

centro
universitário



Programação Científica

Prof. Dr. Danilo H. Perico



pandas





- Pandas (*Python Data Analysis*) é uma biblioteca criada para a linguagem Python para a manipulação e análise de dados
- Em particular, oferece estruturas e operações para manipular tabelas numéricas e séries temporais
- Usada para grandes quantidades de dados
 - *Data Mining* e *Machine Learning*
- É um software livre

<https://pandas.pydata.org/>



*The **Pandas** library is built on **NumPy** and provides easy-to-use data structures and data analysis tools for the Python programming language.*



pandas: a Foundational Python Library for Data Analysis and Statistics

Wes McKinney

Abstract—In this paper we will discuss **pandas**, a Python library of rich data structures and tools for working with structured data sets common to statistics, finance, social sciences, and many other fields. The library provides integrated, intuitive routines for performing common data manipulations and analysis on such data sets. It aims to be the foundational layer for the future of statistical computing in Python. It serves as a strong complement to the existing scientific Python stack while implementing and improving upon the kinds of data manipulation tools found in other statistical programming languages such as R. In addition to detailing its design and features of **pandas**, we will discuss future avenues of work and growth opportunities for statistics and data analysis applications in the Python language.

Introduction

Python is being used increasingly in scientific applications traditionally dominated by [R], [MATLAB], [Stata], [SAS], other commercial or open-source research environments. The maturity and stability of the fundamental numerical libraries ([NumPy], [SciPy], and others), quality of documentation, and availability of “kitchen-sink” distributions ([EPD], [Pythonly]) have gone a long way toward making Python accessible and convenient for a broad audience. Additionally [matplotlib] integrated with [IPython] provides an interactive research and development environment with data visualization suitable for most users. However, adoption of Python for applied statistical modeling has been relatively slow compared with other areas of computational science.

One major issue for would-be statistical Python programmers in the past has been the lack of libraries implementing standard models and a cohesive framework for specifying models. However, in recent years there have been significant new developments in econometrics ([StaM]), Bayesian statistics ([PyMC]), and machine learning ([SciL]), among others fields. However, it is still difficult for many statisticians to choose Python over R given the domain-specific nature of the

a table or spreadsheet of data will likely have labels for the columns and possibly also the rows. Alternately, some columns in a table might be used for grouping and aggregating data into a pivot or contingency table. In the case of a time series data set, the row labels could be time stamps. It is often necessary to have the labeling information available to allow many kinds of data manipulations, such as merging data sets or performing an aggregation or “group by” operation, to be expressed in an intuitive and concise way. Domain-specific database languages like SQL and statistical languages like R and SAS have a wealth of such tools. Until relatively recently, Python had few tools providing the same level of richness and expressiveness for working with labeled data sets.

The **pandas** library, under development since 2008, is intended to close the gap in the richness of available data analysis tools between Python, a general purpose systems and scientific computing language, and the numerous domain-specific statistical computing platforms and database languages. We not only aim to provide equivalent functionality but also implement many features, such as automatic data alignment and hierarchical indexing, which are not readily available in such a tightly integrated way in any other libraries or computing environments to our knowledge. While initially developed for financial data analysis applications, we hope that **pandas** will enable scientific Python to be a more attractive and practical statistical computing environment for academic and industry practitioners alike. The library’s name derives from *panel data*, a common term for multidimensional data sets encountered in statistics and econometrics.

While we offer a vignette of some of the main features of interest in **pandas**, this paper is by no means comprehensive. For more, we refer the interested reader to the online documentation at <http://pandas.pydata.org/pandas.html>.

Corpus ID: 61539023

pandas: a Foundational Python Library for Data Analysis and Statistics

Wes McKinney · Published 2011 · Computer Science

In this paper we will discuss **pandas**, a Python library of rich data structures and tools for working with structured data sets common to statistics, finance, social sciences, and many other fields. The library provides integrated, intuitive routines for performing common data manipulations and analysis on such data sets. It aims to be the foundational layer for the future of statistical computing in Python. It serves as a strong complement to the existing scientific Python stack while... Expand

[\[PDF\]](#) dlr.de [Save to Library](#) [Create Alert](#) [Cite](#)

851 Citations

3 References

Related Papers

Share This Paper [Twitter](#) [Facebook](#) [Google](#) [Email](#)

851 Citations

Highly Influential Citations 57

Background Citations 99

Methods Citations 323

Results Citations 1

[View All](#)

https://dlr.de/sc/portaldata/15/resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf



- Fácil uso e aprendizado da biblioteca
- Muito usado para ***Data Science***
- Pandas é uma ótima biblioteca para realizar análises exploratórias dos dados
- Fortemente otimizada para desempenho, a biblioteca tem base na linguagem C



- **Séries** (*séries*):
 - Ferramentas para geração de intervalo de dados e conversão de frequência, estatísticas, entre outras
- **DataFrames** (*estrutura de dados tabular*):
 - Ferramentas para manipulação de dados, com indexação integrada.
- Ferramentas para ler e escrever dados entre diferentes estruturas de dados e formatos de arquivos



pandas Características

- Alinhamento de dados e manipulação de dados ausentes
- Reformatação e pivoteamento de matrizes
- Divisão (slicing)
- Facilidades para inserir e retirar colunas em conjuntos de dados
- Ferramentas para fundir ou juntar conjuntos de dados
- Filtragem e limpeza de dados

pandas versus NumPy

- Semelhante ao NumPy, fornece estruturas e ferramentas de análise de dados de alto desempenho e fáceis de usar
- Ao contrário da biblioteca NumPy, que fornece objetos para arrays multidimensionais, o Pandas fornece um **objeto de tabela** 2D na memória: o **Dataframe**



pandas Instalação e Importação

- Instalando:
 - Pandas já vem pré-instalado na maioria dos sistemas
 - Caso necessite instalar, use:
 - `pip install pandas`
- Importando:
 - A biblioteca deve ser importada sempre da seguinte maneira:
 - `import pandas as pd`
 - É uma regra de convenção que facilita identificar a biblioteca.



