*Visualizações em Python (Parte II)*

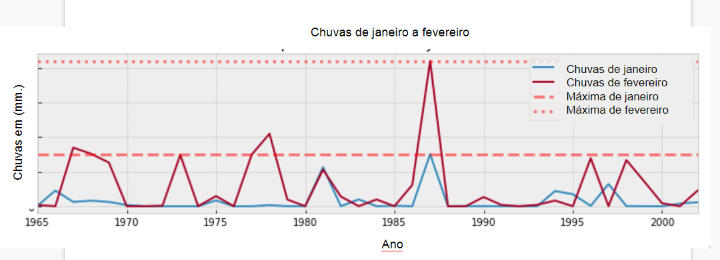
A visualização dos dados é um aspecto fundamental em qualquer projeto de Data Science, tanto como ferramenta de análise quanto para a comunicação efetiva dos resultados. Para esta tarefa, existem várias bibliotecas de visualização em Python, entre as quais podemos citar *Matplotlib, Seaborn, Plotly* e *Bokeh*. Nesta aula, enfocaremos especialmente a **Matplotlib** por sua versatilidade, simplicidade e facilidade de uso. Para representar graficamente as distribuições de variáveis ​​categóricas, faremos uso da **Seaborn**, uma vez que fornece um conjunto de comandos que facilitam essa tarefa.

Enriquecendo as visualizações

Múltiplos elementos em um mesmo gráfico

É possível combinar vários elementos dentro do mesmo gráfico ou eixo (objeto ax). Por exemplo, poderíamos representar graficamente as precipitações de janeiro e fevereiro para anos diferentes e ao mesmo tempo destacar o máximo e o mínimo de cada um com uma linha pontilhada horizontal da seguinte forma:

| fig, ax = plt.subplots(figsize=(12, 3))  ax.plot(df\_chuvas.index, df['Jan'], label='Precipitações de janeiro') ax.plot(df\_chuvas.index, df['Feb'], label='Precipitações de fevereiro', cor='C1')  máximo\_janeiro = df\_chuvas['Jan'].max() máximo\_fevereiro = df\_chuvas['Feb'].max()  ax.axhline(máximo\_janeiro, cor='red', linestyle='--', alpha=0.5, linewidth=3, label='Máxima de janeiro') ax.axhline(máximo\_fevereiro, cor='red', linestyle=':', alpha=0.5, linewidth=3, label='Máxima de fevereiro')  ax.set\_xlabel('Ano') # Etiqueta eixo x ax.set\_ylabel('Precipitação (mm.)') # Etiqueta eixo y ax.set\_title('Precipitações de janeiro e fevereiro') # Título ax.set\_xlim(df\_chuvas.index[0], df\_chuvas.index[-1]) ax.legend() # Inserir a legenda |
| --- |



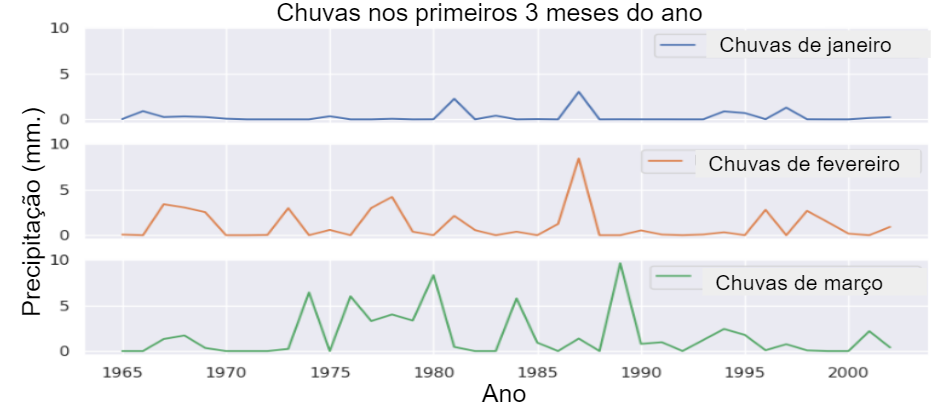
**Observação:** ax.axhline() permite adicionar linhas horizontais na altura especificada pelo parâmetro. Da mesma forma, ax.axvline() permite adicionar linhas verticais.

Quando vários elementos estão representados graficamente no mesmo objeto x, é aconselhável não sobrecarregá-lo, pois isso pode dificultar a leitura. Lembre-se de que o objetivo de qualquer visualização é transmitir sua intenção da maneira mais simples possível.

