LINGUAGEM DE PROGRAMAÇÃO

Linguagem de programação: Introdução a Biblioteca Pandas

Prof. Me. Wesley Viana

- Unidade de Ensino: 04
- Competência da Unidade: Linguagem de programação: Introdução a Biblioteca Pandas
- Resumo: Saber utilizar manipulação de dados panda na linguagem Python.
- Palavras-chave: Algoritmos; Python; Panda; Manipulação de Dados; Orientação a Objeto.
- Título da Teleaula: Linguagem de programação: Introdução a Biblioteca Pandas
- Teleaula nº: 04

Contextualização

- Linguagem de programação: Introdução a Biblioteca Pandas
- Introdução a Manipulação de Dados em Pandas
- Visualização de Dados em Python

Linguagem de programação:

Introdução a Biblioteca

Pandas

Introdução a Biblioteca Pandas

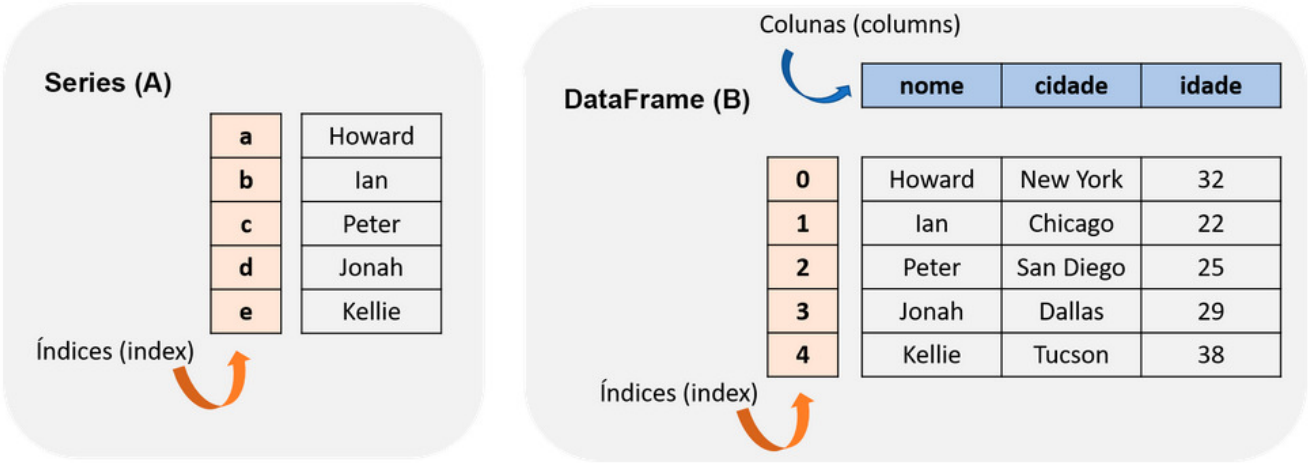
Pandas é um pacote Python que fornece estruturas de dados projetadas para facilitar o trabalho com dados estruturados (tabelas) e de séries temporais. Para utilizar a biblioteca pandas é preciso fazer a instalação.

Pandas possui duas estruturas de dados que são as principais para a análise/manipulação de dados: a **Series** e o **DataFrame**. Uma **Series** é um como um vetor de dados (unidimencional), capaz de armazenar diferentes tipos de dados. Um **DataFrame** é conjunto de Series, ou como a documentação apresenta, um contêiner para Series.

Ambas estruturas, possuem como grande característica, a indexação das linhas, ou seja, cada linha possui um rótulo (nome) que o identifica, o qual pode ser uma string, uma inteiro, um decimal ou uma data.

Introdução a Biblioteca Pandas

A Figura ilustra uma Series (A) e um DataFrame (B). Veja que uma Series possui somente "uma coluna" de informação e seus rótulos (índices). Um DataFrame pode ter uma ou mais colunas e além dos índices, também há um rótulo de identificação com o nome da coluna. Podemos comparar um DataFrame como uma planilha eletrônico, como o Excel (da Microsoft) ou o Calc (do Open Office).