- **Séries:**
 - É um conjunto de dados escalares
 - **É um array de 1 dimensão**
 - Também pode ser visto como uma coluna de uma tabela
- **DataFrames:**
 - É um conjunto de Séries
 - É uma estrutura de dados de 2 dimensões — colunas e linhas
 - **É uma tabela de dados, semelhante a uma planilha Excel**



pandas Séries

- Uma Série é como um array unidimensional, uma lista de valores
- Toda Série possui um índice, o `index`, que dá rótulos a cada elemento da lista
- O objeto Série em pandas é chamado `Series`



Criando uma Série

- Podemos criar um objeto série usando a função `Series()`:

```
import pandas as pd

notas = pd.Series([2,7,5,10,6])
print(notas)
print(type(notas))
```

```
0    2
1    7
2    5
3    10
4    6
dtype: int64
<class 'pandas.core.series.Series'>
```



Criando uma Série

- Também podemos criar uma Série a partir de um array de NumPy:

```
import numpy as np
import pandas as pd

dados = np.array([5,7,4,2,8,9,10])
s = pd.Series(dados)
s
```

0	5
1	7
2	4
3	2
4	8
5	9
6	10

dtype: int32



Atributos das Séries

- Podemos verificar os atributos **values** e **index** da Série:

```
notas.values
```

```
array([ 2,  7,  5, 10,  6], dtype=int64)
```

```
notas.index
```

```
RangeIndex(start=0, stop=5, step=1)
```



Séries com índices específicos

- Como ao criar a Série notas não demos um índice específico, o Pandas usou os inteiros positivos crescentes como padrão
- Em muitos casos pode ser conveniente atribuirmos índices diferentes do padrão



Séries com índices específicos

- Supondo que as notas sejam notas de uma turma, poderíamos atribuir nomes ao **index**:

```
import pandas as pd

notas = pd.Series([2,7,5,10,6], index=["Paulo", "Jorge", "Henrique", "Renato", "Carmem"])
print(notas)

Paulo      2
Jorge      7
Henrique   5
Renato     10
Carmem    6
dtype: int64
```



Séries com índices específicos

- **index** permite acessar os valores pelo seu rótulo:

bracket notation

```
notas["Paulo"]
```

2

```
notas["Carmem"]
```

6

dot notation

```
notas.Henrique
```

5

```
notas.Renato
```

10



pandas Funções Estatísticas

- Existe uma grande quantidade de funções para realizar estatísticas sobre os dados
- Muitas baseadas nas do *NumPy*
 - `mean()`: Calcula a média dos dados
 - `std()`: Calcula o desvio padrão
 - `describe()`: Resumo estatístico dos dados



pandas Exemplo

```
print("Média:", notas.mean())
```

Média: 6.0

```
print("Desvio padrão:", notas.std())
```

Desvio padrão: 2.9154759474226504

```
print( notas.describe() )
```

count	5.000000
mean	6.000000
std	2.915476
min	2.000000
25%	5.000000
50%	6.000000
75%	7.000000
max	10.000000
dtype:	float64



Outras Funções

[abs\(\)](#)

Return a Series/DataFrame with absolute numeric value of each element.

[argmax\(\[axis, skipna\]\)](#)

Return int position of the largest value in the Series.

[argmin\(\[axis, skipna\]\)](#)

Return int position of the smallest value in the Series.

[max\(\[axis, skipna, level, numeric_only\]\)](#)

Return the maximum of the values for the requested axis.

[min\(\[axis, skipna, level, numeric_only\]\)](#)

Return the minimum of the values for the requested axis.

[round\(\[decimals\]\)](#)

Round each value in a Series to the given number of decimals.



Exemplo

```
print( notas.max() )  
print( notas.argmax() )
```

```
10  
3
```

```
print( notas.min() )  
print( notas.argmin() )
```

```
2  
0
```



- Um **DataFrame** é uma **estrutura de dados tabular**
- Um **DataFrame** é **bidimensional**, como uma planilha
- É um conjunto de **Series**
- Os nomes das colunas podem ser usados para acessar seus valores
- O objeto Data Frame em Pandas é chamado **DataFrame**



DataFrame - Criação

- Podemos criar um **DataFrame** a partir de um **array de NumPy** (2x3):

```
import numpy as np
import pandas as pd

A = np.array( [[ 1., 2., 3.],[4., 5., 6.]] )
A

array([[1., 2., 3.],
       [4., 5., 6.]])
```

```
df = pd.DataFrame(A)
df
```

	0	1	2
0	1.0	2.0	3.0
1	4.0	5.0	6.0



DataFrame - Metadata

- DataFrame metadata:

```
df = pd.DataFrame(A)
df
```

	0	1	2
0	1.0	2.0	3.0
1	4.0	5.0	6.0

index

columns

O **Dataframe de pandas** tem rótulos extras para as linhas e colunas chamados de ***index*** e ***columns***.
Estes são os metadados de um dataframe



DataFrame - Criação

- DataFrame metadata:

A screenshot of a Jupyter Notebook cell. The code `df = pd.DataFrame(A)` is shown at the top, followed by the output of `df` which is a DataFrame. The DataFrame has two rows labeled 0 and 1, and three columns labeled 0, 1, and 2. A green arrow points from the word "index" to the row labels 0 and 1. Another green arrow points from the word "columns" to the column labels 0, 1, and 2. The data values are: Row 0: [1.0, 2.0, 3.0]; Row 1: [4.0, 5.0, 6.0].

	0	1	2
0	1.0	2.0	3.0
1	4.0	5.0	6.0

index

columns

Os metadados podem coincidir com a **indexação do NumPy**, *mas nem sempre é assim*



DataFrame - Criação

- DataFrame metadata:

```
df.columns = ['C1', 'C2', 'C3']
df.index = ['L1', 'L2']
df
```

	C1	C2	C3
L1	1.0	2.0	3.0
L2	4.0	5.0	6.0

index

columns

Podemos alterar os metadados



DataFrame - Criação

- Podemos também criar um **DataFrame** que possui valores de diferentes tipos, usando um **dicionário** como entrada de dados:

```
df = pd.DataFrame({  
    'Aluno': ["Paulo", "Jorge", "Henrique", "Renato", "Carmem"],  
    'Faltas' : [3,4,2,1,4],  
    'Prova' : [2,7,5,10,6],  
    'Seminário': [8.5,7.5,9.0,7.5,8.0]})
```



DataFrame

- Para visualizar o Dataframe em um notebook (ipynb) basta colocar o nome do Dataframe e executar a célula
- Podemos usar também o método `display(df)`

```
df
```

	Aluno	Faltas	Prova	Seminário
0	Paulo	3	2	8.5
1	Jorge	4	7	7.5
2	Henrique	2	5	9.0
3	Renato	1	10	7.5
4	Carmem	4	6	8.0



DataFrame - dtypes

- Podemos consultar os tipos de dados de cada coluna:

```
df.dtypes
```

```
Aluno        object
Faltas       int64
Prova        int64
Seminário   float64
dtype: object
```



DataFrame - colunas

- Podemos consultar os metadados das colunas:

```
df.columns
```

```
Index(['Aluno', 'Faltas', 'Prova', 'Seminário'], dtype='object')
```



DataFrame - describe

Usado para visualizar detalhes estatísticos básicos como percentil, média, desvio padrão, valor mínimo, valor máximo etc.

df.describe()

	Faltas	Prova	Seminário
count	5.00000	5.000000	5.00000
mean	2.80000	6.000000	8.10000
std	1.30384	2.915476	0.65192
min	1.00000	2.000000	7.50000
25%	2.00000	5.000000	7.50000
50%	3.00000	6.000000	8.00000
75%	4.00000	7.000000	8.50000
max	4.00000	10.000000	9.00000



