Teoria dos Conjuntos aplicado a Banco de dados

Drª Marise Miranda
Msc Eduardo Verri

Primeiramente, vamos gerar dados para que possamos trabalhar os conjuntos.

Vamos usar libs do Python, para carregar dados de máquina de uma maneira mais profissional. Mas também vamos usar uma API aberta, em python que contém uma estrutura já automatizada usando a lib psutil e a lib cure. Para termos uma referência dos dados que precisamos carregar em nosso banco de dados. Mas ela não será usada no projeto.

Precisamos gerar dados de uma maneira automatizada, mas vamos recordar como é fácil utilizar a lib psutil com comandos de python para obter dados de máquina.

Você já sabe instalar e importar uma lib, foi feito o procedimento com a psutil. Para que serve essa lib, mesmo?

Como estamos usando Microsoft em nosso teste então vamos recordar alguns comandos:

Instalação da biblioteca psutil(**documentação: https://psutil.readthedocs.io/en/latest/#psutil.WindowsService)**

```
$ pip3 install psutil
```

```
Qual versão da lib está instalada? É a mais recente?
```

```
>>> import psutil
```

```
>>> psutil.cpu_times( )
```

métrica: segundos

```
scputimes(user=25922.71875, system=12890.109375, idle=136599.328125, interrupt=816.21875, dpc=1259.0625)
```

```
>>> psutil.cpu_times(True)
```

```
[scputimes(user=6825.828125, system=4662.843749999993, idle=32675.796875, interrupt=659.3125, dpc=841.28125), scputimes(user=5740.109374999999, system=2679.5, idle=35744.578125, interrupt=71.640625, dpc=195.5), scputimes(user=7230.484375, system=2943.2031250000073, idle=33990.49999999999, interrupt=51.21875, dpc=136.078125), scputimes(user=6485.953125, system=2765.53125, idle=34912.703125, interrupt=40.78125, dpc=113.09375)]
```

```
>>> psutil.cpu_times(False)
scputimes(user=26296.4375, system=13057.51562500003, idle=137353.57812499997, interrupt=823.3125, dpc=1286.625)
```

```
>>> psutil.cpu_percent(interval=1, percpu=True)
```

métrica: %

```
[41.5, 34.4, 34.4, 31.2]
```

```
>>> psutil.cpu_count()
4
```

Retorna os núcleos lógicos. Significa o número de núcleos físicos multiplicado pelo número de threads que podem ser executados em cada núcleo (isso é conhecido como Hyper Threading).


```
>>> psutil.cpu_count(logical=False)
2
>>> psutil.cpu_count(False)
2
>>> psutil.cpu_count(True)
4
```

```
>>>_psutil.cpu_freq()
```

Métrica: frequências *atual* , *mínima* e *máxima* expressas em **Mhz**(Mega Hertz).

```
scpufreq(current=2511.0, min=0.0, max=2712.0)
>>> psutil.cpu_freq(percpu=True)
[scpufreq(current=2511.0, min=0.0, max=2712.0)]
```

```
>>> psutil.virtual_memory()
```

Métricas: total e disponível em Bytes)

```
svmem(total=8499134464, available=3723227136, percent=56.2, used=4775907328, free=3723227136)
```

```
>>> psutil.swap_memory()
```

Métricas: **(total - available) / total * 100**

```
sswap(total=11317706752, used=7296192512, free=4021514240, percent=64.5, sin=0, sout=0)
```

```
>>> psutil.disk_partitions()
```

Métricas: sistemas de arquivos do fs.

```
[sdiskpart(device='C:\\', mountpoint='C:\\', fstype='NTFS', opts='rw,fixed'),
sdiskpart(device='D:\\', mountpoint='D:\\', fstype='NTFS', opts='rw,fixed'),
sdiskpart(device='F:\\', mountpoint='F:\\', fstype='FAT32', opts='rw,removable'),
sdiskpart(device='G:\\', mountpoint='G:\\', fstype='FAT32', opts='rw,removable')]
```

```
>>>psutil.disk_usage('/')
```

Métrica: obrigatório ter o path '/'- expressa em Bytes

```
sdiskusage(total=261480951808, used=65910235136, free=195570716672, percent=25.2)
```

```
>>> psutil.disk_io_counters()
```

