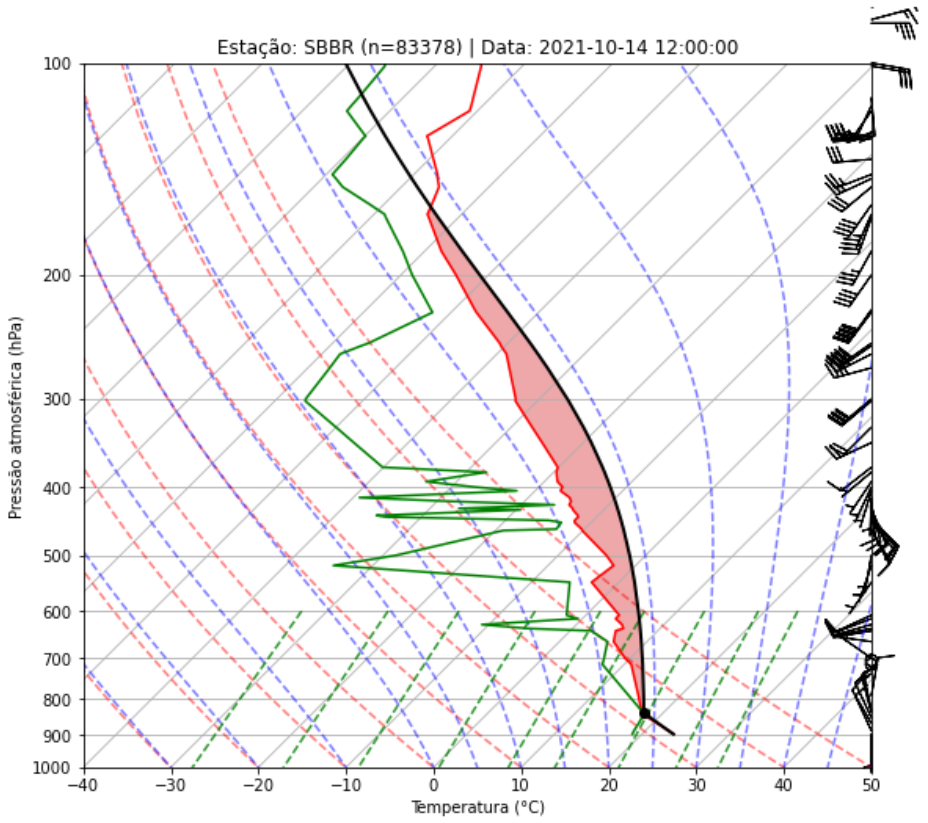
Sendo assim, a consulta destes dados é uma atividade bastante comum entre meteorologistas para diversas finalidades, como previsão do tempo, estudos sinóticos, micrometeorológicos, etc. Um exemplo de uso destas informações é a realização de análises termodinâmicas de um perfil vertical da atmosfera. Portanto, esta seção apresenta uma metodologia de obtenção de radiossondas desta base de dados, usando novamente a biblioteca *siphon*. Em seguida, a biblioteca MetPy é utilizada para gerar um diagrama Skew-T Log-P, bastante usado em análises sinóticas para monitoramento e previsão de tempo.

Exemplo 3: Script ex03.py.

Não esquecer de instalar a biblioteca abaixo:

```
pip install metpy
```

O resultado será:



Módulo Py3GrADS

Uma das principais ferramentas computacionais para processamento e visualização de dados meteorológicos é o GrADS, sendo usado até hoje em diversos centros operacionais e universidades. No entanto, a migração para linguagens de programação mais modernas, como o Python, tem sido cada vez mais comum nos últimos anos. Com isso, esta seção apresenta um algoritmo que faz uso do Py3GrADS para leitura e visualização de previsões numéricas do modelo WRF executadas pelo INPE.