DataFrame - ordenação

- Uma facilidade muito importante é a função para ordenação dos DataFrames
- `sort_values()` :
 - *Sort a Series in ascending or descending order by some criterion.*
 - **by** : Name or list of names to sort by
 - **ascending** : If True, sort values in ascending order, otherwise descending
- Usar o método `sort_values` não modifica o DataFrame original



DataFrame - ordenação

```
df.sort_values(by="Prova")
```

	Aluno	Faltas	Prova	Seminário
0	Paulo	3	2	8.5
2	Henrique	2	5	9.0
4	Carmem	4	6	8.0
1	Jorge	4	7	7.5
3	Renato	1	10	7.5

```
df.sort_values(by="Prova", ascending=False)
```

	Aluno	Faltas	Prova	Seminário
3	Renato	1	10	7.5
1	Jorge	4	7	7.5
4	Carmem	4	6	8.0
2	Henrique	2	5	9.0
0	Paulo	3	2	8.5



DataFrame - colunas

- Os nomes das colunas podem ser usadas para acessar seus valores:

```
df[ "Seminário" ]
```

```
0    8.5  
1    7.5  
2    9.0  
3    7.5  
4    8.0
```

```
Name: Seminário, dtype: float64
```

```
df.Seminário
```

```
0    8.5  
1    7.5  
2    9.0  
3    7.5  
4    8.0
```

```
Name: Seminário, dtype: float64
```



DataFrame - selecionar pelo index

- Para selecionar pelo *index* ou *rótulo* usamos o atributo `.loc`:

Exemplo - *index* da linha:

```
df.loc[2]
```

Aluno	Henrique
Faltas	2
Prova	5
Seminário	9.0
Name: 2, dtype: object	



DataFrame - selecionar pelo index

- Com o `.loc` também podemos fazer buscas no DataFrame:

```
df.loc[df.Faltas > 2]
```

Aluno	Faltas	Prova	Seminário
-------	--------	-------	-----------

0	Paulo	3	2	8.5
1	Jorge	4	7	7.5
4	Carmem	4	6	8.0



DataFrame - selecionar pelo index

- Com o `.loc` também podemos selecionar pelo *rótulo* no DataFrame:

Mudando os índices para serem os Alunos:

```
df2 = df.set_index("Aluno")
```

```
df2
```

Aluno	Faltas	Prova	Seminário
Paulo	3	2	8.5
Jorge	4	7	7.5
Henrique	2	5	9.0
Renato	1	10	7.5
Carmem	4	6	8.0

Exemplo com *rótulo* da linha:

```
df2.loc["Renato"]
```

```
Faltas      1.0
Prova       10.0
Seminário   7.5
Name: Renato, dtype: float64
```



DataFrame - selecionar pelo index

- Para selecionar pelo index usamos o atributo `.iloc`:

```
df.iloc[3]
```

```
Aluno      Renato
Faltas          1
Prova          10
Seminário     7.5
Name: 3, dtype: object
```



pandas

Diferença entre loc e iloc

```
df2 = df.sort_values(by="Seminário")
```

```
df2
```

	Aluno	Faltas	Prova	Seminário
1	Jorge	4	7	7.5
3	Renato	1	10	7.5
4	Carmem	4	6	8.0
0	Paulo	3	2	8.5
2	Henrique	2	5	9.0



Diferença entre loc e iloc

```
df2.loc[3]
```

```
Aluno      Renato
Faltas      1
Prova       10
Seminário   7.5
Name: 3, dtype: object
```

```
df2.iloc[3]
```

```
Aluno      Paulo
Faltas     3
Prova      2
Seminário  8.5
Name: 0, dtype: object
```



Acessando dados

```
df
```

	Aluno	Faltas	Prova	Seminário
0	Paulo	3	2	8.5
1	Jorge	4	7	7.5
2	Henrique	2	5	9.0
3	Renato	1	10	7.5
4	Carmem	4	6	8.0

```
df.loc[2, 'Aluno']
```

```
'Henrique'
```

```
df.loc[[1,2], ['Aluno', 'Prova']]
```

	Aluno	Prova
1	Jorge	7
2	Henrique	5



pandas Acessando dados - Boolean Indexing

- Para selecionar dados de acordo com critérios condicionais, usa-se o que se chama de *Boolean Indexing*
- Se quisermos selecionar apenas as linhas em que o valor da coluna **Seminário** é maior do que **8.0**, podemos realizar esta tarefa passando a condição diretamente como índice



pandas

Acessando dados - Boolean Indexing

bracket notation

```
df[df[ "Seminário"] > 8.0]
```

	Aluno	Faltas	Prova	Seminário
0	Paulo	3	2	8.5
2	Henrique	2	5	9.0

dot notation

```
df[df.Seminário > 8.0]
```

	Aluno	Faltas	Prova	Seminário
0	Paulo	3	2	8.5
2	Henrique	2	5	9.0



Acessando dados - Boolean Indexing

- Este tipo de indexação também possibilita verificar condições em múltiplas colunas com o uso dos operadores `&` (e) e `|` (ou)

```
df[(df.Seminário > 8.0) & (df.Prova > 3)]
```

Aluno	Faltas	Prova	Seminário
-------	--------	-------	-----------

2	Henrique	2	5	9.0
---	----------	---	---	-----



Acessando dados

- Para acessar um dado por uma característica podemos usar comparação por igualdade

```
df[df.Aluno == 'Carmem']
```

Aluno	Faltas	Prova	Seminário
-------	--------	-------	-----------

4	Carmem	4	6	8.0
---	--------	---	---	-----



Manipulação de dados

- Pode-se fazer todo tipo de manipulação para explorar os dados
- Slicing e Reshaping das matrizes, como no Numpy, por exemplo
- Aplicar operações matemáticas básicas, como soma, subtração etc.



Manipulação de dados

- Adicionando 2 a todos os valores presentes em uma Série **s**:

```
s2 = s.add(2)
```

```
s2
```

```
0    7  
1    9  
2    6  
3    4  
4   10  
5   11  
6   12  
dtype: int32
```

s	
0	5
1	7
2	4
3	2
4	8
5	9
6	10
	dtype: int32



pandas Manipulação de dados

- Subtraindo 2 de todos os valores:
 - `s.sub(2)`
- Multiplicando todos os valores por 2:
 - `s.mul(2)`
- Dividindo valores por 2:
 - `s.div(2)`



Manipulação de dados

```
df2
```

	Aluno	Faltas	Prova	Seminário
1	Jorge	4	7	7.5
3	Renato	1	10	7.5
4	Carmem	4	6	8.0
0	Paulo	3	2	8.5
2	Henrique	2	5	9.0

```
df2.Faltas = df2.Faltas.sub(1)  
df2
```

	Aluno	Faltas	Prova	Seminário
1	Jorge	3	7	7.5
3	Renato	0	10	7.5
4	Carmem	3	6	8.0
0	Paulo	2	2	8.5
2	Henrique	1	5	9.0



