Importação/Manipulação de Dados com Python

Benilton Carvalho, Tatiana Benaglia, Fernanda Schumacher

Fatos sobre Python

- Linguagem interpretada, orientada a objeto;
- Possui estruturas de dados de alto nível;
- Usado para scripts ou, apenas, como conector entre diferentes ferramentas;
- Enfatiza a habilidade de leitura do código;
- Utiliza "pacotes", estimulando a modularidade de código;
- Ao contrário do R, não existe um foco (central da linguagem) na análise de dados;
- Gratuito e disponível na maior parte das plataformas.

Pandas

- Biblioteca do Python voltada para a estruturação de dados;
- Permite a leitura e escrita de dados em vários formatos: CSVs, Excel, SQL, entre outros;
- Alta performance, código aberto;
- Permite a realização de análise de dados e modelagem (torna possível a análise sem precisar mudar para R);
- Pandas não implementa estratégias avançadas de análise de dados;
- Estratégias mais elaboradas de análise de dados estão disponíveis em:
  - Statsmodels: http://www.statsmodels.org
  - Scikit-Learn: https://scikit-learn.org
  - TensorFlow: https://www.tensorflow.org/

Mais informações: https://pandas.pydata.org/

Pandas Cheat Sheet: https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

Pandas

Pontos Positivos

- Extremamente fácil de usar, exigindo uma pequena curva de aprendizagem para lidar com dados tabulares;
- Conjunto grande de ferramentas para carregar, transformar e escrever dados em vários formatos;
- Principal escolha para a maioria das bibliotecas usadas em Machine Learning;
- Capacidade para gráficos e visualizações de dados (usando matplotlib);

Ponto Negativo

- Grande ocupação da memória, pois cria muitos objetos adicionais para rápido acesso e facilidade de manipulação;

Instalação do Python e Pandas

No Linux,

```
sudo apt-get install python3 python3-pip
sudo pip3 install pandas
```

No MacOS,

```
brew install python3 python3-pip
pip3 install pandas
```

No Windows,

- Baixar o instalar em https://www.python.org/downloads/windows/
- Marcar a opção `add to PATH` no começo do processo
- `python` e `pip` ficarão disponíveis no Windows PowerShell.
- Executar:

  ```
  pip install pandas
  ```

Chamando o Python a partir do R

- Você pode, também, usar o Python a partir do R;
- Cenários assim são comuns quando você precisa conectar ferramentas disponíveis em cada uma das linguagens;
- Por exemplo, para criar as notas de aula em `RMarkdown`;

- Para acessar o Python, você deve utilizar o pacote `reticulate`. Para instalar o pacote:

```
install.packages("reticulate")
```

- O código a ser executado em Python deve estar contido num `chunk` específico para `python`;

Chamando o Python a partir do R

```
library(reticulate)
py_discover_config() # Qual versão do Python está instalada?
```

```
## python:         /Users/tatiana/Library/r-miniconda/envs/r-reticulate/bin/p
## libpython:      /Users/tatiana/Library/r-miniconda/envs/r-reticulate/lib/l
## pythonhome:     /Users/tatiana/Library/r-miniconda/envs/r-reticulate:/User
## version:        3.6.12 | packaged by conda-forge | (default, Dec  9 2020,
## numpy:          /Users/tatiana/Library/r-miniconda/envs/r-reticulate/lib/p
## numpy_version:  1.19.4
```

Você pode especificar a versão a ser usada:

```
use_python("/usr/local/bin/python3")
```

Referência: McKinney (2013) *Python for data analysis*, O'Reilly. O livro é do criador do pacote `pandas`, Wes McKinney.