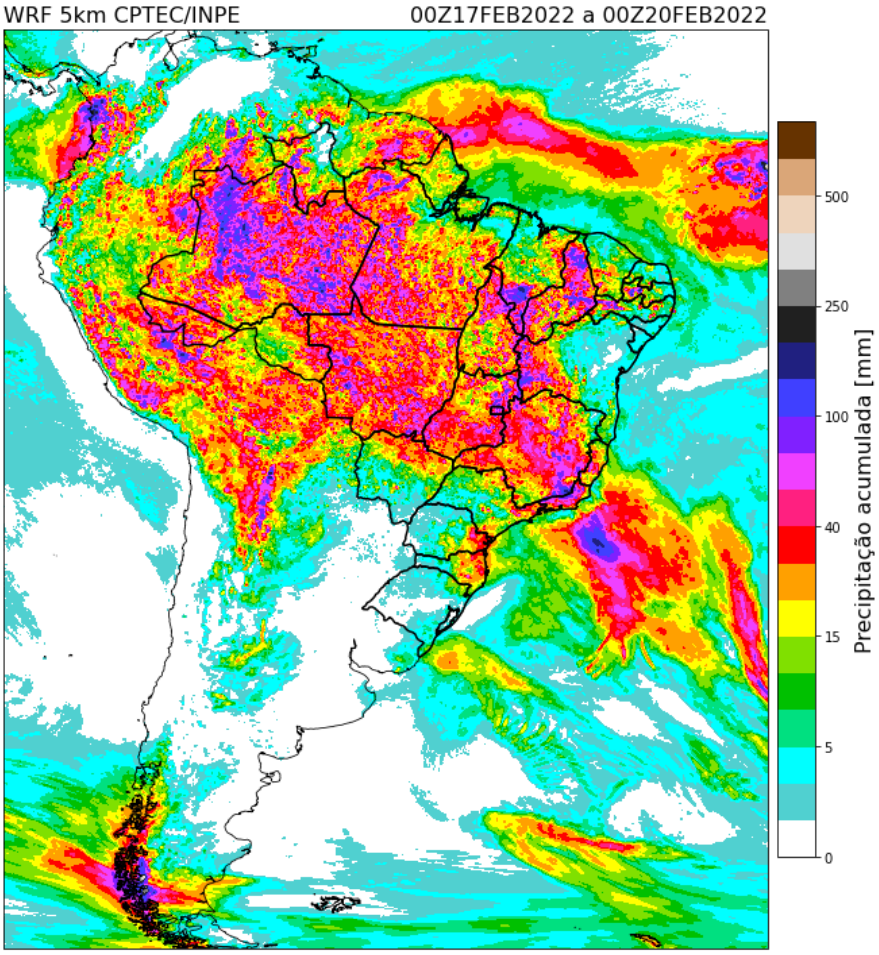
Antes de executar o script ex04.py, leia o arquivo README por meio do link abaixo para preparação do ambiente, configuração do OpenGrADS e obtenção das previsões do WRF do INPE:

https://github.com/jgmsantos/Livro-Python/blob/main/outras_aplicacoes_python/README.md

Com a instalação das dependências e a preparação do ambiente concluída, os dados podem ser processados pelo código. Vale ressaltar que o resultado foi obtido com dados baixados na época posterior ao caso de evento extremo ocorrido em Petrópolis-RJ em 18/fevereiro/2022, o maior da história do município. Além disso, é importante destacar que as previsões operacionais do INPE são disponibilizadas no FTP (visto no link README acima) apenas para os últimos 30 dias.

Exemplo 4: Script ex04.py.