Removendo linhas ou colunas

- Removendo linhas pelo index:
 - `df.drop([0, 1])`
- Removendo colunas utilizando o argumento axis:
 - `df.drop('Aluno', axis=1)`
- Nenhuma destas ações remove a linha ou coluna do DataFrame original: é preciso salvar o resultado em um outro Dataframe



pandas

Removendo linhas ou colunas

Removendo a(s) linha(s)

```
df3 = df2.drop([0,1])  
df3
```

	Aluno	Faltas	Prova	Seminário
3	Renato	0	10	7.5
4	Carmem	3	6	8.0
2	Henrique	1	5	9.0

Removendo a coluna

```
df3 = df2.drop('Aluno', axis=1)  
df3
```

	Faltas	Prova	Seminário
1	3	7	7.5
3	0	10	7.5
4	3	6	8.0
0	2	2	8.5
2	1	5	9.0



Removendo linhas ou colunas

- Para fazer uma alteração no próprio DataFrame, podemos utilizar o parâmetro `inplace=True`

df2				
	Aluno	Faltas	Prova	Seminário
0	Paulo	3	2	8.5
1	Jorge	4	7	7.5
2	Henrique	2	5	9.0
3	Renato	1	10	7.5
4	Carmem	4	6	8.0

```
df2.drop('Aluno', axis=1, inplace=True)
```

df2

	Faltas	Prova	Seminário
0	3	2	8.5
1	4	7	7.5
2	2	5	9.0
3	1	10	7.5
4	4	6	8.0



pandas Funções Customizadas

- É comum querermos aplicar uma função qualquer aos dados, ou à parte deles
- Para isto o pandas fornece o método `apply()`



Funções Customizadas

- Por exemplo, podemos fazer uma função para deixar os nomes dos alunos com apenas as suas três primeiras letras:

```
def truncar(aluno):  
    return aluno[:3]  
  
df.Aluno.apply(truncar)
```

```
0    Pau  
1    Jor  
2    Hen  
3    Ren  
4    Car  
Name: Aluno, dtype: object
```



Entrada e Saída de Dados

- Na maioria das vezes, queremos analisar dados que já estão prontos e guardados em arquivos
- A biblioteca Pandas fornece uma série de funcionalidades de leitura e escrita de dados, para os mais diversos formatos estruturais de dados existentes no mercado



pandas

Entrada e Saída de Dados

- Tipos de dados mais comumente usados:
 - ***Comma Separated Values (CSV):***
 - Arquivo texto com dados separados por vírgulas, com uma entrada por linha
 - Formato de dados aberto muito usado devido a facilidade de manipulação entre diferentes sistemas
 - O caractere separador pode ser diferente da vírgula
 - **Excel:**
 - Formato proprietário da Microsoft



pandas

Entrada e Saída de Dados

- Tipos de dados mais comumente usados:
 - **JSON (*JavaScript Object Notation*)**
 - Um modelo simples com a capacidade de estruturar informações de uma forma bem mais compacta do que a conseguida pelo modelo XML
 - **HTML:**
 - Linguagem de marcação em arquivo texto no qual os sites de internet são escritos
 - **SQL:**
 - Formato de banco de dados relacionais



Leitura de Arquivos em Pandas

- As funções mais usadas em Pandas são:
- `pd.read_csv()`
 - para ler arquivos .csv
- `read_excel()`
 - para ler arquivos do Excel
- `pd.read_html()`
 - para ler tabelas diretamente de um website



pandas Escrita em Arquivos pelo Pandas

- As funções mais usadas em Pandas são:
- `pd.to_csv()`
 - para escrever arquivos .csv
- `pd.to_excel()`
 - para escrever em arquivos Excel
- `pd.to_html()`
 - para escrever em tabelas diretamente de um website



Leitura Excel

- Podemos usar o parâmetro `sheet_name` para selecionar a planilha do documento

A screenshot of Microsoft Excel. On the left, there's a table with columns labeled 'Aluno', 'Faltas', 'Prova', 'Seminário', and 'Média'. The data rows are: Wilfred (3, 2, 8,4, 5,2), Abbie (4, 7, 7,4, 7,2), Harry (2, 5, 9,1, 7,1), Julia (1, 10, 7,4, 8,7), and Carrie (4, 6, 8,1, 7,1). The bottom navigation bar shows tabs for 'turma1' (which is selected) and 'turma2'.

	Aluno	Faltas	Prova	Seminário	Média
2	Wilfred	3	2	8,4	5,2
3	Abbie	4	7	7,4	7,2
4	Harry	2	5	9,1	7,1
5	Julia	1	10	7,4	8,7
6	Carrie	4	6	8,1	7,1
7					

A screenshot of a Jupyter Notebook cell. The code is:

```
import pandas as pd
df = pd.read_excel("notas_e_faltas.xlsx", sheet_name='turma1')
df
```

The resulting output is a DataFrame:

	Aluno	Faltas	Prova	Seminário	Média
0	Wilfred	3	2	8.4	5.20
1	Abbie	4	7	7.4	7.20
2	Harry	2	5	9.1	7.05
3	Julia	1	10	7.4	8.70
4	Carrie	4	6	8.1	7.05



Leitura Excel

- Podemos usar o parâmetro `usecols` para selecionar as colunas

	A	B	C	D	E	F
1	Aluno	Faltas	Prova	Seminário	Média	
2	Wilfred	3	2	8,4	5,2	
3	Abbie	4	7	7,4	7,2	
4	Harry	2	5	9,1	7,1	
5	Julia	1	10	7,4	8,7	
6	Carrie	4	6	8,1	7,1	
7						

```
import pandas as pd
df = pd.read_excel("notas_e_faltas.xlsx", usecols=['Aluno', 'Faltas', 'Média'])
df
```

	Aluno	Faltas	Média
0	Wilfred	3	5.20
1	Abbie	4	7.20
2	Harry	2	7.05
3	Julia	1	8.70
4	Carrie	4	7.05



Leitura CSV



Dataset:

https://www.kaggle.com/datasets/rubenssjr/brasilian-houses-to-rent?select=houses_to_rent_v2.csv

- Dataset com dados de casas para alugar no ano de 2020
- Possui 10692 casas para alugar com 13 diferentes características



```
df = pd.read_csv("houses_to_rent_v2.csv")
```

- O DataFrame tem muitas linhas de dados
- Para visualizar sucintamente as primeiras linhas de um DataFrame existe o método `.head()`
- Similarmente o `.tail()` exibe por padrão as últimas 5 linhas do DataFrame

```

import pandas as pd

df = pd.read_csv("houses_to_rent_v2.csv")