Métricas: Count é expresso em números, R &W em bytes, time em milisegundos

```
sdiskio(read_count=760150, write_count=1024825, read_bytes=20649329152,
write_bytes=27362576384, read_time=1580, write_time=1464)
```



```
>>> psutil.disk_io_counters(perdisk=True)
```

Métricas por Device:qte, Bytes, milissegundos

```
{'PhysicalDrive0': sdiskio(read_count=760127, write_count=1028141, read_bytes=20635787264,
write_bytes=27443145728, read_time=1579, write_time=1465), 'PhysicalDrive1':
sdiskio(read_count=22, write_count=1, read_bytes=7629824, write_bytes=4096, read_time=0,
write_time=0), 'PhysicalDrive2': sdiskio(read_count=65, write_count=0, read_bytes=7900160,
write_bytes=0, read_time=1, write_time=0)}
```

```
>>> psutil.net_io_counters()
```

Métricas: Bytes, qte pacotes, qte erros, pacotes perdidos

```
snetio(bytes_sent=2179553230, bytes_recv=2746763686, packets_sent=3298168,
packets_recv=3141570, errin=0, errout=0, dropin=0, dropout=0)
```

```
>>> psutil.net_io_counters(pernic=True)
```

```
{'Ethernet': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0,
dropin=0, dropout=0), 'Conexão Local* 1': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0,
packets_recv=0, errin=0, errout=0, dropin=0, dropout=0), 'Conexão Local* 2': snetio(bytes_sent=0,
bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0, dropout=0), 'Wi-Fi':
snetio(bytes_sent=2179638125, bytes_recv=2746921173, packets_sent=3298400,
packets_recv=3141988, errin=0, errout=0, dropin=0, dropout=0), 'Conexão de Rede Bluetooth':
snetio(bytes_sent=0, bytes_recv=0, packets_sent=0, packets_recv=0, errin=0, errout=0, dropin=0,
dropout=0), 'Loopback Pseudo-Interface 1': snetio(bytes_sent=0, bytes_recv=0, packets_sent=0,
packets_recv=0, errin=0, errout=0, dropin=0, dropout=0)}
```

```
>>> psutil.net_connections()
```

Métrica: rastreia todas as conexões

```
[sconn(fd=-1, family=<AddressFamily.AF_INET: 2>, type=<SocketKind.SOCK_STREAM: 1>,
laddr=addr(ip='0.0.0.0', port=445), raddr=(), status='LISTEN', pid=4), sconn(fd=-1,
family=<AddressFamily.AF_INET: 2>, type=<SocketKind.SOCK_DGRAM: 2>, laddr=addr(ip='0.0.0.0',
port=50762), raddr=(), status='NONE', pid=13924), sconn(fd=-1, family=<AddressFamily.AF_INET6:
23>, type=<SocketKind.SOCK_DGRAM: 2>, laddr=addr(ip='::', port=5355), raddr=(), status='NONE',
pid=3160), sconn(fd=-1, family=<AddressFamily.AF_INET: 2>, type=<SocketKind.SOCK_STREAM: 1>,
laddr=addr(ip='127.0.0.1', port=49720), raddr=addr(ip='127.0.0.1', port=5001), status='ESTABLISHED',
pid=2056),.....)]
```

```
>>> psutil.users()
```

```
[suser(name='Marise', terminal=None, host=None, started=1598914821.5034204, pid=None)]
```

```
>>> psutil.pids()
```

```
[0, 4, 56, 108, 252, 392, 420, 524, 596, 608, 708, 716, 720, 780, 800, 808, 932, 956, 980, 988,
1000, 1060, 1108, 1172, 1200, 1308, 1316, 1324, 1364, 1368, 1436, 1452, 1524, 1568, 1588,
1596, 1604, 1656, 1724, 1788, 1824, 2020, 2044, 2056, 2076, 2180, 2184, 2204, 2216, 2248,
2260, 2280, 2288, 2296, 2400, 2408, 2448, 2496, 2516, 2576, 2584, 2692, 2728, 2736, 2756,
```