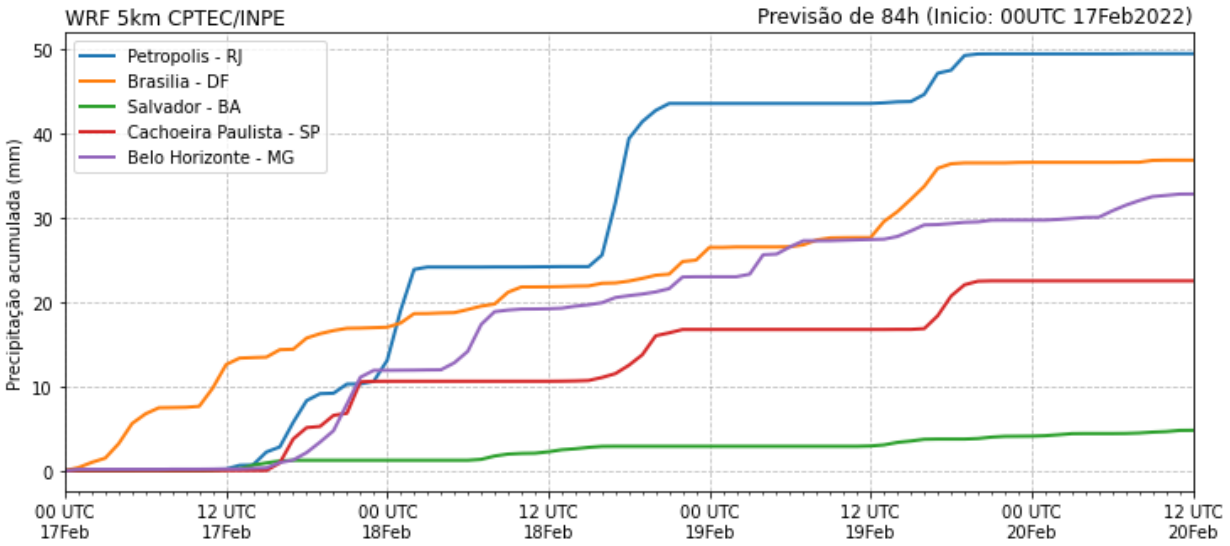
O resultado será:



O próximo exemplo faz a comparação da precipitação horária acumulada em diferentes cidades brasileiras durante o evento ocorrido em Petrópolis-RJ.

Exemplo 5: Script ex05.py.

O resultado será:



Tutoriais para obtenção e processamento de dados de satélites

O blog do GEONETCAST, mantido pelo Diego Souza do INPE, possui uma grande variedade de tutoriais relacionados com Python para diferentes satélites, disponíveis no link abaixo:

https://geonetcast.wordpress.com/gnc-a-product-manipulation-tutorials

a. Download do GOES-R via THREDDS

   https://geonetcast.wordpress.com/2019/05/17/geonetclass-accessing-grb-data-from-unidata-thredds-with-python-part-i/

b. Processamento das imagens Sentinel-3:

   https://geonetcast.wordpress.com/2018/09/26/processing-sentinel-3-data-with-python/

c. Download do LANDSAT-8 via *Amazon Web Services* - AWS:

   http://gallery.pangeo.io/repos/pangeo-data/landsat-8-tutorial-gallery/landsat8.html

Por fim, são apresentados outros módulos com aplicabilidade em várias áreas das geociências.

- **PyWavelets**

    - https://pywavelets.readthedocs.io/en/latest
    - Cálculo para diversos tipos de ondeletas.

- **StatsModels**
    - https://www.statsmodels.org/stable/index.html
    - Modelos estatísticos variados (regressões, análise de séries temporais, etc.).

- **Pangeo**
    - http://gallery.pangeo.io
    - Galeria com diversos exemplos de dados geocientíficos.

- **geoCAT**
    - https://geocat-examples.readthedocs.io/en/latest/gallery/index.html#
    - Reproduzir plots do NCL (*NCAR Command Language)* em Python.

- **pyMannKendall**
    - https://pypi.org/project/pymannkendall
    - Análise de tendência de Mann-Kendall

- **Pangeo-data**
    - https://github.com/pangeo-data/awesome-open-climate-science
    - Dezenas de dicas com módulos e tutoriais para várias áreas das geociências.