Python como uma calculadora

Tutorial: https://docs.python.org/3/tutorial/

```
# Comentários são precedidos por #
dir() # Lista variáveis no workspace
```

```
## ['R', '__annotations__', '__builtins__', '__doc__', '__loader__', '__name_
```

```
a = 5 # int
print(a) # Desnecessário em modo interativo
```

```
## 5
```

```
a + 2
```

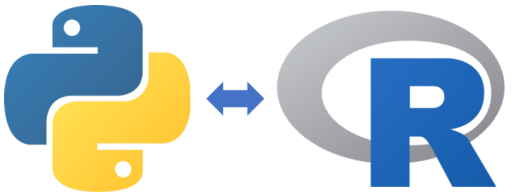
```
## 7
```

```
a / 2
```

```
## 2.5
```

Python vs R

Se olharmos lado a lado, ambos funcionam de maneira similar e são bem parecidos, apenas com algumas pequenas diferenças em suas sintaxes.



Tipos de Dados:

```
# Python
type()
type(5) #int
type(5.5) #float
type('Hello') #string
type(True) #bool
```

```
# R
class()
class(5) #numeric
class(5.5) #numeric
class('Hello') #character
class(True) #logical
```

Assinalando variáveis:

```
# Python
a = 5
```

```
# R
a <- 5
```

Python vs R

Operadores algébricos e lógicos são iguais!

```
# Python                                  # R
+ - / *                                   + - / *


# The same goes for logical operators
< #less than                              < #less than
> #greater than                           > #greater than
<= #less than or equal to                 <= #less than or equal to
== #is equal to                           == #is equal to
!= #is not equal to                       != #is not equal to
& #and                                    & #and
| #or                                     | #or
```

```
a == 5
```

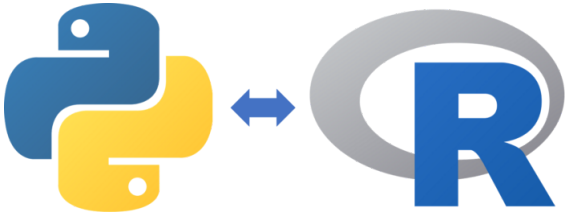
```
## True
```

```
a < 3
```

```
## False
```

Listas e Vetores

Em Python, uma **lista** é uma coleção mutável de elementos de qualquer tipo. O índice de uma lista em Python começa em 0 e é não inclusivo.



Em R, um **vetor** é uma coleção mutável de elementos do mesmo tipo. O índice de um vetor em R começa em 1 e é inclusivo.

```
# Python                                  # R
ls = [1, 'a', 3, False]                   vc <- c(1, 2, 3, 4)


# Python indexing starts at 0, R indexing starts at 1
b = ls[0]                                 b = vc[1]
print(b) #returns 1                       print(b) #returns 1


c = ls[0:1]                               c = vc[1:2]
print(c) #returns 1                       print(c) #returns 1, 2
```

Os objetos carregam seus próprios métodos

```
b = "ME315" # str
print(b)
# help(b)  ## ajuda: mas nao dentro do RStudio
# help(matplotlib) ## ajuda: tambem para pacotes
```

```
## ME315
```

```
print(b.endswith("A"))
```

```
## False
```

```
print(b.endswith("5"))
```

```
## True
```

Indexando Vetores

Índices de vetores no Python começam no 0. Strings são vetores de caracteres.

```
print(b)
```

```
## ME315
```

```
print(b[0]) # First letter
```

```
## M
```

```
print(b[1]) # Second letter
```

```
## E
```

```
print(b[-1]) # Last letter
```

```
## 5
```

Indexando Vetores

Referência parcial a objetos e concatenação.

```
print(b[:2])
```

```
## ME
```

```
print(b[2:])
```

```
## 315
```

```
print(b[:2] + b[2:])
```

```
## ME315
```

Vetores numéricos

Vetores em Python são guardados em listas.

```
x = [1, 2, 4, 8, 16]
print(x)
```

```
## [1, 2, 4, 8, 16]
```

```
print(x[-2:]) # Últimos 2 elementos
```

```
## [8, 16]
```

```
x2 = x + [32, 64, 128] # Concatenação
print(x2)
```

```
## [1, 2, 4, 8, 16, 32, 64, 128]
```

Mais sobre listas

```
print(x2)
```

```
## [1, 2, 4, 8, 16, 32, 64, 128]
```

```
x2[0] = 3 # Modificar valores
print(x2)
```

```
## [3, 2, 4, 8, 16, 32, 64, 128]
```

```
del x2[0] # Remover valores
print(x2)
```

```
## [2, 4, 8, 16, 32, 64, 128]
```

Listas de listas

```
a = ["a","b","c"]
n = [12, 15, 7]
x = [a,n] # Listas de listas
print(x)
```

```
## [['a', 'b', 'c'], [12, 15, 7]]
```

Note: no R, `data.frames` são listas de listas; é errado, do ponto de vista de programação, pensar em `data.frames` como planilhas (i.e. acessar linhas é *lento*, acessar colunas é relativamente rápido).

```
print(x[1]) # Acesso à segunda lista
```

```
## [12, 15, 7]
```

```
print(x[1][0])
```

```
## 12
```

Controle de Fluxo

O python naturalmente possui as ferramentas usuais de controle de fluxo, tais como: **if, for, while**

If/Else

```
# Python
if True:
    print('Hello World!')
else:
    print('Not true!')
```

```
# R
if (TRUE) {
    print('Go to sleep!')
} else {
    print('Not true!')
}
```

For loops

```
# Python
for i in range(2, 5):
    a = i
```

```
# R
for(i in 1:10) {
    a <- i }
```

Estrutura de programação é um tópico complexo. Iremos apenas ilustrar o uso das ferramentas com alguns exemplos.

IF

Importante: Identação é feita com tabulações (`\t`) ou dois espaços, e é parte da sintaxe!

```
x = 2
if x < 0:
  x = 0
  print('Negative changed to zero')
elif x == 0:
  print('Zero')
elif x == 1:
  print('Single')
else:
  print('More')
```

```
## More
```

FOR

Importante: Identação é feita com tabulações (`\t`) ou dois espaços, e é parte da sintaxe!

```
words = ['Matemática', 'Estatística', 'Ciência de Dados']
for w in words:
  print(w, len(w))
```

```
## Matemática 10
## Estatística 11
## Ciência de Dados 16
```

Ainda sobre FOR

```
words = ['Matemática', 'Estatística', 'Ciência de Dados', 'IMECC']
for w in words:
  if len(w) >= 6:
    print(w[:3] + '...')
  else:
    print(w)
```

```
## Mat...
## Est...
## Ciê...
## IMECC
```

DEF

A função `def` permite definir funções. Note o escopo!

```
def wins(arr, howMany): # Média Winsorized
  """Linha com documentacao"""
  if len(arr) < 2*howMany:
    return NaN # error
  else:
    arr.sort()
    for i in range(0, howMany, 1):
      arr[i] = arr[howMany]
      arr[-(i+1)] = arr[-(howMany+1)]
    result = sum(arr)/len(arr)
    return result

a = [4, 7, 3, 4, 5, 2, 1, 6, 999]
print(wins(a, 2))
```

```
## 4.444444444444445
```

```
a = [4, 7, 3, 4, 5, 2, 1, 6, 999] # Escopo!!
print(sum(a)/len(a))
```

Manipulação de Dados

- Em R, o `tidyverse` é a biblioteca que vocês utilizaram para carregar e manipular bancos de dados usando o objeto `data.frame`.
- Em python, `pandas` é a biblioteca equivalente ao `tidyverse`, mais comumente usada para carregar e manipular data frames usando o objeto `DataFrame`.

```python
# Python
import pandas as pd


# load and view data
df = pd.read_csv('path.csv')

df.head()

df.sample(100)

df.describe()


# write to csv
df.to_csv('exp_path.csv')
```

```r
# R
library(tidyverse)


df <- read_csv('path.csv')

head(df)

sample(df, 100)

summary(df)


write_csv(df, 'exp_path.csv')
```

Lendo CSV em Python, pandas

Primeiramente, `import` evoca pacotes e tem função similar a `library()` e/ou `require()` no `R`.

```
import numpy as np #para matrizes e arrays
import matplotlib.pyplot as plt
import pandas as pd #para dataframes
print(pd.__version__) # Checks pandas version, we need 0.18 at least
```

```
## 1.1.5
```

Baby names

Dados são da SSA (Social Security Agency), mas eu só consegui baixá-los de https://github.com/hadley/data-baby-names/blob/master/baby-names.csv.

```
#R
babyNamesR = readr::read_csv("../dados/baby-names.csv") %>% as.data.f
babyNamesR %>% head(n=3)
```

```
##   year    name  percent sex
## 1 1880    John 0.081541 boy
## 2 1880 William 0.080511 boy
## 3 1880   James 0.050057 boy
```

```
#python
babyNamesPY = pd.read_csv("../dados/baby-names.csv", header = 0)
babyNamesPY.head()
```

```
##    year     name   percent  sex
## 0  1880     John  0.081541  boy
## 1  1880  William  0.080511  boy
## 2  1880    James  0.050057  boy
## 3  1880  Charles  0.045167  boy
## 4  1880   George  0.043292  boy
```

Em R

```
class(babyNamesR)
```

```
## [1] "data.frame"
```

```
class(py$babyNamesPY)
```

```
## [1] "data.frame"
```

Em python

```
type(babyNamesPY)
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
type(r.babyNamesR)
```

```
## <class 'pandas.core.frame.DataFrame'>
```

Em python

```
babyNamesPY.shape
```

```
## (258000, 4)
```

```
babyNamesPY.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 258000 entries, 0 to 257999
## Data columns (total 4 columns):
##  #   Column   Non-Null Count   Dtype
## ---  ------   --------------   -----
##  0   year     258000 non-null  int64
##  1   name     258000 non-null  object
##  2   percent  258000 non-null  float64
##  3   sex      258000 non-null  object
## dtypes: float64(1), int64(1), object(2)
## memory usage: 7.9+ MB
```

Selecionando colunas

```
print(babyNamesPY.year) #ou print(babyNamesPY['year'])
```

```
## 0         1880
## 1         1880
## 2         1880
## 3         1880
## 4         1880
##           ...
## 257995    2008
## 257996    2008
## 257997    2008
## 257998    2008
## 257999    2008
## Name: year, Length: 258000, dtype: int64
```

Apply functions on DataFrames

```
df = pd.DataFrame(
  {'A': [1, 2, 3],
   'B':[4, 5, 6]
  })
df.apply(np.mean,axis=0)
```

```
## A    2.0
## B    5.0
## dtype: float64
```

```
df.apply(np.mean,axis=1)
```

```
## 0    2.5
## 1    3.5
## 2    4.5
## dtype: float64
```

Dados do SSA contêm somente os 1000 nomes mais comuns de cada ano....

```
print(babyNamesPY.groupby(['year','sex']).name.count())
```

```
## year  sex
## 1880  boy     1000
##       girl    1000
## 1881  boy     1000
##       girl    1000
## 1882  boy     1000
##               ...
## 2006  girl    1000
## 2007  boy     1000
##       girl    1000
## 2008  boy     1000
##       girl    1000
## Name: name, Length: 258, dtype: int64
```

Alguns verbos coincidem com `dplyr`

```
print(babyNamesPY.groupby(['year','sex']).percent.sum())
```

```
## year  sex
## 1880  boy     0.930746
##       girl    0.934546
## 1881  boy     0.930439
##       girl    0.932690
## 1882  boy     0.927532
##                 ...
## 2006  girl    0.684830
## 2007  boy     0.801105
##       girl    0.677453
## 2008  boy     0.795414
##       girl    0.672516
## Name: percent, Length: 258, dtype: float64
```

Indexando (linhas)

Não é possível indexar diretamente um DataFrame, você precisa acessar o atributo `iloc`

```
print(babyNamesPY.iloc[0]) # ou print(babyNamesPY.iloc[0,:])
```

```
## year           1880
## name           John
## percent    0.081541
## sex             boy
## Name: 0, dtype: object
```

```
print(babyNamesPY.iloc[0:3])
```

```
##    year     name   percent  sex
## 0  1880     John  0.081541  boy
## 1  1880  William  0.080511  boy
## 2  1880    James  0.050057  boy
```

No (significant number of) boys named Sue...

```
print(babyNamesPY.loc[babyNamesPY.name == "Sue",])
```

```
##         year name   percent   sex
## 129189  1880  Sue  0.000666  girl
## 130185  1881  Sue  0.000678  girl
## 131171  1882  Sue  0.000726  girl
## 132216  1883  Sue  0.000566  girl
## 133194  1884  Sue  0.000669  girl
## ...      ...  ...       ...   ...
## 229543  1980  Sue  0.000193  girl
## 230654  1981  Sue  0.000152  girl
## 231777  1982  Sue  0.000116  girl
## 232885  1983  Sue  0.000096  girl
## 233984  1984  Sue  0.000082  girl
##
## [105 rows x 4 columns]
```

Indexando (linhas e colunas)

```
print(babyNamesPY.iloc[0,0])
```

```
## 1880
```

```
print(babyNamesPY.loc[0,'name'])
```

```
## John
```

```
print(babyNamesPY.loc[[0,10,100],['name','year']])
```

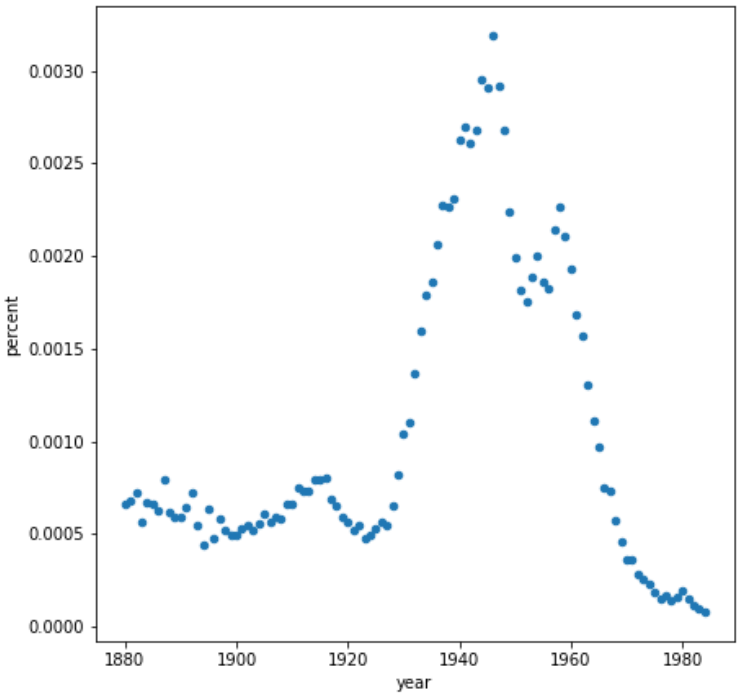
```
##         name  year
## 0       John  1880
## 10   Edward  1880
## 100    Perry  1880
```

```
print(babyNamesPY.index)
```

```
## RangeIndex(start=0, stop=258000, step=1)
```

Plot usando matplotlib

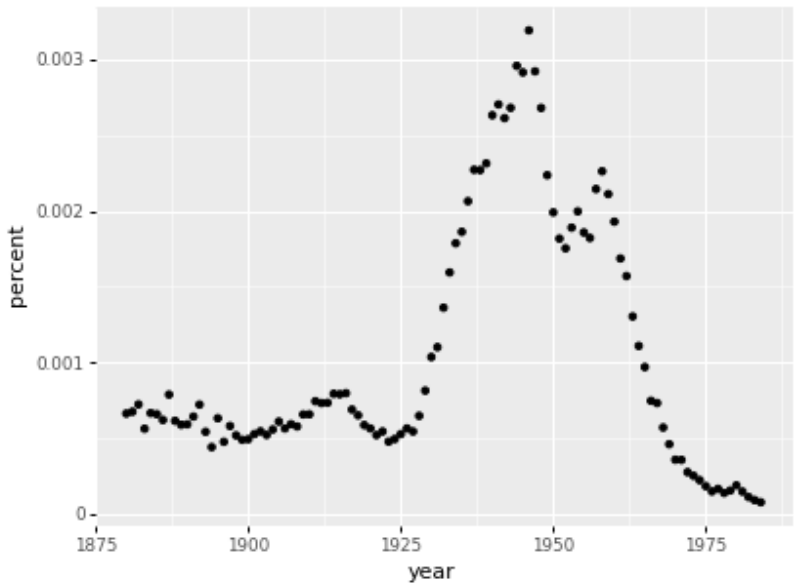
```
babySue = babyNamesPY.loc[babyNamesPY.name == "Sue",]
babySue.plot(kind = 'scatter', x = 'year', y = 'percent')
plt.show() # from matplotlib
```