2776, 2848, 2984, 3000, 3008, 3044, 3120, 3136, 3144, 3160, 3176, 3196, 3240, 3300, 3376, 3400, 3448, 3452, 3488, 3512, 3560, 3672, 3752, 3844, 3892, 3972, 3976, 4000, 4008, 4016, 4044, 4056, 4068, 4080, 4100, 4124, 4132, 4140, 4164, 4220, 4248, 4308, 4320, 4356, 4448, 4504, 4520, 4544, 4600, 4696, 4736, 4856, 5056, 5060, 5092, 5188, 5240, 5276, 5344, 5380, 5408, 5412, 5472, 5500, 5512, 5536, 5708, 5716, 5732, 5888, 5904, 5912, 6016, 6308, 6360, 6464, 6528, 6756, 6932, 6968, 6972, 7008, 7140, 7176, 7204, 7216, 7364, 7396, 7448, 7488, 7504, 7628, 7768, 7800, 7828, 7836, 7920, 7972, 8024, 8080, 8096, 8284, 8348, 8444, 8464, 8472, 8488, 8604, 8700, 8844, 9036, 9148, 9396, 9624, 9748, 9896, 10004, 10060, 10108, 10168, 10208, 10372, 10384, 10496, 10580, 10588, 10712, 10724, 10852, 10880, 10992, 11204, 11248, 11272, 11400, 11436, 11492, 11548, 11628, 11808, 11852, 11884, 11924, 11940, 12076, 12120, 12264, 12400, 12576, 12660, 12676, 12744, 12752, 12780, 12908, 12940, 13020, 13572, 13844, 13924, 14412, 14428, 14572, 14592, 14908, 14988, 15000, 15008]

```
>>> for proc in psutil.process_iter(['pid', 'name', 'username']):
...         print(proc.info)
...
{'pid': 0, 'username': 'NT AUTHORITY\\SYSTEM', 'name': 'System Idle Process'}
{'pid': 4, 'username': 'NT AUTHORITY\\SYSTEM', 'name': 'System'}
{'pid': 56, 'username': None, 'name': ''}
{'pid': 108, 'username': None, 'name': 'Registry'}
{'pid': 252, 'username': None, 'name': 'audiodg.exe'}
{'pid': 392, 'username': 'DESKTOP-OQ1491A\\Marise', 'name': 'chrome.exe'}
{'pid': 420, 'username': None, 'name': 'smss.exe'}
{'pid': 524, 'username': None, 'name': 'winlogon.exe'}
{'pid': 596, 'username': None, 'name': 'svchost.exe'}
{'pid': 608, 'username': None, 'name': 'csrss.exe'}
{'pid': 708, 'username': None, 'name': 'wininit.exe'}
{'pid': 716, 'username': None, 'name': 'csrss.exe'}
{'pid': 720, 'username': None, 'name': 'fontdrvhost.exe'}
Etc etc etc }

>>> procs = {p.pid: p.info for p in psutil.process_iter(['name', 'username'])}
>>> procs
```

{0: {'username': 'NT AUTHORITY\\SYSTEM', 'name': 'System Idle Process'}, 4: {'username': 'NT AUTHORITY\\SYSTEM', 'name': 'System'}, 56: {'username': None, 'name': ''}, 108: {'username': None, 'name': 'Registry'}, 252: {'username': None, 'name': 'audiodg.exe'}, 388: {'username': None, 'name': 'SearchFilterHost.exe'}, 392: {'username': 'DESKTOP-OQ1491A\\Marise', 'name': 'chrome.exe'}, 420: {'username': None, 'name': 'smss.exe'}, 524: {'username': None, 'name': 'winlogon.exe'}, 596: {'username': None, 'name': 'svchost.exe'}, 608: {'username': None, 'name': 'csrss.exe'}, 708: {'username': None, 'name': 'wininit.exe'}, 716: {'username': None, 'name': 'csrss.exe'}, 720: {'username': None, 'name': 'fontdrvhost.exe'}, 780: {'username': None, 'name': 'services.exe'}, 800: {'username': None, 'name': 'LsaIso.exe'}, 808: {'username': None, 'name': 'lsass.exe'}, 932: {'username': None, 'name': 'svchost.exe'}, 956: {'username': None, 'name': 'svchost.exe'}, 980: {'username': None, 'name': 'WUDFHost.exe'}, 988: {'username': None, 'name': 'fontdrvhost.exe'}, ...

```
>>> p = psutil.Process()
>>> with p.oneshot():
...         p.name()
...         p.cpu_times()
...         p.cpu_percent()
...         p.create_time()
...         p.ppid()
...          p.status()
...
'python3.8.exe'
pcputimes(user=0.953125, system=7.125, children_user=0.0, children_system=0.0)
0.0
1598985445.1833448
5060
'running'

>>> psutil.Process().exe()
'C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.8_3.8.1520.0_x64__qbz5n2kfra8p0\\python3.8.exe'

>>> psutil.Process().cmdline()
```