```

df

	city	area	rooms	bathroom	parking spaces	floor	animal	furniture	hoa (R\$)	rent amount (R\$)	property tax (R\$)	fire insurance (R\$)	total (R\$)
0	São Paulo	70	2	1	1	7	acept	furnished	2065	3300	211	42	5618
1	São Paulo	320	4	4	0	20	acept	not furnished	1200	4960	1750	63	7973
2	Porto Alegre	80	1	1	1	6	acept	not furnished	1000	2800	0	41	3841
3	Porto Alegre	51	2	1	0	2	acept	not furnished	270	1112	22	17	1421
4	São Paulo	25	1	1	0	1	not accept	not furnished	0	800	25	11	836

```
df.head()
```

	city	area	rooms	bathroom	parking spaces	floor	animal	furniture	hoa (R\$)	rent amount (R\$)	property tax (R\$)	fire insurance (R\$)	total (R\$)
0	São Paulo	70	2	1	1	7	acept	furnished	2065	3300	211	42	5618
1	São Paulo	320	4	4	0	20	acept	not furnished	1200	4960	1750	63	7973
2	Porto Alegre	80	1	1	1	6	acept	not furnished	1000	2800	0	41	3841
3	Porto Alegre	51	2	1	0	2	acept	not furnished	270	1112	22	17	1421
4	São Paulo	25	1	1	0	1	not accept	not furnished	0	800	25	11	836



pandas Manipulação de Dados

- Pode-se fazer todo tipo de manipulação para explorar os dados
 - `value_counts()` :
 - É usada para obter de uma série a quantidade de valores únicos
 - `groupby()` :
 - Agrupa o DataFrame por uma série de colunas
 - `pivot_table()` :
 - Retorna o DataFrame remodelado organizado por valores de índice/coluna fornecidos



value_counts

Quantos imóveis por cidade

```
df.city.value_counts()
```

São Paulo	5887
Rio de Janeiro	1501
Belo Horizonte	1258
Porto Alegre	1193
Campinas	853
Name: city, dtype: int64	

Quantos imóveis aceitam animais

```
df.animal.value_counts()
```

acept	8316
not acept	2376
Name: animal, dtype: int64	



value_counts

Quantos imóveis por cidade (percentual)

```
df.city.value_counts(normalize=True)
```

```
São Paulo      0.550599
Rio de Janeiro 0.140385
Belo Horizonte 0.117658
Porto Alegre   0.111579
Campinas        0.079779
Name: city, dtype: float64
```



pandas groupby

```
df2 = df.groupby(['city', 'furniture']).mean()  
df2
```

		area	rooms	bathroom	parking spaces	hoa (R\$)	rent amount (R\$)	property tax (R\$)	Tire insurance (R\$)	total (R\$)
city	furniture									
Belo Horizonte	furnished	453.824859	2.920904	2.666667	2.220339	793.514124	4756.994350	405.536723	68.045198	6024.372881
	not furnished	167.064755	3.037003	2.358927	1.912118	2574.827012	3485.185014	251.045328	51.322849	6362.868640
Campinas	furnished	104.297297	2.036036	1.927928	1.522523	728.927928	2889.648649	151.981982	38.342342	3808.918919
	not furnished	142.537736	2.402965	1.964960	1.563342	613.962264	2285.699461	147.010782	31.497305	3078.187332
Porto Alegre	furnished	116.081505	2.097179	1.799373	1.210031	584.150470	3026.949843	160.805643	46.639498	3818.605016
	not furnished	99.057208	2.156751	1.699085	0.983982	457.845538	2086.131579	110.596110	32.697941	2687.272311
Rio de Janeiro	furnished	107.000000	2.167488	1.800493	0.706897	1418.583744	4227.091133	308.800493	55.061576	6009.556650
	not furnished	104.735160	2.272146	1.739726	0.757991	953.683105	2864.283105	237.592694	37.819178	4093.387215
São Paulo	furnished	148.548650	2.388575	2.482109	1.835530	1456.456372	5573.575016	442.163214	73.103578	7545.463905
	not furnished	162.739404	2.622031	2.462273	1.893107	1063.219609	4311.199814	515.563577	58.468794	5948.773405



pandas groupby

```
df2 = df.groupby(['city', 'furniture']).count()  
df2
```

		area	rooms	bathroom	parking spaces	floor	animal	hoa (R\$)	rent amount (R\$)	property tax (R\$)	fire insurance (R\$)	total (R\$)
city	furniture											
Belo Horizonte	furnished	177	177	177	177	177	177	177	177	177	177	177
	not furnished	1081	1081	1081	1081	1081	1081	1081	1081	1081	1081	1081
Campinas	furnished	111	111	111	111	111	111	111	111	111	111	111
	not furnished	742	742	742	742	742	742	742	742	742	742	742
Porto Alegre	furnished	319	319	319	319	319	319	319	319	319	319	319
	not furnished	874	874	874	874	874	874	874	874	874	874	874
Rio de Janeiro	furnished	406	406	406	406	406	406	406	406	406	406	406
	not furnished	1095	1095	1095	1095	1095	1095	1095	1095	1095	1095	1095
São Paulo	furnished	1593	1593	1593	1593	1593	1593	1593	1593	1593	1593	1593
	not furnished	4294	4294	4294	4294	4294	4294	4294	4294	4294	4294	4294



pandas pivot_table

```
df2 = df.pivot_table(columns="city")
df2
```

city	Belo Horizonte	Campinas	Porto Alegre	Rio de Janeiro	São Paulo
area	207.411765	137.561547	103.609388	105.347768	158.899439
bathroom	2.402226	1.960141	1.725901	1.756163	2.467641
fire insurance (R\$)	53.675676	32.388042	36.425817	42.483011	62.428911
hoa (R\$)	2324.197138	628.922626	491.618609	1079.432378	1169.627994
parking spaces	1.955485	1.558030	1.044426	0.744171	1.877527
property tax (R\$)	272.782194	147.657679	124.021794	256.853431	495.701716
rent amount (R\$)	3664.127981	2364.290739	2337.699916	3232.904064	4652.793783
rooms	3.020668	2.355217	2.140821	2.243837	2.558859
total (R\$)	6315.242448	3173.276671	2989.782900	4611.684877	6380.831833

 **pandas** pivot_table

```
df2 = df.pivot_table(columns="parking spaces")
df2
```

parking spaces	0	1	2	3	4	5	6	7	8
area	78.985091	82.526446	154.968599	231.515496	334.225602	562.486957	429.588957	435.575758	477
bathroom	1.376072	1.548209	2.592271	3.549587	4.181242	4.486957	4.619632	4.636364	4
fire insurance (R\$)	29.310473	34.401928	61.669082	87.321281	107.215463	117.073913	130.343558	141.787879	133
hoa (R\$)	1324.239657	737.325620	1155.293237	1618.941116	2165.039290	1716.147826	1120.263804	989.000000	353
property tax (R\$)	119.263884	223.933609	327.545894	642.158058	994.358682	1208.952174	1349.981595	1425.575758	1232
rent amount (R\$)	2140.854268	2602.666116	4525.238647	6430.960744	7752.551331	8278.117391	8867.846626	9486.363636	8704
rooms	1.726798	2.077410	2.922705	3.463843	3.750317	3.917391	4.233129	4.333333	4
total (R\$)	3613.884085	3598.479063	6070.043961	8779.504132	11019.461343	11320.678261	11468.515337	12043.333333	10423



Leitura de Dados de um servidor

- As funções podem ser usadas para ler dados hospedados em servidores



Leitura de Dados de um servidor

- Por exemplo, do servidor do IBGE:

The screenshot shows the IBGE API service page with a dark header containing the IBGE logo and social media links. Below the header, there are six service cards arranged in two rows of three:

- Agregados**: An image of a 3D cube made of smaller cubes, with the text "Análise multidimensional ao seu alcance." and version "3.0".
- Calendário**: An image of a calendar page, with the text "Conheça o cronograma de ações e publicações do IBGE" and version "3.0".
- CNAE**: An image of blue gears, with the text "API referente à Classificação Nacional de Atividades Econômicas" and version "2.0".

- Localidades**: An image of a location pin, with the text "Obtenha os dados sobre as divisões administrativas do Brasil" and version "3.0".
- Malhas geográficas**: An image of a 3D map of Brazil, with the text "Malhas com as mais diversas resoluções e formatos para você liberar a sua imaginação" and version "3.0".
- Metadados**: An image of a double-angle bracket symbol (</>), with the text "Metadados associados às pesquisas" and version "2.0".

<https://servicodados.ibge.gov.br/api/docs>



Leitura de Dados de um servidor

- Exemplo de acesso aos dados:

```
censo_nomes = pd.read_json(  
    "https://servicodados.ibge.gov.br/api/v2/censos/nomes/")
```



pandas

Leitura de Dados de um servidor

```
censo_nomes = pd.read_json("https://servicodados.ibge.gov.br/api/v1/censos/nomes/")
```

```
censo_nomes
```

	nome	regiao	freq	rank	sexo
0	MARIA	0	11734129	1	
1	JOSE	0	5754529	2	
2	ANA	0	3089858	3	
3	JOAO	0	2984119	4	
4	ANTONIO	0	2576348	5	
5	FRANCISCO	0	1772197	6	
6	CARLOS	0	1489191	7	
7	PAULO	0	1423262	8	
8	PEDRO	0	1219605	9	

9	LUCAS	0	1127310	10
10	LUIZ	0	1107792	11
11	MARCOS	0	1106165	12
12	LUIS	0	935905	13
13	GABRIEL	0	932449	14
14	RAFAEL	0	821638	15
15	FRANCISCA	0	725642	16
16	DANIEL	0	711338	17
17	MARCELO	0	693215	18
18	BRUNO	0	668217	19
19	EDUARDO	0	632664	20



Leitura de Dados de um servidor

- Exemplo de acesso aos dados:

```
censo_nomes = pd.read_json(  
    "https://servicodados.ibge.gov.br/api/v2/censos/nomes/ran  
    king?qtd=200")
```



Leitura de Dados de um servidor

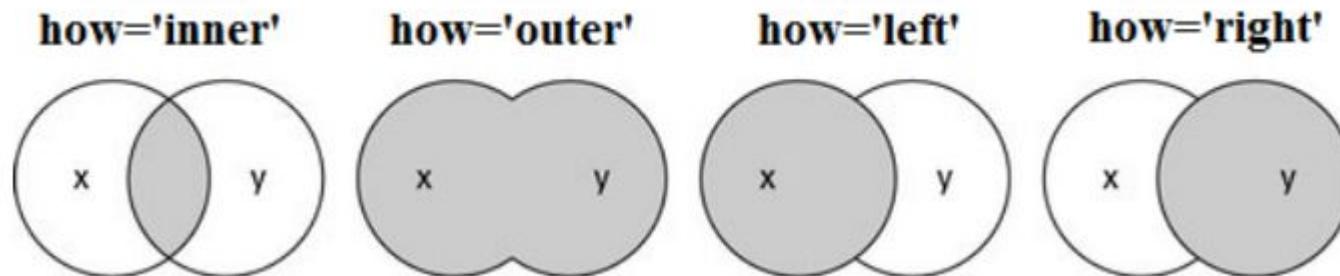
	nome	regiao	freq	rank	sexo
0	MARIA	0	11734129	1	
1	JOSE	0	5754529	2	
2	ANA	0	3089858	3	
3	JOAO	0	2984119	4	
4	ANTONIO	0	2576348	5	
...
195	FABIANO	0	159150	196	
196	MILENA	0	159042	197	
197	WESLEY	0	157205	198	
198	DIOGO	0	156119	199	
199	ADILSON	0	155430	200	

200 rows × 5 columns



pandas Merge

- Merge é usado para unir tabelas
- Tipos possíveis de combinação:





Merge

```
import pandas as pd

tabela1 = pd.DataFrame({
    'Nome': ['João', 'João', 'Pedro', 'Caio'],
    'Telefone': ['11111', '22222', '333333', '4444444'],
    'Carros': ['c1', 'c2', 'c3', 'c4']})
```

tabela1

	Nome	Telefone	Carros
0	João	11111	c1
1	João	22222	c2
2	Pedro	333333	c3
3	Caio	4444444	c4

```
tabela2 = pd.DataFrame({
    'Nome': ['João', 'Marcelo', 'Thiago', 'Caio'],
    'Irmãos': [1, 3, 2, 2]})
```

tabela2

	Nome	Irmãos
0	João	1
1	Marcelo	3
2	Thiago	2
3	Caio	2



Merge

```
df = pd.merge(tabela1, tabela2, how = 'inner', on = 'Nome')
```

```
df
```

	Nome	Telefone	Carros	Irmãos
0	João	11111	c1	1
1	João	22222	c2	1
2	Caio	4444444	c4	2



Merge

```
df = pd.merge(tabela1, tabela2, how = 'outer')
```

```
df
```

	Nome	Telefone	Carros	Irmãos
0	João	11111	c1	1
1	João	22222	c2	1
2	Pedro	333333	c3	NaN
3	Caio	4444444	c4	2
4	Marcelo	NaN	NaN	3
5	Thiago	NaN	NaN	2



pandas Merge

```
df = pd.merge(tabela1, tabela2, how = 'left', on = 'Nome')
```

```
df
```

	Nome	Telefone	Carros	Irmãos
0	João	11111	c1	1
1	João	22222	c2	1
2	Pedro	333333	c3	NaN
3	Caio	4444444	c4	2

```
df = pd.merge(tabela1, tabela2, how = 'right', on = 'Nome')
```

```
df
```

	Nome	Telefone	Carros	Irmãos
0	João	11111	c1	1
1	João	22222	c2	1
2	Marcelo	NaN	NaN	3
3	Thiago	NaN	NaN	2
4	Caio	4444444	c4	2

Pandas + Visualização



Visualização de Dados

- A maioria das ferramentas para visualização de dados trabalham bem com o Pandas
- Os métodos de visualização do Pandas são normalmente construídos com base no Matplotlib
- Para se ter mais liberdade no conteúdo e possibilidades de visualização se recomenda usar o Matplotlib ou o Seaborn ou o Plotly



Visualização de Dados

- Tanto ***Series*** como ***DataFrame*** possuem o método **`.plot()`** que também pode ser encadeado para gerar visualização de diversos tipos, como histograma, área, pizza, dispersão, entre outros.
- Metodos `.hist()`, `.area()`, `.pie()` e `.scatter()`

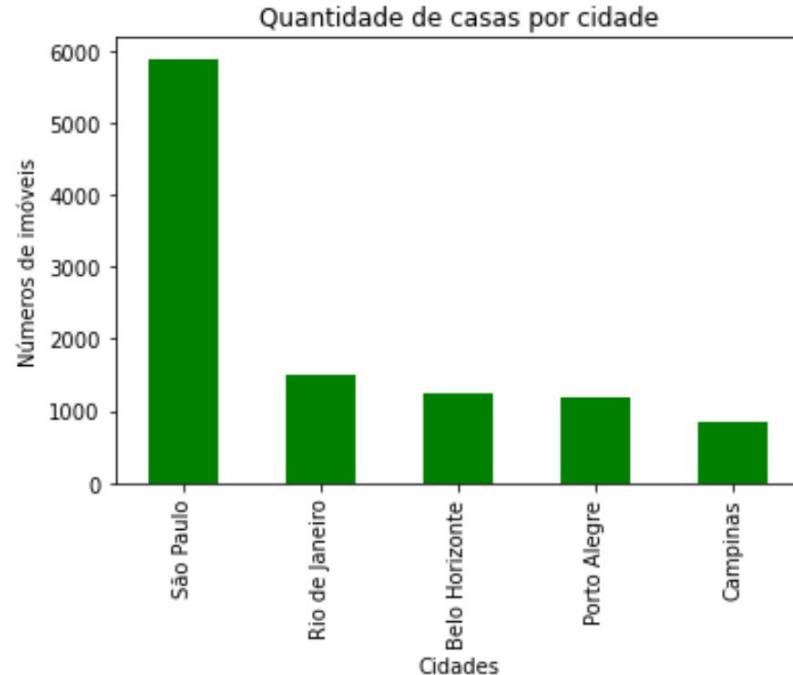


- Utilizando o dataset:

houses_to_rent_v2

```
import matplotlib.pyplot as plt

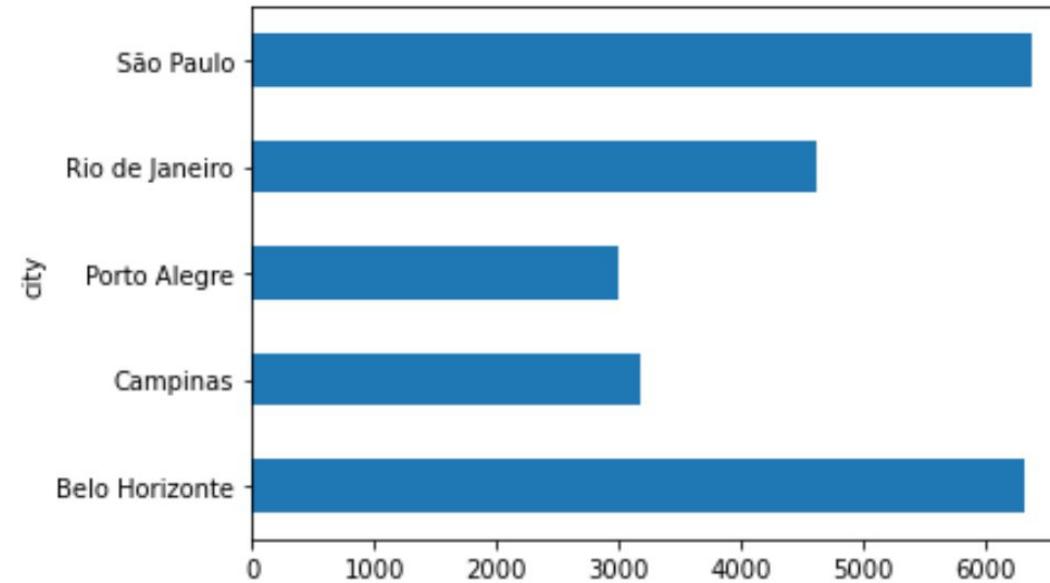
df.city.value_counts().plot(kind='bar', color='green')
plt.title('Quantidade de casas por cidade')
plt.xlabel('Cidades')
plt.ylabel('Números de imóveis')
plt.show()
```





pandas

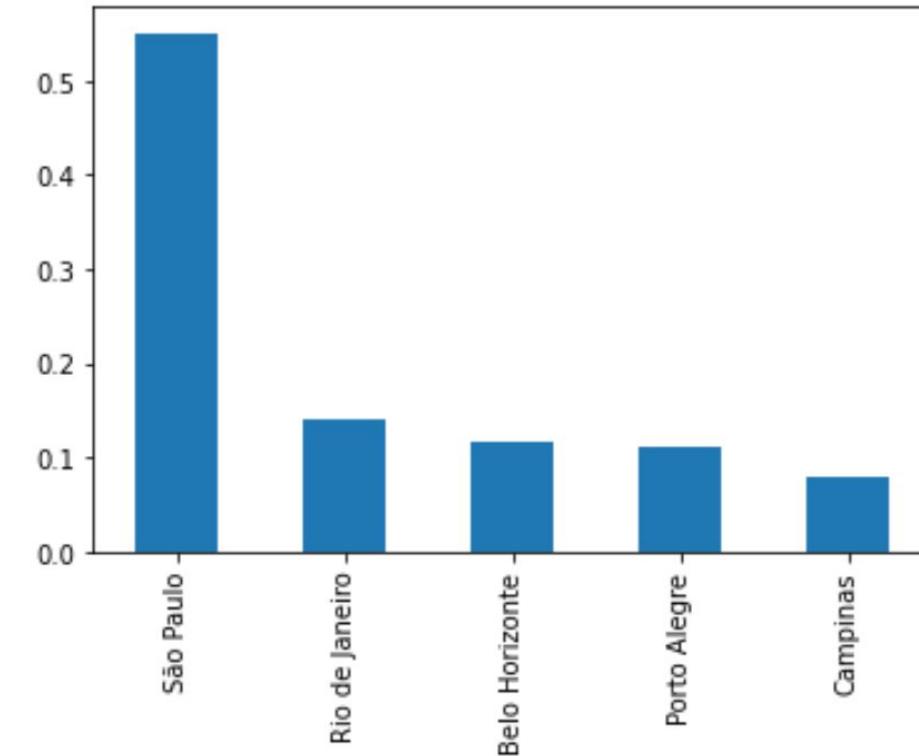
```
df.groupby(['city']).mean()['total (R$)'].plot(kind="barh")  
  
<AxesSubplot:ylabel='city'>
```





```
df.city.value_counts(normalize=True).plot(kind='bar')
```

```
<AxesSubplot:>
```