Introdução a Biblioteca Pandas

Vamos importar a biblioteca antes de começar nossa primeira linha de código. Por convenção, a biblioteca é importada com o apelido (as) pd. Logo, para utilizar as funcionalidades, vamos utilizar a sintaxe pd.funcionalidade.

```
import pandas as pd
```

Series: Precisamos utilizar o método Series() do pacote pandas. O método possui o seguinte construtor: pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False).

Dentre todos os parâmetros esperados, somente um é obrigatório para se criar uma Series com dados (se for uma Series sem dados, nenhum parâmetro é obrigatório), o parâmetro data=XXXX.

Introdução a Biblioteca Pandas

Criamos uma Series com um único valor, veja que aparece 0 como índice e 5 como valor. Quando não deixamos explícito os rótulos (índices) que queremos usar é construído um range de 0 até N-1, onde N é a quantidade de valores.

Criamos uma Series a partir de uma lista de nomes, veja que agora os índices variam de 0 até 4 e o dtype é "object".

Criamos uma Series a partir de um dicionário, a grande diferença desse tipo de dado na construção é que a chave do dicionário é usada como índice.

```
pd.Series(data=5) # Cria uma Series com o valor a


lista_nomes = 'Howard Ian Peter Jonah Kellie'.split()

pd.Series(lista_nomes) # Cria uma Series com uma lista de
nomes


dados = {

    'nome1': 'Howard',

    'nome2': 'Ian',

    'nome3': 'Peter',

    'nome4': 'Jonah',

    'nome5': 'Kellie',

}

pd.Series(dados) # Cria uma Series com um dicionário
```

Introdução a Biblioteca Pandas

Criamos uma série contando números e um valor nulo (None). As informações extraídas das linhas 3 a 7, são mais com relação a "forma" dos dados, portanto poderiam ser usadas independente do tipo de dado armazenado na Series, inclusive em um cenário de dados com diferentes tipos.

```
series_dados = pd.Series([10.2, -1, None, 15, 23.4])


print('Quantidade de linhas = ', series_dados.shape) # Retorna uma tupla com o número de linhas

print('Tipo de dados', series_dados.dtypes) # Retorna o tipo de dados, se for misto será object

print('Os valores são únicos?', series_dados.is_unique) # Verifica se os valores são únicos (sem duplicações)

print('Existem valores nulos?', series_dados.hasnans) # Verifica se existem valores nulos

print('Quantos valores existem?', series_dados.count()) # Conta quantas valores existem (excluí os nulos)
```

Introdução a Biblioteca Pandas

Já as informações das linhas 9 a 15, como se tratam de funções matemáticas e estatísticas, podem fazer mais sentido quando utilizadas para tipos numéricos.

print('Qual o menor valor?', series_dados.min()) # Extrai o menor valor da Series (nesse caso os dados precisam ser do mesmo tipo)

print('Qual o maior valor?', series_dados.max()) # Extrai o valor máximo, com a mesma condição do mínimo

print('Qual a média aritmética?', series_dados.mean()) # Extrai a média aritmética de uma Series numérica

print('Qual o desvio padrão?', series_dados.std()) # Extrai o desvio padrão de uma Series numérica

print('Qual a mediana?', series_dados.median()) # Extrai a mediana de uma Series numérica

print('\nResumo:\n', series_dados.describe()) # Exibe um resumo sobre os dados na Series

Introdução a Biblioteca Pandas

DataFrame

Utilizar o método DataFrame() do pacote pandas. O método possui o seguinte construtor: pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False). Veja que todos os parâmetros possuem valores padrões (default) o que permite instanciar um objeto de diferentes formas.

Dentre todos os parâmetros esperados, somente um é obrigatório para se criar um DataFrame com dados, o parâmetro data=XXXX. Esse parâmetro pode receber, um objeto iterável, como uma lista, tupla, um dicionário ou um DataFrame.

Introdução a Biblioteca Pandas

Construtor DataFrame com lista

Invocando o método DataFrame e passando como parâmetro a lista de nomes e um nome (rótulo) para a coluna. Essa construção é para criar um DataFrame, no qual cada lista passe a ser uma coluna.

```
pd.DataFrame(lista_nomes, columns=['nome'])
```

Introdução a Biblioteca Pandas

Construtor DataFrame com dicionário

DataFrames também podem ser construídos a partir de estruturas de dados do tipo dicionário. Cada chave será uma coluna e pode ter atribuída uma lista de valores. Obs: cada chave deve estar associada a uma lista de mesmo tamanho.