- **CMIP6**
    - http://gallery.pangeo.io/repos/pangeo-gallery/cmip6
    - Notebooks com vários exemplos de consulta, processamento e análise dos modelos CMIP6.

- **UNIDATA**
    - https://unidata.github.io/python-training/gallery/gallery-home/
    - Galeria de vários exemplos de plots meteorológicos (Mapas, séries temporais, perfis verticais, etc).

Trabalhando com dados de estação

A visualização de variáveis sobre os mapas é útil porque permite avaliar facilmente a sua distribuição espacial, no entanto, muitas vezes elas são disponibilizadas a partir de estações à superfície.

Neste caso, as informações podem ser avaliadas pontualmente ou a partir de alguma técnica de interpolação que converte os dados irregularmente espaçados em uma grade regular e permite melhot avaliação como também a variação espacial das variáveis analisadas.

Os scripts abaixo fornecem informações disponíveis a partir de estações à superfície por meio do arquivo `data_station.dat` referente a variável temperatura.

As informações do arquivo `data_station.dat` podem ser baixadas a partir do link:

https://portal.inmet.gov.br/paginas/catalogoaut

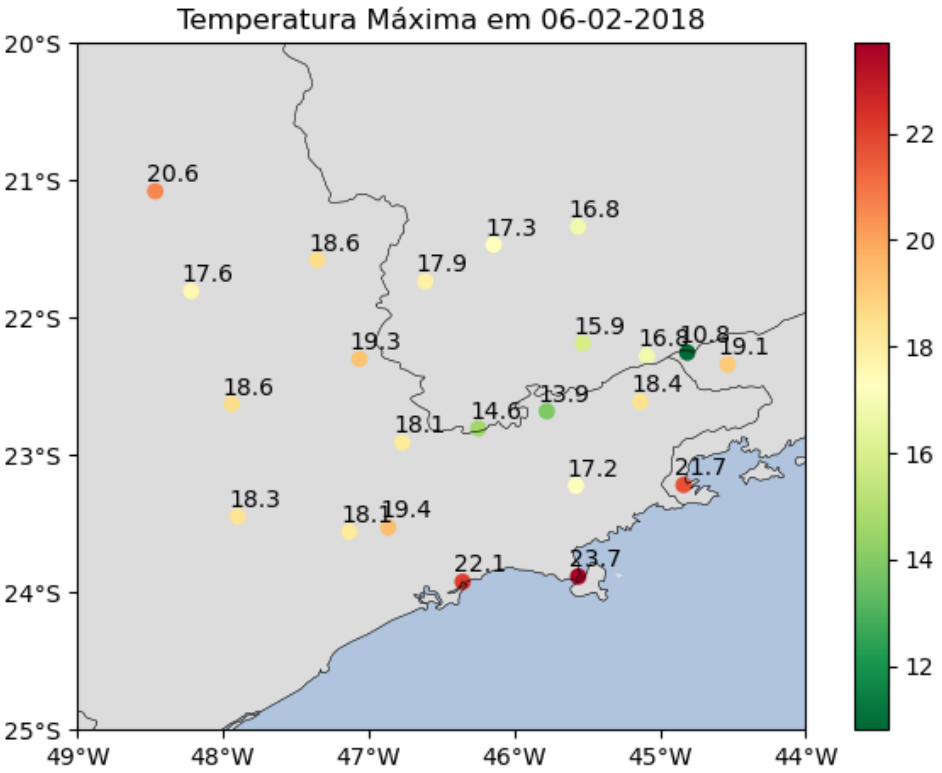
Todos os scripts podem ser acessados pelo link abaixo:

https://github.com/jgmsantos/Livro-Python/tree/main/manipulacao_dados_estacoes

Exemplo 1: Script ex01.py.

Plotando os dados pontuais da variável temperatura

O resultado será:



Exemplo 2: Script ex02.py.

Como já mencionado, o mesmo conjunto de dados pode ser interpolado e apresentado como uma matriz sobre o mapa, com as informações regularmente espaçadas.

O resultado será: