Python - Introdução

História e contextualização

Python é uma linguagem de programação criada por Guido van Rossum em 1991. Os objetivos do projeto da linguagem eram: produtividade e legibilidade. Em outras palavras, Python é uma linguagem que foi criada para produzir código bom e fácil de manter de maneira rápida. Entre as características da linguagem que ressaltam esses objetivos estão:

- baixo uso de caracteres especiais, o que torna a linguagem muito parecida com *pseudo-código executável*;
- o uso de identação para marcar blocos;
- quase nenhum uso de palavras-chave voltadas para a compilação;
- coletor de lixo para gerenciar automaticamente o uso da memória;

Porque Python ?

É fácil ver que a linguagem tem facilidades incríveis para uso geral. A pergunta é: por que Python é a linguagem ideal para aplicações científicas?

As respostas são muitas, mas podemos resumir algumas aqui. A primeira razão, e provavelmente a principal, é: Python é uma linguagem expressiva, em que é fácil traduzir o raciocínio em um algoritmo. Em aplicações científicas, o raciocínio é essencialmente complicado — essa é a natureza das ciências. É um problema adicional para o cientista ter que se preocupar com, além do assunto básico de sua pesquisa, a correção do programa em detalhes pouco relevantes: alocação de memória, gerenciamento de recursos, etc. Python faz isso tudo automaticamente de maneira muito eficiente, permitindo ao cientista se concentrar exclusivamente no problema sendo estudado.

Mais informações

- **Site oficial da linguagem:** http://www.python.org/.
- **Site oficial da comunidade brasileira**: http://www.pythonbrasil.com.br/
- **Perguntas frequentes:** http://www.pythonbrasil.com.br/moin.cgi/PerguntasFrequentes/SobrePython

Anaconda

O Anaconda é uma iniciativa que tem como objetivo agregar todas as ferramentas para análise de dados em um único arquivo.

Resumidamente, é um arquivo que irá instalar em seu computador todas as bibliotecas e recursos necessários para você começar seus projetos de Data Science e Machine Learning, como o Python em si, Jupyter Notebook, a IDE Spyder, além de famosas bibliotecas, como NumPy, Pandas, Scikit-learn, etc

Jupyter - O que é ?

Em 2014, Fernando Pérez, criou um subproduto do projeto IPython que chamou de projeto Jupyter. Jupyter Notebook é um ambiente computacional web, interativo para criação de documentos “Jupyter Notebooks”. O documento é um documento JSON com um esquema e contém uma lista ordenada de células que podem conter código, texto, fórmulas matemáticas, plotagens e imagens. A extensão dos notebooks é “.ipynb”.

Os documentos Jupyter Notebooks podem ser convertidos em outros formatos como HTML, slides, Latex, PDF, Python, etc.

O Jupyter Notebook apresenta uma interface Web construída sobre algumas bibliotecas open-source, como o IPython, 0MQ, Tornado, jQuery, Bootstrap e o MathJax. Ele pode conectar a núcleos de diferentes linguagens de programação. Além do Python, pode conectar-se a linguagens como o **R, Julia, Ruby, Scala e Haskell**. Atualmente, são suportadas mais de 40 linguagens de programação.

Markdown

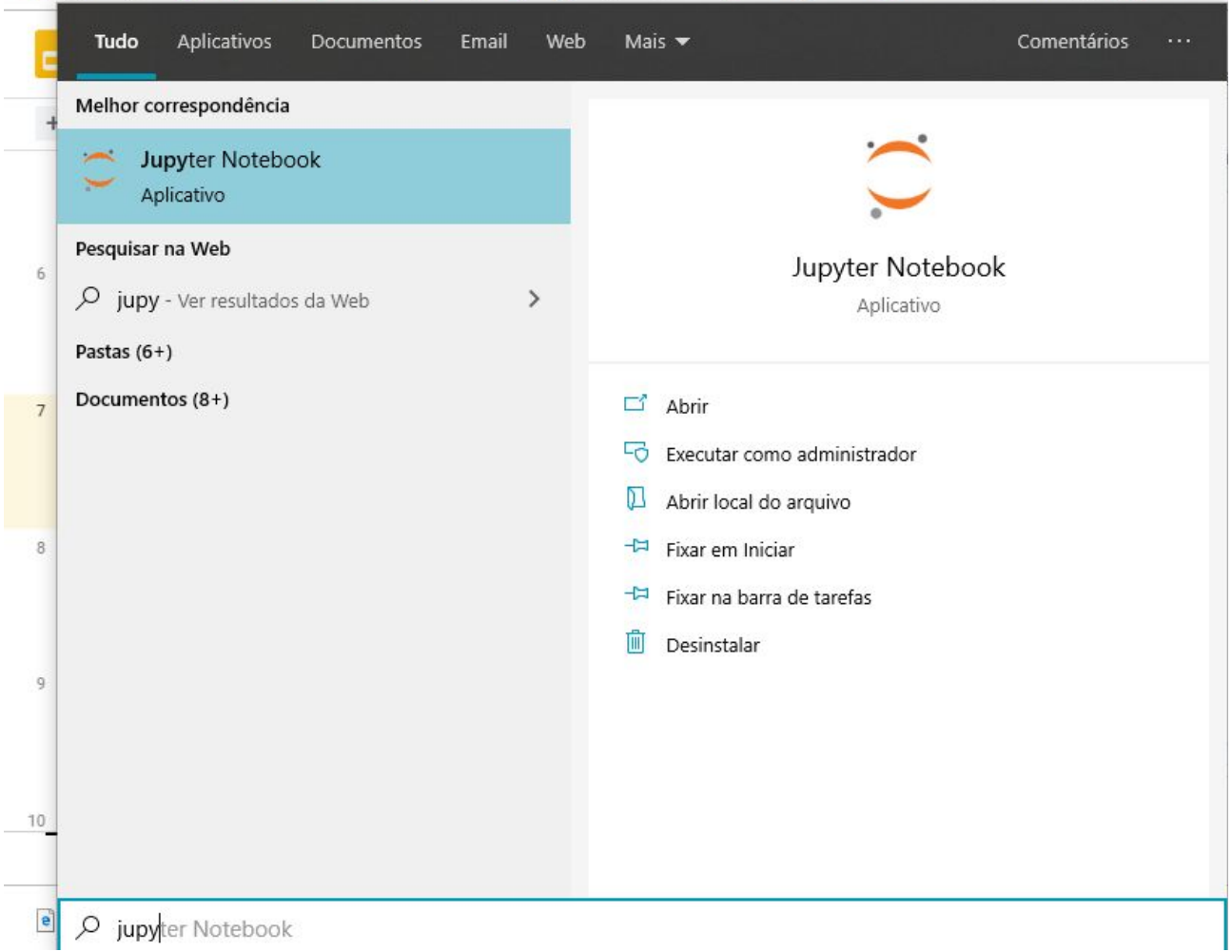
Desenvolvido em 2004 por John Gruber e Aaron Swartz para simplificar a estruturação de um texto, o Markdown é um sistema de formatação aberto que torna a escrita e a leitura mais simples. Com uma codificação mínima, além de fácil, ele é visualmente mais "limpo" e pode ser convertido facilmente para HTML.
Basicamente, ele marca alterações nos textos (subtítulos, negrito, itálico etc) apenas com os símbolos do teclado, sem usar teclas de atalho, menus, selecionando o texto e sem aquele visual complexo - para os que não estão acostumados - de HTML.

A linguagem Markdown pode ser processada em diversos programas

- **Para aprender um pouco mais sobre a sintaxe do markdown:**
  ➔ https://blog.da2k.com.br/2015/02/08/aprenda-markdown/

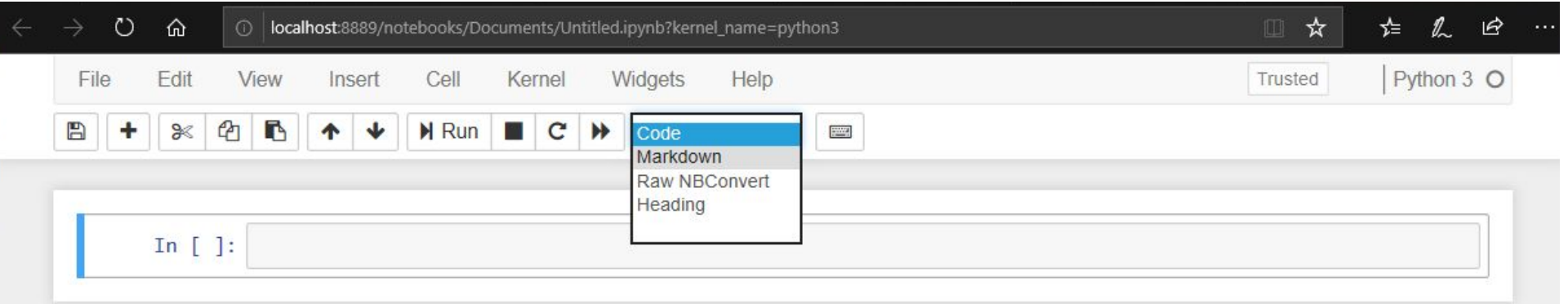
Jupyter Notebook

Jupyter Notebook é um ambiente executado em browser capaz de criar e editar documentos compostos por código, elementos de texto, figuras, gráficos e tabelas. Assim como o nome do aplicativo sugere cada documento pode ser tratado como um “caderno” onde se realiza a análise e visualização de dados, permitindo assim uma exibição do processo por completo, da execução do código ao resultado gráfico.


Tudo
Aplicativos
Documentos
Email
Web
Mais
Comentários
Melhor correspondência
Jupyter Notebook
Aplicativo
Pesquisar na Web
jupy - Ver resultados da Web
Pastas (6+)
Documentos (8+)
Jupyter Notebook
Aplicativo
Abrir
Executar como administrador
Abrir local do arquivo
Fixar em Iniciar
Fixar na barra de tarefas
Desinstalar
jupyter Notebook


localhost:8888/tree/Dropbox/Projeto
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
/ Dropbox / Projeto
..
Dados & Decisões
Notebooks
Notebook:
Python 3
Other:
Text File
Folder
Terminals Unavailable
Create a new notebook with

Código e Texto



Alguns comandos úteis

- **# -** usado para deixar o texto como comentário no código.
- **%%time -** Calcula o tempo que o software demora para rodar aquele bloco de códigos.
- **os.getwcd() -** Mostra o diretório que o software vai buscar e salvar os arquivos. Necessário importar o pacote “os” (import os).

Comandos Básicos

Operadores básicos:

- **+**: Soma
- **-**: Subtração
- ***** : Produto
- **/**: Divisão de ponto flutuante
- **//**: Divisão inteira
- **%**: Resto da divisão inteira
- ******: Potencialização

Operadores Relacionais:

- **==**: Igual a
- **!=**: Diferente de
- **>**: Maior que
- **<**: Menor que
- **>=**: Maior ou igual
- **<=**: Menor ou igual

Operadores Lógicos e especiais:

- and ou & (Bitwise AND)
- or ou | (Bitwise or)
- not ou ~(bitwise not)
- in e not in
- is e is not

Mais operadores em :https://www.programiz.com/python-programming/operators

Curso de introdução ao Python

In [1]:
```
# Contas como em uma calculadora
1+1
```

Out[1]: 2

In [2]:
```
2*3
```

Out[2]: 6

In [3]:
```
2**2
```

Out[3]: 4

In [4]:
```
2-1
```

Out[4]: 1

In [ ]:

Comandos Básicos

Tipos de valores:

- **int**: para inteiro (número inteiro)
- **float**: para ponto flutuante (número decimal)
- **bool**: para booleano (lógico)
- **str**: para string (texto)

In [6]:
```
x=5
type(x)
```

Out[6]:
```
int
```

In [8]:
```
y = True
type(y)
```

Out[8]:
```
bool
```

In [9]:
```
z = 1.55
type(z)
```

Out[9]:
```
float
```

In [10]:
```
k  = "Demografia"
type(k)
```

Out[10]:
```
str
```

Comandos Básicos

Estrutura de dados:

- Vetor
  - **split()**
- Matriz
  - **append()**
- String (Cadeia de Caracteres)
  - **find()**

Estruturas de Controle:

- Estrutura de Sequência
  - **suite**
- Estrutura de Seleção
  - **if,if-else,if-elif,if-elif-else**
- Estrutura de repetição
  - **while** e **for**

O Curso não cobre comandos básicos de lógica de programação, para mais informações procurar material de apoio indicado no final da apresentação.

Comandos Básicos

```
for i in range(100):
    if i % 3 == 0 and i % 5 == 0:
        print('fizzbuzz')
    elif i % 3 == 0:
        print('fizz')
    elif i % 5 == 0:
        print('buzz')
    else:
        print('-')
```

Estrutura de for, if e while no python devem ser indentadas.

Não iremos entrar em detalhes sobre isso no curso mas caso necessitem estes links explicam com mais detalhes:
https://data36.com/python-for-loops-explained-data-science-basics-5/

https://data36.com/python-if-statements-explained-data-science-basics/

Outras possibilidades python

```
In [4]: #### Loop for

        lista = [1,5,7,8]

        resultado =[]
        for item in lista:
            novo_item = item ** 2
            resultado.append(novo_item)

        resultado
```

```
Out[4]: [1, 25, 49, 64]
```

```
In [5]: ## Usando List Comprehension
        result2 = [item ** 2 for item in lista]
        #resultado = [faça_algo_com(item) para cada item na lista]
        result2
```

```
Out[5]: [1, 25, 49, 64]
```

```
In [6]: #### map
        resultado_3 = map(lambda x: x*x, lista)
        print(list(resultado_3))
```

```
[1, 25, 49, 64]
```

Python oferece outras estruturas para lidar com listas e interação que podem ser utilizadas para simplificar o código.

Aqui apresentados **Map e List Comprehension**, o link abaixo oferece mais informações:

https://www.thepythoncorner.com/2017/12/the-art-of-avoiding-nested-code/

Como instalar/Carregar um pacote no Python ?

O Anaconda contém os principais pacotes que auxiliam nas análises de dados e modelagem estatística. Mas, para chamar um pacote precisamos utilizar alguns comandos.

Carregar pacote : Para importar/carregar um pacote utilizamos o comando **import**. Exemplo, para importar o pacote “math” utilizamos o código : **import math**.

Do pacote importar uma função ou item: O código abaixo importará o módulo **sqrt** do pacote **math**. Exemplo : **from** math **import** sqrt.

Para diminuir a digitação, costuma-se importar todas as funções de math dessa forma: **from** math **import** *

Mais informações : http://www.devfuria.com.br/python/imports/

Help de pacotes e funções

- Geralmente cada pacote tem o seu site no qual consta a documentação das funções e aplicações. Geralmente pesquiso no google o **nome do pacote + documentação** e procuro o site oficial. Exemplo : **“Pandas Documentation”** : https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html.

- Para ter acesso ao help de funções básicas do python e definições utilizar o comando: **help(object)**. Exemplo : **help(list), help('print')** etc. Mais exemplos no link : https://www.programiz.com/python-programming/methods/built-in/help.

- Existem alguns casos em que conseguimos acessar a descrição da função com **“?”** na frente do comando. Exemplo : ***pd.DataFrame.describe?*** *que é equivalente a* ***help(pd.DataFrame.describe),*** *não se esqueça de importar o pacote.*

Pacote Pandas

Listas e Array’s

Lista - Uma Lista é uma estrutura que pode conter itens. Em Python, para trabalharmos com listas, nos é fornecido um objeto do tipo *list* e todos elementos que estiverem delimitados por colchetes, será interpretado pela linguagem, como um conjunto de itens pertencentes a uma lista.

Array - Um Array é uma Lista onde cada elemento contido possui um número inteiro não negativo vinculado. O array pode ser acessado através de um índice, permitindo assim a sua manipulação.

Data Frame e Series

Posso estar sendo meio óbvio falando sobre Series e DataFrame para alguém que já está acostumado a usar o Pandas, mas quero deixar claro para aqueles que estão começando a principal diferença entre esses dois tipos de Estrutura de Dados.

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

- Um **DataFrame** é simplesmente um conjunto de Series. Trata-se de uma estrutura de dados de 2 dimensões — colunas e linhas — que transforma os dados em uma bela tabela. Exemplo:

Pandas

- Pandas é a biblioteca para manipulação de dados no Python.
- Implementa a estrutura de *Data Frame* e métodos que atuam sobre ele.
- Contém todas as operações usuais disponíveis em instruções SQL.
- Além destas, dispõe de funções estatísticas e recursos de visualização.
- Em termos de performance computacional, aproxima do `data.table` do R e empata com o `tidyverse`.

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

Operações Típicas

- Importar e/ou acessar dados.
- Ordenar os registros da tabela.
- Selecionar e fatiar nos índices/eixos.
- Filtrar registros por predicado.
- Renomear os índices/eixos.
- Modificar a disposição do conteúdo.
- Modificar/transformar o conteúdo.
- Aplicar funções/calcular medidas resumo.
- Agregar por categorias e aplicar.
- Concatenar tabelas.
- Juntar ou conciliar tabelas.

Leitura de dados em .txt e Excel

O primeiro passo quando trabalhamos com análise de dados é obter os dados, o segundo é usar alguma ferramenta para ler esses dados.

Vamos usar a função **read_csv()**, ela lê e converte dados tabulares(text files) em um DataFrame. Este nada mais é do que uma tabela, assim como a do Excel.