Extraindo informações de um DataFrame

Cada objeto possui seus próprios atributos e métodos, logo, embora Series e DataFrame tenham recursos em comum, eles também possuem suas particularidades. No DataFrame temos o método info() que mostra quantas linhas e colunas existem.

Também exibe o tipo de cada coluna e quanto valores não nulos existem ali. Esse método também retorna uma informação sobre a quantidade de memória RAM essa estrutura está ocupando.

Introdução a Biblioteca Pandas

Seleção de colunas em um DataFrame

Podemos realizar operações em colunas específicas de um DataFrame ou ainda criar um novo objeto contendo somente as colunas que serão usadas em uma determinada análise. Para selecionar uma coluna, as duas possíveis sintaxes são:

```
nome_df.nome_coluna
nome_df[nome_coluna]
```

Introdução a Biblioteca Pandas

A primeira forma é familiar aos desenvolvedores que utilizar a linguagem SQL, porém ela não aceita colunas com espaços entre as palavras.

Já a segunda aceita. Se precisarmos selecionar mais do que uma coluna, então precisamos passar uma lista, da seguinte forma: nome_df[['col1', 'col2', 'col3']], se preferir a lista pode ser criada fora da seção e passada como parâmetro.

Através da seleção de certas colunas podemos extrair informações específicas e até compará-las com outras colunas ou com outros dados. Esse recurso é muito utilizado por quem trabalha na área de dados.

Introdução a Biblioteca Pandas

Um dos grandes recursos da biblioteca pandas é sua capacidade de fazer leitura de dados estruturados, através de seus métodos, guardando em um DataFrame. A biblioteca possui uma série de métodos "read", cuja sintaxe é: pandas.read_XXXXX() onde a sequência de X representa as diversas opções disponíveis.

Vamos utilizar o método read_html() para capturar os dados e carregar em um DataFrame. Observe o código, o método read_html capturou todas as tabelas no endereço passado como parâmetro, sendo que cada tabela é armazenada em um DataFrame e o método retorna uma lista com todos eles.

Veja na linha 4, que ao imprimirmos o tipo do resultado guardado na variável dfs, obtemos uma lista e ao verificarmos quantos DataFrames foram criados (len(dfs)), somente uma tabela foi encontrada, pois o tamanho da lista é 1.

```
url =
'https://www.fdic.gov/bank/individual/failed/banklist.
html'

dfs = pd.read_html(url)


print(type(dfs))

print(len(dfs))
```

Sabendo que o tamanho da lista resultado do método é 1, então para obter a tabela que queremos, basta acessar a posição 0 da lista.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Métodos para leitura e escrita da biblioteca pandas

A biblioteca pandas foi desenvolvida para trabalhar com dados estruturados, ou seja, dados dispostos em linhas e colunas. Os dados podem estar gravados em arquivos, em páginas web, em APIs, em outros softwares, em object stores (sistemas de armazenamento em cloud) ou em bancos de dados. Para todas essas origens (e até mais), a biblioteca possui métodos capazes de fazer a leitura dos dados e carregar em um DataFrame.

Dentre todos os possíveis métodos para leitura, nessa aula vamos estudar o read_json, o read_csv e a função read_sql, que contempla a função read_sql_query.

JSON (JavaScript Object Notation - Notação de Objetos JavaScript) é uma formatação leve de troca de dados e independente de linguagem de programação.

CSV (comma-separated values - valores separados por vírgula) é um formato de arquivo, nos quais os dados são separados por um delimitador.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Leitura de JSON e CSV com pandas

Fazer a importação da biblioteca, só precisamos importar uma única vez no notebook ou no script .py.

```
import pandas as pd
```

A leitura de um arquivo JSON deve ser feita com o método: pandas.read_json(path_or_buf=None, orient=None, typ='frame', dtype=None, convert_axes=None, convert_dates=True, keep_default_dates=True, numpy=False, precise_float=False, date_unit=None, encoding=None, lines=False, chunksize=None, compression='infer').

O único parâmetro que é obrigatório para se carregar os dados é o "path_or_buf", no qual deve ser passado um caminho para o arquivo ou um "arquivo como objeto" que é um arquivo lido com a função open(), por exemplo.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Estamos usando o método read_json para carregar dados de uma API. Veja que estamos passando o caminho para o método. Nessa fonte de dados, são encontradas a taxa selic de cada dia.

```
pd.read_json("https://api.bcb.gov.br/dados/serie/bcdata.sgs.11/dados?formato=json").head()
```

Para realizar o carregamento os dados, é necessário incluir o caminho(diretório), portanto o parâmetro "filepath_or_buffer" é obrigatório. Outro parâmetro que é importante para a leitura desse arquivo é o sep ou delimiter (ambos fazem a mesma coisa), veja que sep, por padrão possui o valor ',', ou seja, caso não seja especificado nenhum valor, então o método fará a leitura dos dados considerando que estão separados por vírgula.

O parâmetro header, tem como valor padrão 'infer', que significa que o método realiza a inferência para os nomes das colunas a partir da primeira linha de dados do arquivo.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Manipulação de dados com pandas

Além de vários métodos para carregar e salvar os dados, a biblioteca pandas possui uma diversidade de métodos para a transformação dos dados e a extração de informação para áreas de negócio.

O trabalho com dados: capturar os dados em suas origens, fazer transformações nos dados a fim de padronizá-los, aplicar técnicas estatísticas clássicas ou algoritmos de machine/deep learning feito por engenheiros e cientistas de dados.

Cada profissional atuando em uma parte específica, dependendo da organização da empresa. Em todo esse trabalho é comum fazer a divisão em duas etapas: (i) captura e transformação/padronização dos dados, (ii) extração de informações.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Etapa de captura e transformação/padronização dos dados

A extração dos dados pode ser realizada por meio do método read_json() e guardando em um DataFrame (DF) pandas. Ao carregar os dados em um DF, podemos visualizar quantas linhas e colunas, bem como, os tipos de dados em cada coluna, com o método info().

Remover linhas duplicadas

Para o carregamento de uma base de dados, um dos primeiros tratamentos que devemos fazer é remover os dados duplicados. Certamente, qual registro remover depende da área de negócio e do problema a ser resolvido.

Por exemplo, queremos manter o registro da compra atual, ou queremos manter a primeira compra. Um DataFrame da bilioteca pandas possui o método meu_df.drop_duplicates() que permite fazer essa remoção de dados duplicados.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Criar novas colunas

A segunda transformação que veremos é como criar uma nova coluna. A sintaxe é similar a criar uma nova chave em um dicionário: meu_df['nova_coluna'] = dado. Uma coluna que adiciona a data de extração das informações.

```
from datetime import date
from datetime import datetime as dt

data_extracao = date.today()

df_selic['data_extracao'] = data_extracao
df_selic['responsavel'] = "Autora"

print(df_selic.info())
df_selic.head()
```

Guardamos a data em uma nova coluna, a biblioteca pandas "entende" que esse valor deve ser colocado em todas as linhas. Na linha 7, estamos criando uma nova coluna com o nome do responsável pela extração.

Temos um problema com o tipo de dados das datas, embora cada valor seja do tipo "date", veja que pelo info() ainda obtemos uma coluna object, para que de fato, a biblioteca interprete como um tipo data, vamos ter que utilizar o método da própria biblioteca para fazer a conversão.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Método to_datetime() e astype()

Trabalhar com o tipo "data" pode trazer vantagens, como por exemplo, ordenar da data mais recente para mais antiga. Vamos utilizar os métodos pandas.to_datime() e minha_series.astype() para fazer a conversão e transformar as colunas data e data_extracao.

Nessa conversão usamos o método astype, que transforma os dados de uma coluna (que é uma Series) em um determinado tipo, nesse caso, o tipo datetime especificado. Com astype() podemos padronizar valores das colunas, por exemplo, transformando todos em float, ou int, ou str, ou outro tipo. Veja que agora, ao usar o método info(), temos que ambas colunas são do tipo datetime (datetime da biblioteca pandas).

O formato resultante ano-mês-dia é um padrão do datetime64[ns], que segue o padrão internacional, no qual o ano vem primeiro, seguido do mês e por último o dia. Poderíamos usar o stftime() para transformar o traço em barra (/), mas aí o resultado seriam strings e não datas.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Series.str: Quando selecionamos uma coluna no DF sabemos que o resultado é uma Series e esse objeto tem um recurso "str", que permite aplicar as funções de string para todos os valores da Series.

Acessamos o recurso str e aplicamos o método upper(). Dessa forma, a biblioteca pandas "entende" que queremos converter todos os valores dessa coluna para letras maiúsculas.

Método sort_values(): sort_values() que permite ordenar o DF, de acordo com os valores de uma coluna. Esse método é do DataFrame, por isso a notação meu_df.metodo(). Utilizamos três parâmetros do método sort_values, o primeiro informando qual coluna deve ser usada para ordenar, o segundo, para que seja feito em ordem decrescente (do maior para o menor) e o terceiro (inplace=True) significa que queremos modificar o próprio objeto, na prática estamos sobrescrevendo o DF.

```
df_selic.sort_values(by='data', ascending=False, inplace=True)
df_selic.head()
```

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Método reset_index() e set_index()

Nenhuma transformação afeta o índice, lembra-se como não especificamos rótulos ele usa um intervalo numérico, mas esse intervalo é diferente da posições de um vetor, pois é um nome e vai acompanhar a linha independente da transformação. As únicas formas de alterar o índice são com os métodos reset_index() e set_index(). O primeiro redefine o índice usando o padrão e o segundo utiliza define novos índices.

Durante a transformação dos dados, pode ser necessário definir novos valores para os índices, ao invés de usar o range numérico. Essa transformação pode ser feita usando o método meu_df.set_index(). O método permite especificar os novos valores usando uma coluna já existente ou então passando uma lista, de tamanho igual a quantidade de linhas.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Filtros com loc

Um dos recursos mais utilizados por equipes das áreas de dados é a aplicação de filtros. A distinção por gênero pode ser um filtro. Esse filtro vai possibilitar comparar a idade de todos, com a idade de cada grupo e entender se as mulheres ou homens estão abaixo ou acima da média geral.

DataFrames da biblioteca pandas possuem uma propriedade chamada loc, essa propriedade permite acessar um conjunto de linhas (filtrar linhas), por meio do índice ou por um vetor booleano (vetor de True ou False).

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Filtros com testes booleanos

Podemos usar operadores relacionais e lógicos para fazer testes condicionais com os valores das colunas de um DF. Ao criarmos um teste condicional, internamente, a biblioteca testa todas as linhas do DF ou da Series, retornando uma Series booleana, ou seja, composta por valores True ou False.

O teste condicional pode ser construído também utilizando operadores lógicos. Para a operação lógica AND (E), em pandas, usa-se o caracter &. Para fazer a operação lógica OR (OU), usa-se o caracter. Cada teste deve estar entre parênteses, senão ocorre um erro. Ao criar as condições, basta aplicá-las no DataFrame para criar o filtro. A construção é feita passando a condição para a propriedade loc.

INTRODUÇÃO A MANIPULAÇÃO DE DADOS EM PANDAS

Banco de dados com pandas

Além dos métodos para acessar arquivos, a biblioteca pandas possui dois métodos que permitem executar instruções SQL em banco de dados. Os métodos são:

pandas.read_sql(sql, con, index_col=None, coerce_float=True, params=None, parse_dates=None, columns=None, chunksize=None)

pandas.read_sql_query(sql, con, index_col=None, coerce_float=True, params=None, parse_dates=None, chunksize=None)

O mínimo de parâmetros que ambos métodos exigem é a instrução SQL e uma conexão com um banco de dados (con). A conexão com o banco de dados, deve ser feita usando uma outra biblioteca, por exemplo, sqlalchemy (suporte para diversos bancos), pyodbc para SQL Server, cx_Oracle para Oracle, psycopg2 para Postgresql, dentre outras.

Quais as vantagens de utilizar o recurso DataFrame?

Exercício: Quais as funções dos métodos de leitura JSON e CSV

Quais as funções dos métodos de leitura JSON e CSV?