['C:\\Program Files\\WindowsApps\\PythonSoftwareFoundation.Python.3.8_3.8.1520.0_x64__qbz5n2kfra8p0\\python3.8.exe']

```
>>> psutil.Process().environ()
```

Linux	Windows
cpu_num()	cpu_percent()
cpu_percent()	cpu_times()
cpu_times()	io_counters()
create_time()	memory_info()
name()	memory_maps()
ppid()	num_ctx_switches()
status()	num_handles()
terminal()	num_threads()
	username()
gids()	
num_ctx_switches()	exe()
num_threads()	name()
uids()	
username()	
memory_full_info()	
memory_maps()	

Vamos ver como uma API em Python funciona para capturar os dados da máquina

Projeto Glances (requer Psutil)

Docs:

https://glances.readthedocs.io/en/stable/quickstart.html

https://docs.python.org/pt-br/dev/library/curses.html#module-curses

https://pip.pypa.io/en/stable/installing/

Verifique a versão do pip

Abra o prompt de comando:

C:\Users\Marise Miranda>pip --version

pip 20.2.2 from c:\users\marise miranda\appdata\local\programs\python\python38-32\lib\site-packages\pip (python 3.8)

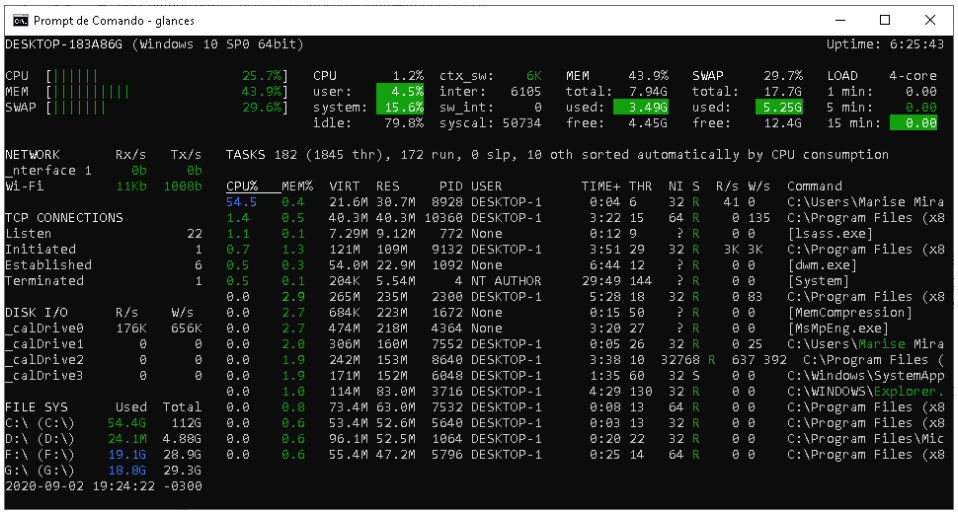
$ pip install glances

$ pip install windows-curses

$ glances

Agora a tela da API aparece com a captura em tempo real dos dados de máquina:

Esta tela é de um computador Windows que tem pequenas diferenças no computador com SO Linux.



Vamos instalar no Linux usando Conteiner WSL

Entre com o WSL Ubuntu (material de Sistemas Operacionais)

No usuário:

$ sudo apt update && sudo apt upgrade

Estes comandos vão instalar e atualizar os pacotes do Ubuntu xx

$ sudo apt install python3

$ python3 --version

$sudo apt-get update

$Python 3.6.9

$ sudo apt upgrade python3

$ pip install update python3

$ sudo apt install python3-pip

```
$ sudo apt install python3 python3-pip ipython3
```

```
$ pip install windows_curses-2.1.0-cp38-cp38-win_amd64.whl
```

$ sudo apt install python-pip

Entre no dretório mnt em C, vá em users e vá até o diretório Python

```
$sudo apt install glances
```

```
$pip install windows-curses
```