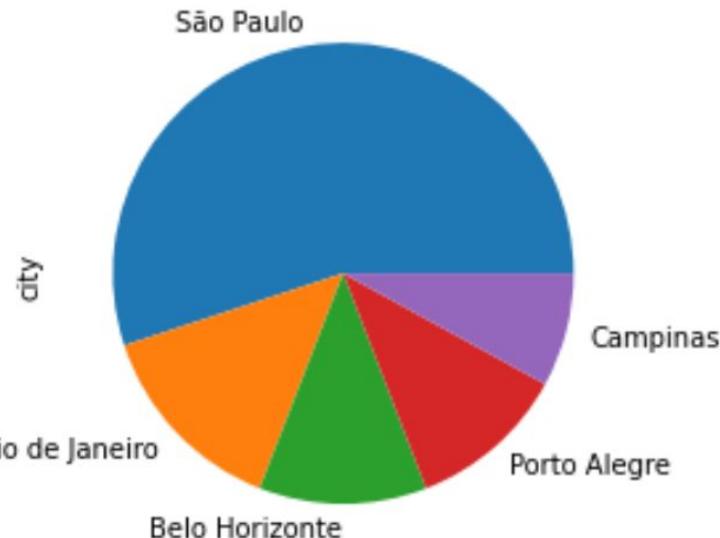




pandas

```
df.city.value_counts(normalize=True).plot.pie()
```

```
<AxesSubplot:ylabel='city'>
```

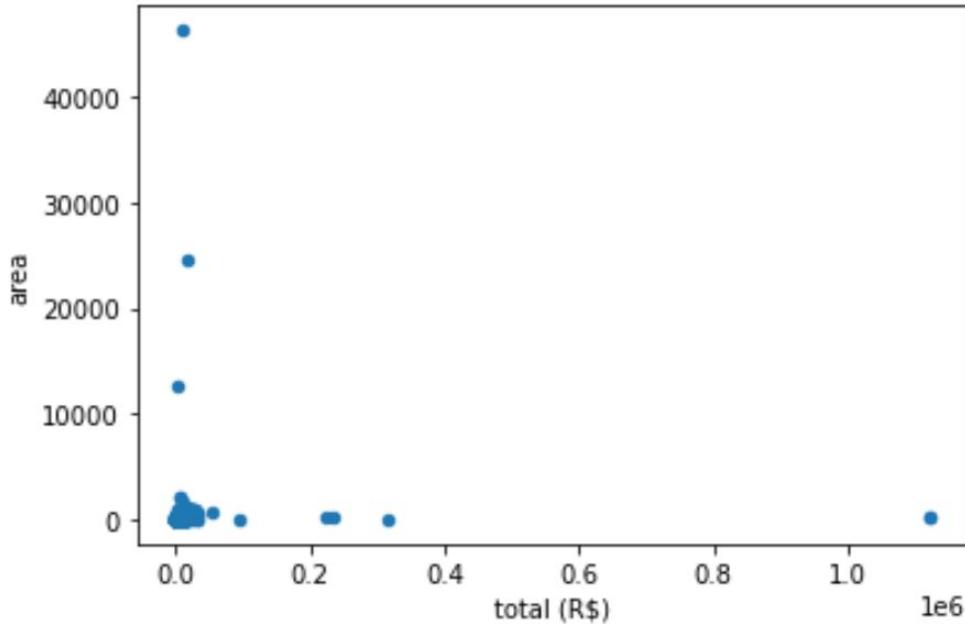




- ***Outliers***: dados que se diferenciam drasticamente de todos os outros

```
df.plot.scatter(x='total (R$)', y='area')
```

```
<AxesSubplot:xlabel='total (R$)', ylabel='area'>
```

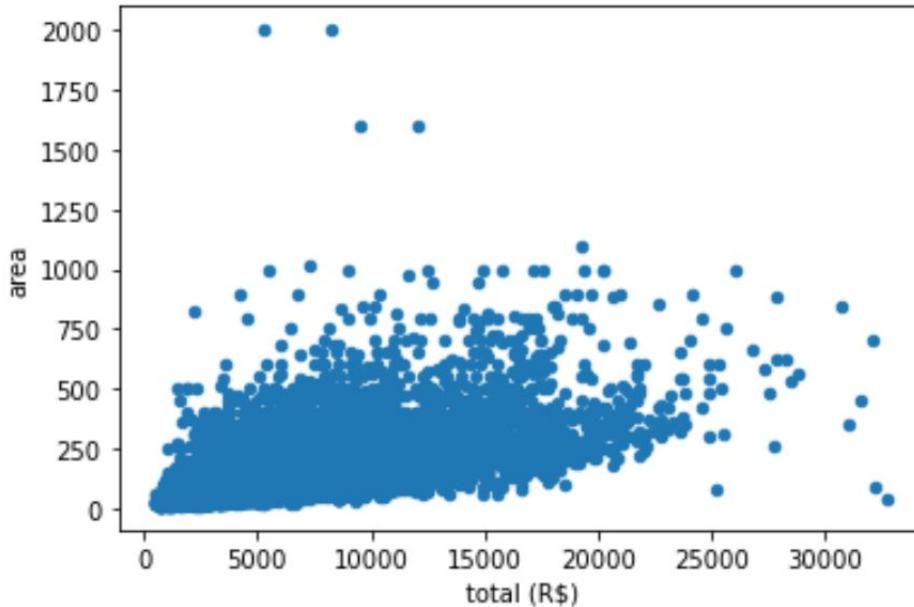


pandas

- Podemos tratar os *outliers*
- Nesse caso, vamos somente não considerá-los

```
df2 = df[df['total (R$)'] <= 50000]
df2 = df2[df2['area'] <= 10000]
```

```
df2.plot.scatter(x='total (R$)', y='area')
<AxesSubplot:xlabel='total (R$)', ylabel='area'>
```





Exercício para Entrega

- Utilize o dataset sintético historico.xlsx para entender o comportamento de alunos reprovados em disciplinas ofertadas em uma escola
- Dentre as análises é obrigatório verificar:
 - Qual é, percentualmente, a taxa de alunos que reprovam em uma disciplina pela 1º vez? E a mesma disciplina pela 2ª vez? 3ª? 4ª? 5ª?...



Exercício para Entrega

- Qual disciplina reprova mais? Qual reprova menos?
- Qual é a relação entre a nota obtida na reprovação e o sucesso/fracasso da próxima vez que o aluno faz a mesma disciplina?
- Faça gráficos e tabelas para compor a solução do problema
- Apresente a sua conclusão sobre a análise realizada