JSON (JavaScript Object Notation - Notação de Objetos JavaScript) é uma formatação leve de troca de dados e independente de linguagem de programação. Os dados nesse formato podem ser encontrados como uma coleção de pares chave/valor ou uma lista ordenada de valores (https://www.json.org/json-pt.html). Uma chave pode conter uma outra estrutura interna, criando um arquivo com multiníveis. Nessa aula vamos estudar somente a leitura de arquivos com um único nível, ou seja, para uma chave há somente um valor.

CSV (comma-separated values - valores separados por vírgula) é um formato de arquivo, nos quais os dados são separados por um delimitador. Originalmente, esse delimitador é uma vírgula (por isso o nome), mas na prática um arquivo CSV pode ser criado com qualquer delimitador, por exemplo, por ponto e vírgula (;), por pipe (|), dentre outros. Por ser um arquivo de texto, é fácil de ser lido em qualquer sistema, por isso se tornou tão democrático.

VISUALIZAÇÃO DE DADOS EM PYTHON

VISUALIZAÇÃO DE DADOS EM PYTHON

Bibliotecas e funções para criação de gráficos

Para a criação de gráficos em Python são utilizadas as bibliotecas matplotib e outras baseadas na matplotlib, e também funções que permitem criar e personalizar os gráficos.

Matplotlib

A instalação da biblioteca pode ser feita via pip install: pip install matplotlib, lembrando que em ambientes como o projeto Anaconta e o Colab esse recurso já está disponível.

VISUALIZAÇÃO DE DADOS EM PYTHON

Existem duas sintaxes que são amplamente adotadas para importar essa biblioteca para o projeto:

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

Em ambas formas de importação utilizamos o apelido "plt" que é uma convenção adotada para facilitar o uso das funções. Ao trabalharmos no jupyter notebook com o kernel do IPython (o kernel do IPython é o backend de execução do Python para o Jupyter), podemos habilitar uma opção de fazer a impressão do gráfico "inline", ou seja, no próprio notebook. Para habilitar utiliza-se a sintaxe: %matplotlib inline.

Portanto, projetos no jupyter notebook, que utilizem o matplotlib sempre terão no começo, os comandos a seguir.

```
from matplotlib import pyplot as plt
%matplotlib inline
```

VISUALIZAÇÃO DE DADOS EM PYTHON

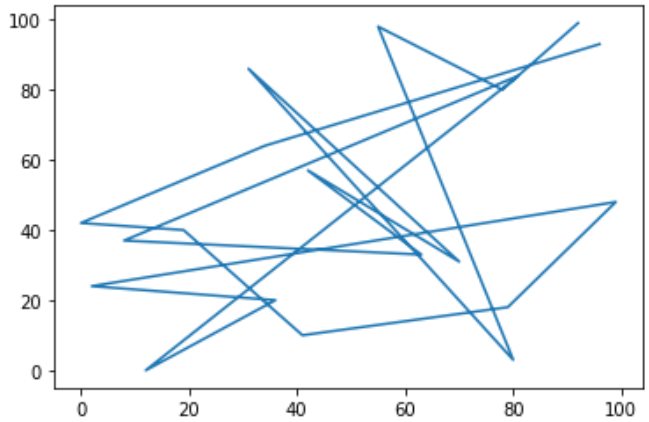
Vamos criar duas listas aleatórias de valores inteiros com o módulo random e então plotar um gráfico de linhas, com a função plt.plot() do módulo pyplot.

Após criar duas listas com valores aleatórios, a função plot() as recebe como parâmetros, utilizando-as para os valores dos eixos horizontal (x) e vertical (y) e já cria o gráfico. Parâmetros e muitos outros, podem ser configurados!

```
import random


dados1 = random.sample(range(100), k=20)
dados2 = random.sample(range(100), k=20)
```

```
[<matplotlib.lines.Line2D at 0x1cf32995940>]
```




```
plt.plot(dados1, dados2) # pyplot gerencia a figura e o eixo
```

VISUALIZAÇÃO DE DADOS EM PYTHON

Existem essencialmente duas maneiras de usar o Matplotlib:

- Deixar para o pyplot criar e gerenciar automaticamente figuras e eixos, e usar as funções do pyplot para plotagem.

- Criar explicitamente figuras e eixos e chamar métodos sobre eles (o "estilo orientado a objetos (OO)").