Plot usando ggplot

```
from plotnine import *
ggplot(babySue) + geom_point(aes(x = 'year', y = 'percent'))
```

```
## <ggplot: (-9223372036508940449)>
```



Formatos Suportados pelo Pandas

Format Type	Data Description	Reader	Writer
text	CSV (*)	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	

Observações:

- `read_csv` possui o argumento `delimiter`, que pode ser ajustado para outros tipos de arquivos em texto plano retangulares;
- `read_csv` também possui o argumento `chunksize`, que pode ser usado para leitura por partes.

Formatos Suportados pelo Pandas

Format Type	Data Description	Reader	Writer
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

Pandas e Chunks

```
reader = pd.read_csv("../dados/baby-names.csv",
        header = 0, chunksize=1000)
soma = 0
for df in reader:
  soma += df.loc[df.name == "Sue", 'percent'].sum()

print(soma)
```

```
## 0.10973800000000006
```

SQLite, Pandas e Python

```
import pandas as pd
import sqlite3
conn = sqlite3.connect("../dados/disco.db")
pd.read_sql_query("SELECT * FROM artists LIMIT 5", conn)
```

```
##    ArtistId                Name
## 0         1              AC/DC
## 1         2              Accept
## 2         3           Aerosmith
## 3         4  Alanis Morissette
## 4         5    Alice In Chains
```

```
conn.close()
```

MongoDB, Pandas e Python

```
from pymongo import MongoClient
import pprint
myurl = "mongodb+srv://fernandaBD:mongo123@cluster0.2ph3s.mongodb.net
client = MongoClient(myurl)
db = client['me315mongodb']
collection = db['diamantes']
collection
```

```
## Collection(Database(MongoClient(host=['cluster0-shard-00-02.2ph3s.mongodb.
```

MongoDB

```
doc = collection.find_one()
pprint.pprint(doc)
```

```
## {'_id': ObjectId('5fd034c6e17a0000d50063aa'),
##  'carat': 0.22,
##  'clarity': 'VS2',
##  'color': 'E',
##  'cut': 'Fair',
##  'depth': 65.1,
##  'price': 337,
##  'table': 61.0,
##  'x': 3.87,
##  'y': 3.78,
##  'z': 2.49}
```

MongoDB

```
doc = collection.find_one({"cut":"Premium"})
pprint.pprint(doc)
```

```
## {'_id': ObjectId('5fd034c6e17a0000d50063ae'),
##  'carat': 0.22,
##  'clarity': 'SI1',
##  'color': 'F',
##  'cut': 'Premium',
##  'depth': 60.4,
##  'price': 342,
##  'table': 61.0,
##  'x': 3.88,
##  'y': 3.84,
##  'z': 2.33}
```

MongoDB

```
doc = collection.find({"cut":"Premium"}).limit(5)
for x in doc:
  pprint.pprint(x,width=10)
```

```
## {'_id': ObjectId('5fd034c6e17a0000d50063ae'),
##  'carat': 0.22,
##  'clarity': 'SI1',
##  'color': 'F',
##  'cut': 'Premium',
##  'depth': 60.4,
##  'price': 342,
##  'table': 61.0,
##  'x': 3.88,
##  'y': 3.84,
##  'z': 2.33}
## {'_id': ObjectId('5fd034c6e17a0000d50063da'),
##  'carat': 0.3,
##  'clarity': 'SI2',
##  'color': 'J',
##  'cut': 'Premium',
##  'depth': 59.3,
##  'price': 405,
```