- Para arquivos que estejam no mesmo diretório do notebook, não precisamos passar o endereço inteiro do arquivo, apenas o nome completo.
- **Obs.:** nunca esqueça de colocar a extensão do arquivo, .csv, .xlsx etc.
- Como padrão, a função usa a primeira linha do arquivo como cabeçalho. Caso a primeira coluna não esteja nomeada, ela será usada como Index.
- Se não vamos ler um arquivo com ponto e vírgula, iremos usar o argumento sep para explicitar o separador correto (exemplo : sep=’,’)

Fonte e mais exemplos :
https://www.linkedin.com/pulse/lendo-arquivos-csv-com-pandas-rog%C3%A9rio-guimar%C3%A3es-de-campos-j%C3%BAnior/

In [ ]:
```
!pip install pandas
import pandas as pd
```

In [28]:
```
df = pd.read_csv('iris.csv')
```

In [29]:
```
df.head()
```

Out[29]:

	Unnamed: 0	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa

In [30]:
```
len(df)
```

Out[30]: 150

In [31]:
```
df.describe()
```

Resumo dos dados

Sejá “**df**” nossa base de dados:

#Soma dos valores de um DataFrame
>>> df.sum()
#Menor valor de um DataFrame
>>> df.min()
#Maior valor
>>> df.max()
#Index do menor valor
>>> df.idmin()
#Index do maior valor
>>> df.idmax()
#Resumo estatístico do DataFrame, com quartis, mediana, etc.
>>> df.describe()

#Média dos valores
>>> df.mean()
#Mediana dos valores
>>> df.median()
 # Desvio Padrão
>>> df.std()

Extraindo informações básicas da Base de dados

Para conseguir fazer operações e obter insights sobre a base de dados precisamos conhecer bem a base de dados. Vamos apresentar algumas funções que nos permite ter noções gerais sobre os dados.

- **head() -** cabeçalho
- **shape -** dimensão da base de dados (linhas x colunas)
- **len -** número de linhas da base de dados
- **columns -** nome das colunas
- **describe() -** algumas informações sumarizadas sobre as colunas (quantos valores, valores únicos etc).
- **isnull()-** mostra os missings/ NA

iloc , loc

```
#podemos chamar uma linha pelo seu índice
df.loc[5]


#ou com um array de índices
df.loc[[0,1,2]]
```

Referência :
https://medium.com/data-hackers/uma-introdu%C3%A7%C3%A3o-simples-ao-pandas-1e15eea37fa1

Exemplo loc:

```
#Criar uma classificação das porcentagens em faixas para facilitar interpretação
censo_cor.loc[censo_cor['PORCENTAGEM DE NEGROS']<25,'cor_cat'] = 'a- <25%'
censo_cor.loc[(censo_cor['PORCENTAGEM DE NEGROS']>=25) & ( censo_cor['PORCENTAGEM DE NEGROS']<50) ,'cor_cat'] = 'b - 25%-50%'
censo_cor.loc[(censo_cor['PORCENTAGEM DE NEGROS']>=50 ) & ( censo_cor['PORCENTAGEM DE NEGROS']<75),'cor_cat'] = 'c- 50%-75%'
censo_cor.loc[censo_cor['PORCENTAGEM DE NEGROS']>=75,'cor_cat'] = 'd- >75%'
censo_cor.head()
```

.loc utilizado com operadores lógicos para criar nova coluna baseada em categorias, dispensando o uso do **if.**

Este código adiciona o nome na categoria na coluna **cor_cat** e nas linhas que cumpre os critérios definidos pelos operadores lógicos.

Ordenando valores

 # Ordenando em ordem crescente

df.sort_values()

 # Ordenando em ordem decrescente

df.sort_values(ascending=False)

 # Ordenando por determinada coluna:

df.sort_values(by="Seminário")

#Ordenando por índice\ resentando indíce:

df.sort_index() df.reset_index()

Tabelas

Uma forma útil de sumarizar as informações de uma coluna é em forma de tabelas. Vamos apresentar os comandos para fazer tabelas simples e tabelas cruzadas.

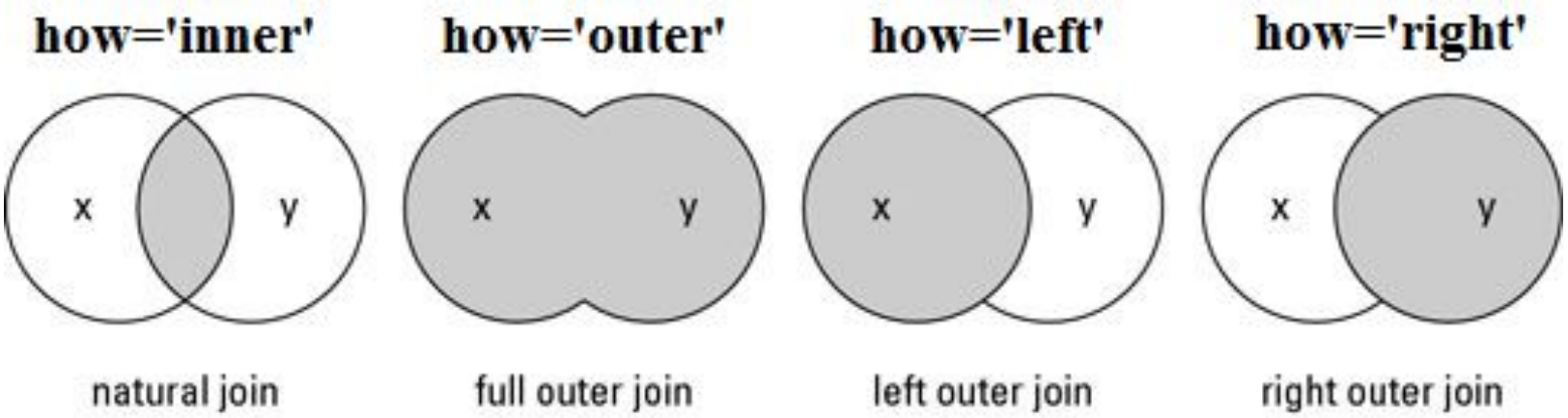
- Para tarefas de contar valores podemos sempre aproveitar de outro método disponível, o **df[‘coluna’].value_counts().**
- Os valores contados também podem ser normalizados para expressar porcentagens: **df["coluna"].value_counts(normalize=True).**
- Para calcular uma tabela entre duas variáveis ou seja, tabelas cruzadas, temos o comando **pd.crosstab(df['Var2'], domicilio_dom['Var1'], margins=True)**

Criando uma coluna nova na base de dados

Para acessar uma coluna do banco de dados basta usar o comando: **df['Nome da Coluna'].**

Caso você queira criar uma nova coluna basta usar o comando : **df['Nova Coluna'] = 0** , em que você pode dar o nome que desejar e atribuir com o sinal de **“=”** um número , vetor, string ou coisa do tipo.

Combinar Tabelas - Join, merge, concat



Para unir tabelas o Pandas tem as funções **join(),merge() e concat()**.

Combinar Tabelas -Join, merge, concat

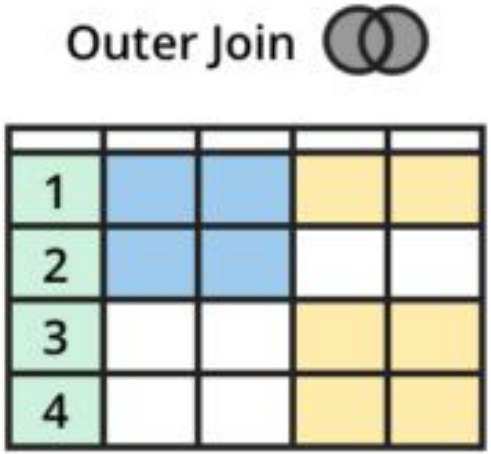
merge() -Combina 2 ou mais dataframes baseado nos valores das colunas em comum, pode se basear nos índices desde que indicado (left_index = True ou right_index= True). Não é necessário que número de linhas ou colunas se igualem.

concat() - Empilha dois ou mais dataframes um abaixo do outro ou um do lado do outro, baseado no eixo selecionado (axis=1 ou axis=0). Para concatenar é necessário que o número de linhas ou de colunas sejam iguais (Dependendo se está concatenando colunas ou linhas)

join()- É uma maneira simplificada de utilizar o merge baseado no índice (left_index=True)

.

Merge- Outer Join




```
censo_merged = pd.merge(domicilio_dom,pessoas_sel,/
                        on='CONTROLE',how="outer")
censo_joined = pessoas_sel.join(domicilio_dom,/
                                how="outer")
pd.concat([domicilio_dom,pessoas_sel], axis=1, /
                                join="outer")
```

Merge- Inner Join

Inner Join


```
censo_joined = pd.merge(domicilio_dom,pessoas_sel/
                        ,on='CONTROLE',how="inner")
```

Left Join


```
censo_joined = pd.merge(domicilio_dom,pessoas_sel/
                        ,on='CONTROLE',how="right")
```

Agregar dados

A função `groupby` Permite realizar as operações `sum()`, `mean()`, `median()`, `min()`, and `max()` agregando conforme as categorias estabelecidas.

```
df2.groupby(by=['Date','Type']).mean()
df4.groupby(level=0).sum()
df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x),
                          'b': np.sum})
```

Agregar dados- Exemplo

A função `groupby` Permite realizar as operações `sum()`, `mean()`, `median()`, `min()`, and `max()` agregando conforme as categorias estabelecidas.

```
# juntando as dummies na nossa base de dados e agregando por residência
censo_joined2 = domicilio_dom.merge(pessoas_cor,on='CONTROLE', how="left")
censo_cor=censo_joined2.groupby('CONTROLE').aggregate(sum)
censo_cor.head()
```

Tipos de dados ausentes

Dados ausentes são uma das maiores dificuldades da etapa exploratória dos dados em um projeto de Data Science. E é extremamente importante identificar os dados ausentes para que isso não vicie sua análise
Para identifica-los, podemos usar os comandos:

- `df.describe()` para retornar um resumo estatístico das variáveis numéricas
- `df.info()` para dar um resumo de valores não-nulos encontrados
- `df.isnull().sum()` para retornar a soma dos valores nulos encontrados

Por muitos motivos pode haver incompletude no dataset, o np.nan é um valor especial definido no **Numpy**, sigla para **Not a Number**, o pandas preenche células sem valores em um DataFrame lido com np.nan.

Excluir dados ausentes

Esta é uma decisão mais radical, e deve ser feita apenas em casos onde não haverá impacto significativo no modelo. Quando o *dataset* é consideravelmente grande e a quantidade de valores ausentes é relativamente insignificante.

Para fazer isso, você vai usar o método `df.dropna()`. Esse método é direto, e remove os valores `NaN` encontrados no *DataFrame*.

Por padrão, se você não informar o eixo, serão eliminadas todas as linhas relativas à celula contendo o valor ausente (`df.dropna(axis=0)`).

Caso você deseje eliminar uma coluna inteira onde existam `NaN`, você deve informar explicitamente com `df.dropna(axis=1)`

Gráficos

Referência :
https://medium.com/data-hackers/uma-introdu%C3%A7%C3%A3o-simples-ao-pandas-1e15eea37fa1

Matplotlib e Seaborn

- Matplotlib tem os recursos básicos para gráficos no Python.
- Seaborn é feita sobre a Matplotlib e constrói gráficos estatísticos.
- Objetos Pandas possuem alguns métodos gráficos.

Gráficos

Sempre que precisar fazer um gráfico pesquise a documentação na internet. Lá você encontra todos os argumentos que a função tem. Assim, você pode analisar melhor o que pode ser feito com o seu gráfico.

sns.boxplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, width=0.8, dodge=True, fliersize=5, linewidth=None, whis=1.5, notch=False, ax=None, **kwargs)

Fonte :
https://www.google.com/search?q=sns.boxplot&oq=sns.boxplot&aqs=chrome..69i57j69i59l2.6481j0j7&sourceid=chrome&ie=UTF-8

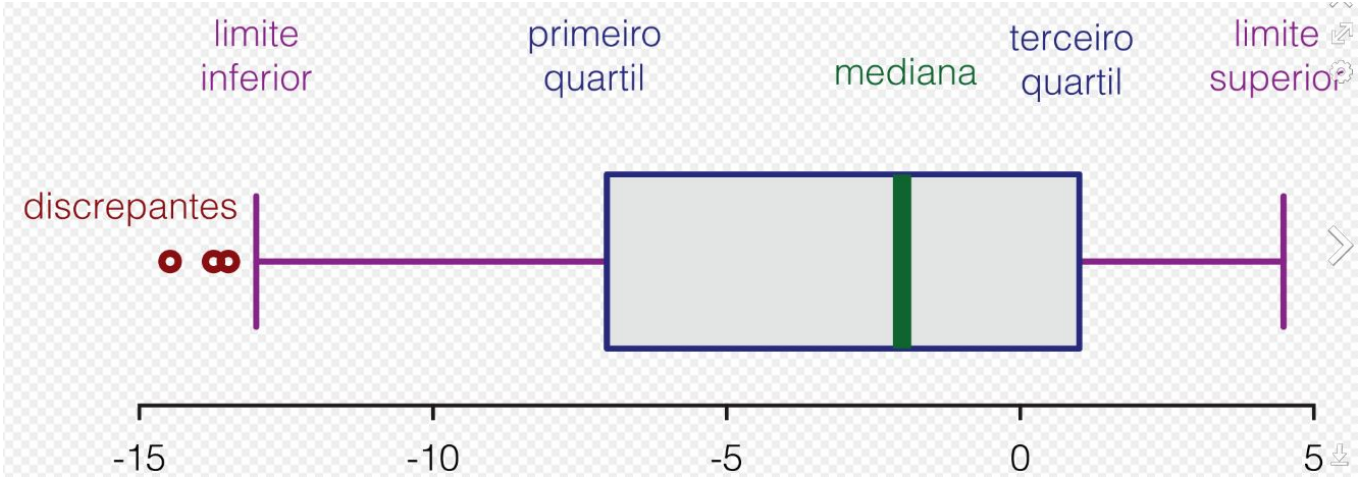
Boxplot

Na estatística, o Boxplot, ou diagrama de caixa, é uma maneira gráfica de representar a alteração dos dados de uma variável por meio de quartis.

O Box Plot fornece informação sobre as seguintes características do conjunto de dados: localização, dispersão, assimetria, comprimento da cauda e outliers (medidas discrepantes).

Em um boxplot são apresentados 5 estatísticas: o mínimo, o primeiro quartil (Q1), a mediana, o terceiro quartil (Q3) e o máximo.

Comando : **sns.boxplot(x)**



Histograma

Um **histograma** é um gráfico de frequência que tem como objetivo ilustrar como uma determinada amostra ou população de dados está distribuída.A altura de cada retângulo representa a quantidade ou a frequência absoluta com que o valor da classe ocorre no conjunto de dados.

Comando: **sns.distplot(x)**

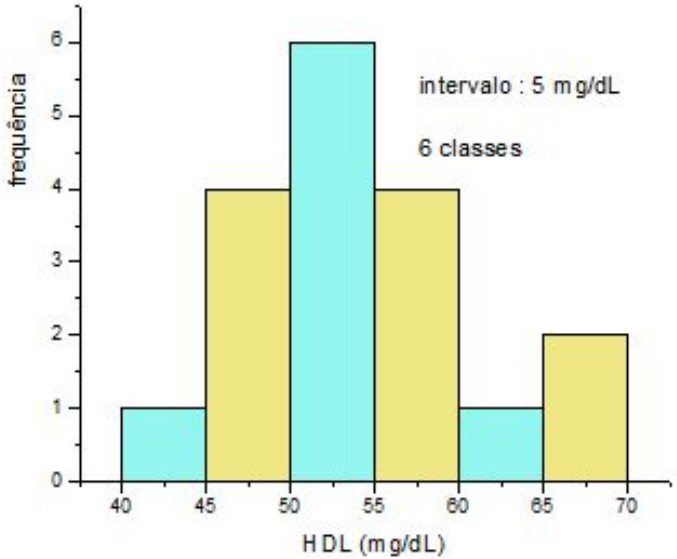


Gráfico de dispersão

Os **diagramas de dispersão** ou **gráficos de dispersão** são representações de dados de duas (tipicamente) ou mais variáveis que são organizadas em um gráfico com a intenção de exibir quanto uma variável é afetada por outra. O gráfico de dispersão utiliza coordenadas cartesianas para exibir valores de um conjunto de dados.Os dados são exibidos como uma coleção de pontos.

Comando: **sns.scatterplot()**

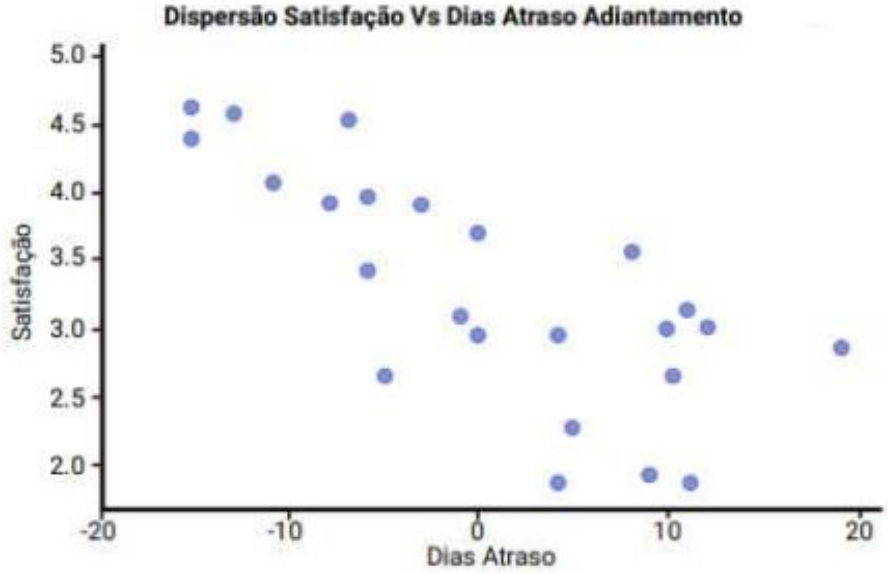


Gráfico de barras

O **gráfico de barras** é um gráfico com barras retangulares e comprimento proporcional aos valores que ele representa. As barras podem ser desenhadas verticalmente ou horizontalmente.

Comando: **sns.barplot()**

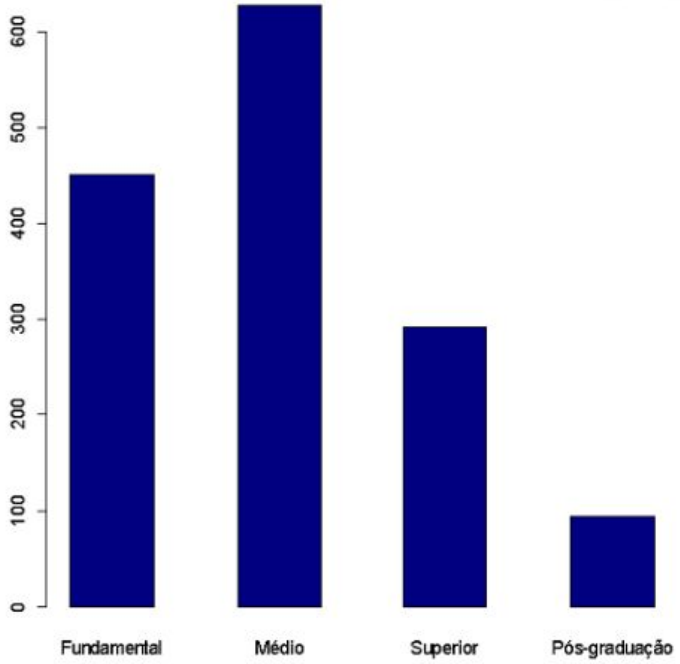


Gráfico de Pizza ou setores

O gráfico de pizza, também conhecido como gráfico de setores ou gráfico circular é um diagrama circular onde os valores de cada categoria estatística representada são proporcionais às respectivas frequências. Este gráfico pode vir acompanhado de porcentagens.

Comando:

```
pie_values = domicilio_dom['UF'].value_counts()/
.rename_axis('estado').reset_index(name="valor")
plt.pie(pie_values["valor"],labels = pie_values["estado"]/
        ,autopct='%1.1f%%', shadow=True, startangle=140)
plt.axis('equal')
plt.show()
```

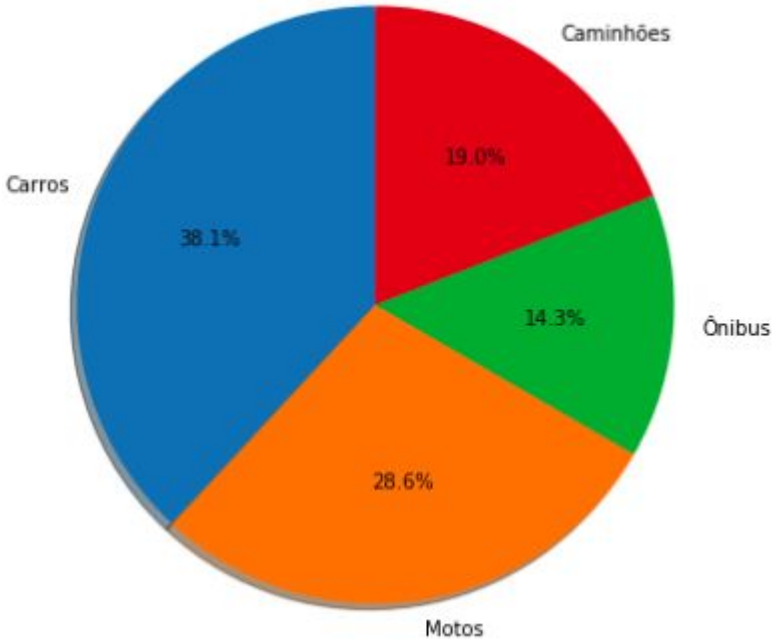


Gráfico de correlação

Apresenta a Correlação de Pearson entre as variáveis escolhidas. O coeficiente,assume apenas valores entre -1 e 1.

Comando : **sns.heatmap()**

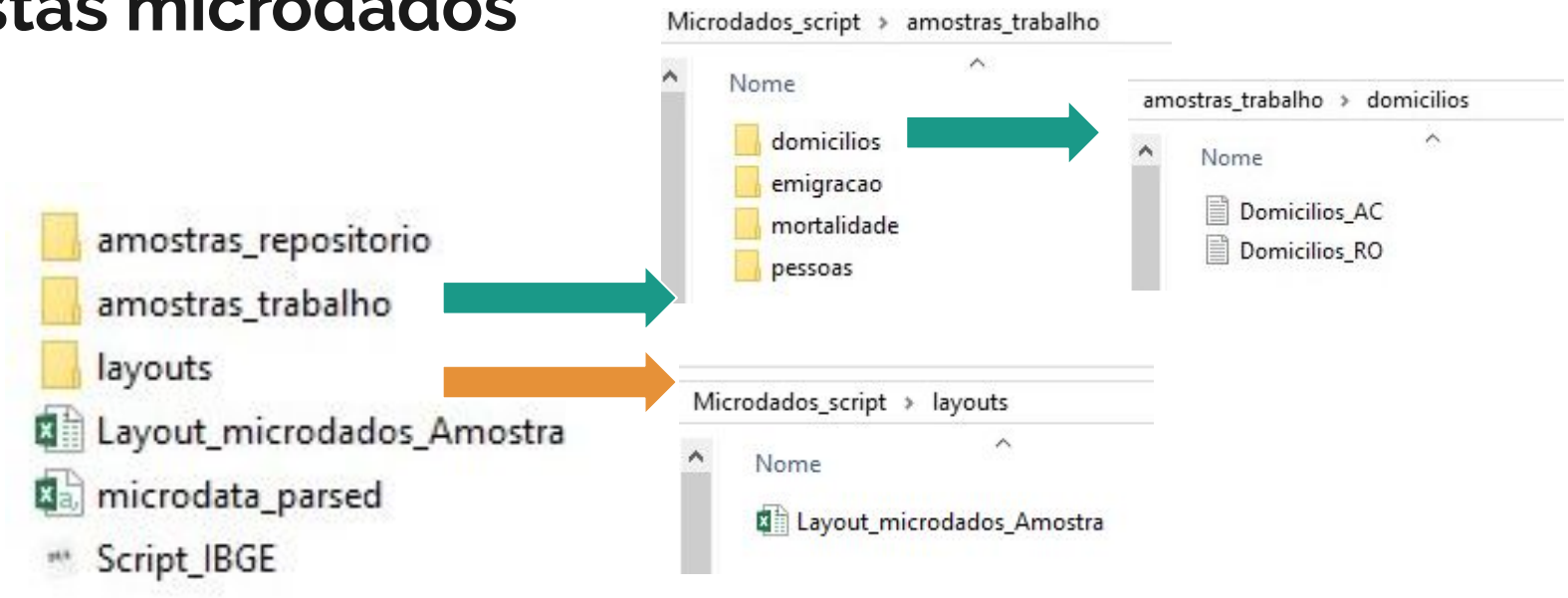
Microdados

Microdados

Os Microdados são bancos de dados em que os registros ou casos (isto é, as linhas) representam as unidades de coleta mais desagregada. No caso dos Censos demográficos, a unidade de coleta é o indivíduo: um questionário é aplicado a cada pessoa residente no Brasil. Assim, um banco de microdados de um Censo é aquele em que cada linha é um indivíduo e cada coluna traz características específicas aplicáveis àquela unidade de análise.

As bases de microdados podem ser lidas por *software* específicos, como R, SAS e SPSS, que permitem ao usuário a manipulação para a composição de novas tabelas e, consequentemente, novas análises, tudo de maneira mais rápida.

Pastas microdados



Abrir uma base de dados do IBGE

Para baixar os dados do censo IBGE, disponibilizamos um tutorial. Para carregar os dados em Python, como vimos nos comandos acima, devemos utilizar uma função que leia o arquivo na sua extensão (.txt, .xlsx).

Para manipular a base de dados utilize as funções aprendidas. Vamos agora fazer alguns exemplos no notebook que esta no formato jupyter notebook.

Teste T

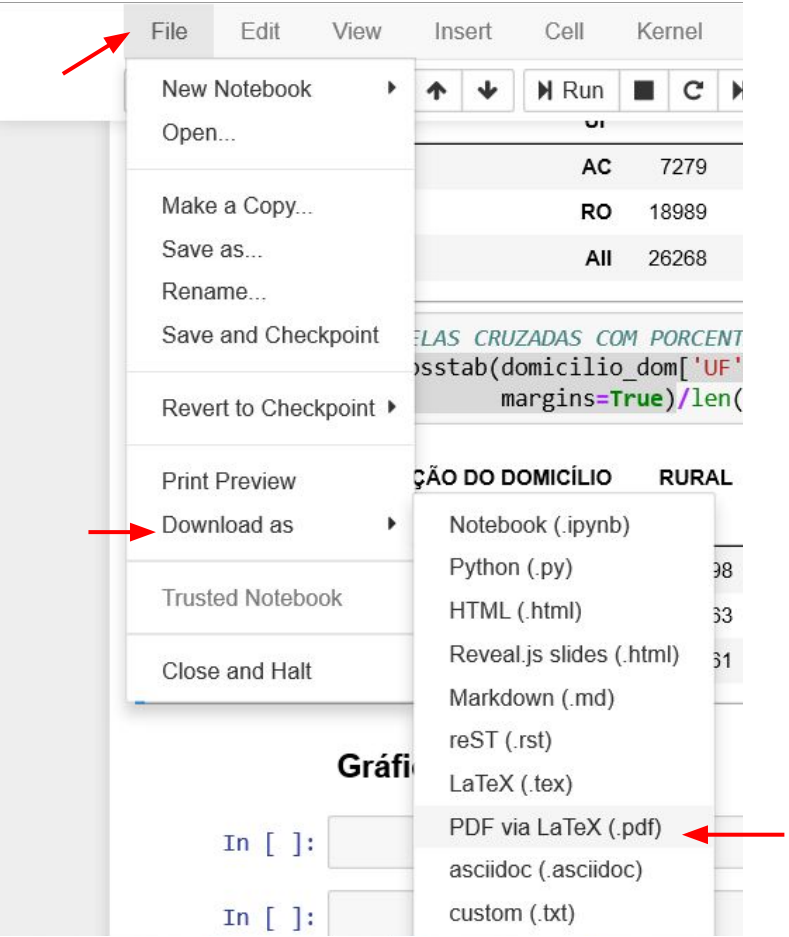
Teste T

Geralmente o teste T é utilizado para calcular se as médias de duas amostras são independentes.

Este é um teste bilateral para a hipótese nula de que 2 amostras independentes têm valores médios (esperados) idênticos. Este teste pressupõe que as populações tenham variações idênticas por padrão.

Existem vários pacotes com o teste T, nesse curso vamos usar o teste do pacote **scipy,** o comando para o teste é : **stats.ttest_ind(a,b)**.

Exportar notebook em PDF



Referências e Material de suporte

Cursos Online

- https://www.udemy.com/course/curso-de-programacao-em-python-do-basico-ao-avancado/
- https://www.udemy.com/course/machine-learning-e-data-science-com-python-y/
- https://www.coursera.org/specializations/python
- https://www.coursera.org/specializations/data-science-python
- https://www.datacamp.com/courses/intro-to-python-for-data-science
- https://www.datacamp.com/courses/pandas-foundations
- https://www.datascienceacademy.com.br/path-player?courseid=python-fundamentos&unit=5aef84565e4cde94cc8b4578Unit

Folhas de cola

- https://i.pinimg.com/originals/4a/42/07/4a42071973b68d100ea5e5a9e7a13a1a.png
- https://s3.studylib.net/store/data/025268801_1-1bb4205c74b96358224e9e1be6dbfbda.png
- https://datacamp-community-prod.s3.amazonaws.com/dbed353d-2757-4617-8206-8767ab379ab3
- https://sinxloud.com/python-cheat-sheet-beginner-advanced/

Referências

- http://pyscience-brasil.wikidot.com/python:python-oq-e-pq
- https://rpubs.com/gomes555/comandos_python
- **Livro :** An Introduction to Statistics with Python: With Applications in the Life Sciences, Tomas Haslwanter Springer.
- **Livro:** Introduction to Data Science with Python: Basics of Numpy and Pandas, Mark Smart.
- **Livro:** Python Para Análise de Dados: Tratamento de Dados com Pandas, NumPy e IPython, Wes McKinney.