No gráfico que criamos, nós utilizamos a opção 1, ou seja, foi o próprio módulo que criou o ambiente da figura e do eixo. Ao utilizar a segunda opção, podemos criar uma figura com ou sem eixos, com a função plt.subplots(), que quando invocada sem parâmetros, cria um layout de figura com 1 linha e 1 coluna.

VISUALIZAÇÃO DE DADOS EM PYTHON

Biblioteca pandas

As principais estruturas de dados da biblioteca pandas (Series e DataFrame) possuem o método plot(), construído com base no matplotlib e que permite criar gráficos a partir dos dados nas estruturas.

A partir de um DataFrame, podemos invocar o método: df.plot(*args, **kwargs) para criar os gráficos. Os argumentos dessa função, podem variar, mas existem três que são triviais: os nomes das colunas com os dados para os eixos x e y, bem como o tipo de gráfico (kind).

VISUALIZAÇÃO DE DADOS EM PYTHON

Biblioteca seaborn

Seaborn é outra biblioteca Python, também baseada na matplotlib, que foi desenvolvida especificamente para criação de gráficos. Seaborn pode ser instalado via pip install: pip install seaborn, e para utilizar no projeto existe uma convenção para sintaxe: import seaborn as sns. A biblioteca conta com um repositório de datasets que podem ser usados para explorar as funcionalidades.

O tipo de dados que uma coluna possui é muito importante para a biblioteca seborn, uma vez que as funções usadas para construir os gráficos são dividas em grupos: relacional, categóricos, distribuição, regressão, matriz e grid.

VISUALIZAÇÃO DE DADOS EM PYTHON

Função barplot()

Dentro do grupo de funções para gráficos de variáveis categóricas, temos o barplot(), que permite criar gráficos de barras, mas por que usaríamos essa função e não a da biblioteca pandas? A resposta está nas opções de parâmetros que cada biblioteca suporta.

Construtor da função barplot possui uma série de parâmetros estatísticos, que dão muita flexibilidade e poder aos cientista de dados, vamos falar sobre o parâmetro "estimator", que por default é a função média. Isso significa que cada barra do gráfico, exibirá a média dos valores de uma determinada coluna, o que pode não fazer sentido, uma vez que queremos exibir a quantidade dos valores (len) ou a soma (sum).

Existem muitos conceitos estatísticos envolvidos e a biblioteca seaborn fornece mecanismos para que essas informações estejam presentes nos resultados visuais.

VISUALIZAÇÃO DE DADOS EM PYTHON

Função countplot()

Esse método não aceita que sejam passados valores de x e y ao mesmo tempo, pois a contagem será feita sobre uma variável categórica, portanto devemos especificar x ou y, a diferença será na orientação do gráfico. Se informamos x, teremos uma gráfico na vertical, se y, na horizontal.

Função scartterplot()

Os gráficos do grupo relacional, permitem avaliar, de forma visual a relação entre duas variáveis: x, y. O gráfico scatterplot é muito utilizado por cientistas de dados que estão buscando por padrões nos dados.

APLICAÇÃO DE BANCO DE DADOS COM PYTHON

Criamos um objeto cursor. Na linha 2, invocamos o método execute() desse objeto para, enfim, criar a tabela pelo comando armazenado na variável ddl_create. Como o cursor é uma classe, ele possui métodos e atributos. Nas linhas 6 e 7 estamos acessando os atributos description e rowcount.

O primeiro diz respeito a informações sobre a execução; e o segundo a quantas linhas foram afetadas. No módulo sqlite3, o atributo description fornece os nomes das colunas da última consulta. Como se trata de uma instrução DDL, a description retornou None e a quantidade de linhas afetadas foi -1. Todo cursor e toda conexão, após executarem suas tarefas, devem ser fechados pelo método close().

```
cursor = conn.cursor()

cursor.execute(ddl_create)

print(type(cursor))


print("Tabela criada!")

print("Descrição do cursor: ", cursor.description)

print("Linhas afetadas: ", cursor.rowcount)

cursor.close()

conn.close()
```

Qual a biblioteca foi desenvolvida especificamente para criação de gráficos?

Recapitulando

Recapitulando

- Linguagem de programação: Introdução a Biblioteca Pandas
- Introdução a Manipulação de Dados em Pandas
- Visualização de Dados em Python