Subgráficos

É possível gerar vários gráficos dentro da mesma figura. Isso pode ser útil, por exemplo, se quisermos comparar variáveis ​​diferentes sem sobrecarregar um gráfico específico. Para fazer isso, passaremos como parâmetros nrows e ncols para a função plt.*subplots*() que representam o número de linhas e o número de colunas do gráfico respectivamente. Ao fazer isso, o objeto ax se torna uma matriz, onde cada elemento é um objeto Axes específico. Para adicionar uma linha ao primeiro gráfico, vamos nos referir aos primeiros elementos da matriz ax, por ax [0]. Para adicionar uma reta ao segundo gráfico, faremos isso com ax [1] e assim por diante.

| fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 5), sharex=True, sharey=True)  ax[0].plot(df\_chuvas.index, df\_chuvas['Jan'], label='Precipitações de janeiro') ax[1].plot(df\_chuvas.index, df\_chuvas['Feb'], label='Precipitações de fevereiro', cor='C1') ax[2].plot(df\_chuvas.index, df\_chuvas['Mar'], label='Precipitações de março', cor='C2')  ax[0].set\_title('Precipitações dos primeiros três meses do ano)  ax[2].set\_xlabel('Ano')  ax[1].set\_ylabel('Precipitação (mm.)')   ax[0].legend()  ax[1].legend() ax[2].legend() |
| --- |



**Observações:**

* Observe os parâmetros *sharex* e *sharey* em *plt.subplots*(). Isso faz com que as subparcelas compartilhem os eixos e, portanto, matplotlib apenas adiciona as caixas dos anos no último gráfico. Por sua vez, sharey é particularmente útil nesta ocasião, pois força todos os subgráficos a terem a mesma escala no eixo y, permitindo ao leitor comparar facilmente quais meses tiveram mais ou menos chuva. À primeira vista você pode ver que janeiro geralmente acumula menos precipitação do que fevereiro e março, o que indica que é um mês seco.
* Cada subgráfico pode ter seu próprio título e rótulos de eixo. Neste caso, adicionamos um único título geral ao primeiro subgráfico e adicionamos a etiqueta do eixo horizontal que é adicionado apenas ao último subgráfico. O último é recomendado, desde que os subgráficos compartilhem o eixo x.
* Esses tipos de gráficos de várias linhas também são conhecidos como *gráficos tandem*.
* Também é possível gerar *grades de gráfico*. Se, por exemplo, passarmos *nrows = 2* e *ncols = 2* para o método *plt.subplots(),* o objeto ax torna-se uma *matriz* ou *grade de gráfico*. Se quiséssemos plotar uma linha no subplot superior direito, teríamos que executar *ax [0, 1] .plot()* referindo-se ao gráfico da primeira linha e da segunda coluna. Se, em vez disso, quisermos adicionar um histograma ao subgráfico no canto direito inferior, devemos fazê-lo com *ax[1,1].hist().*

Modificando os parâmetros globais da Matplotlib

Um aspecto importante é manter um estilo consistente nos diferentes gráficos que fazemos. A Matplotlib facilita a personalização de seus gráficos, modificando seus parâmetros globais. Ao importar a biblioteca, a Matplotlib estabelece seus parâmetros padrão que definem as diferentes características das visualizações que gera, como o tamanho das etiquetas, o tamanho dos eixos, os tamanhos das fontes, a tipografia, a espessura das linhas e muitos mais.

Uma extensa lista de parâmetros modificáveis ​​pode ser vista com a seguinte linha:

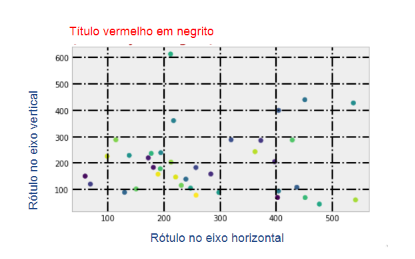
| mpl.rcParams.keys() |
| --- |

Alguns dos parâmetros mais importantes são os seguintes:

* **'axes.grid':** Permite ocultar a grade se estiver definida como False. Por padrão, é True.
* **'axes.titleweight':** A espessura da tipografia dos títulos. Os valores válidos são 'bold', 'normal', 'light'. Por padrão, é 'normal'.
* **'axes.titlelocation':** permite que você modifique a posição do título. Por padrão, os títulos são centralizados porque este parâmetro tem um valor de "centro". Você pode alinhar o título à esquerda atribuindo-o à "left".
* **'axes.titlecor':** cor do título. Os núcleos pu.
* **'axes.grid.axis':** Eixos da grade, por padrão, é definido como 'ambos', o que significa que a grade consiste em linhas verticais e horizontais.
* **'axes.labelcor':** Cor da etiqueta dos eixos. Por padrão, é 'black'.
* **'axes.labelsize':** Tamanho das etiquetas. Por padrão, é 'large'.
* **'axes.labelweight':** espessura da fonte do rótulo. Por padrão, é 'normal', mas os rótulos podem ser colocados em negrito atribuindo 'bold'.
* **'axes.linewidth':** A espessura dos eixos, por padrão 0,8.
* **'grid.alpha':** A transparência da grade, por padrão, é 1.0.
* **'grid.cor':** Cor da linha de grade. Por padrão, é hexadecimal cor #b2b2b2.
* **'grid.linestyle':** O estilo de linha da grade. Por padrão '--'.
* **'grid.linewidth':** Largura da linha da grade. Por padrão, é 0,5.
* **'legend.fontsize':** tamanho da fonte da legenda. Por padrão, 'medium'.

Um exemplo exagerando as mudanças

| mpl.rcParams['axes.titleweight'] = 'bold'  mpl.rcParams['axes.titlelocation'] = 'left'  mpl.rcParams['axes.titlecor'] = 'firebrick'  mpl.rcParams['axes.labelcor'] = 'blue'  mpl.rcParams['axes.labelsize'] = '10'  mpl.rcParams['axes.labelweight'] = 'light'  mpl.rcParams['axes.linewidth'] = '1'  mpl.rcParams['grid.cor'] = 'black'  mpl.rcParams['grid.linestyle'] = '-.'  mpl.rcParams['grid.linewidth'] = '2'   fig, ax = plt.subplots(figsize=(7, 4))  ax.scatter(df\_chuvas['Aug'], df\_chuvas['Sep'], c=df\_chuvas.index)  ax.set\_title('(Título vermelho em negrito)') ax.set\_xlabel('(Etiqueta eixo horiz.)') ax.set\_ylabel('(Etiqueta eixo vert.)') |
| --- |



Uma maneira (talvez mais detalhada) de obter o mesmo resultado é usando o método plt.rc() em vez de atribuir os valores ao dicionário mpl.rcParams:

| plt.rc('axes', titlelocation='left', titlecolor='firebrick', ...) plt.rc('grid', cor='black', linestyle='-.', ...) |
| --- |

Para restabelecer os parâmetros padrão, execute a seguinte linha:

| mpl.rcParams.update(mpl.rcParamsDefault) |
| --- |

Uma alternativa a Matplotlib: a livraria Seaborn

*A Seaborn* é uma biblioteca de visualização construída sobre o matplotlib. Uma das características das Seaborn que justifica a sua utilização é sua integração com as estruturas de dados do Pandas, o que permite gerar visualizações de frames de dados com muita facilidade e em poucas linhas de código.

A Seaborn oferece seus próprios estilos gráficos. Você pode usar o estilo e núcleos Seaborn e continuar plotando normalmente com a matplotlib, como fez até agora com a seguinte linha:

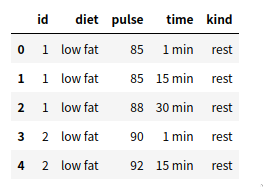
sns.set()

O método *set*() sobrescreve os parâmetros internos do matplotlib para usar o estilo Seaborn. Dessa forma, todos os gráficos subsequentes terão uma aparência melhor.

Embora todas as visualizações anteriores pudessem ser geradas com o Seaborn, deve-se notar que esta tem uma sintaxe um pouco diferente, portanto, não vale a pena usar para gráficos que poderiam ser facilmente feitos com o matplotlib. No entanto, a real vantagem do Seaborn está na visualização de conjuntos de dados com variáveis ​​categóricas.

Para demonstrar isso, vamos importar o conjunto de dados do exercício fornecido pelo próprio pacote.

| df\_exercício = sns.load\_dataset('exercise') df\_exercício = df\_exercício.drop('Unnamed: 0', axis='columns') df\_exercício.head() |
| --- |

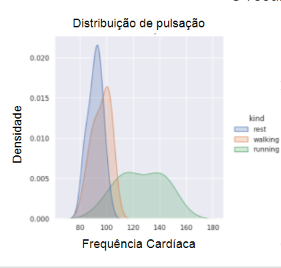


O conjunto de dados contém medições da frequência cardíaca de 30 pessoas diferentes após fazer um determinado tipo de exercício. As colunas são:

* **id:** o id da pessoa sobre quem a medição foi feita.
* **diet:** a dieta daquela pessoa (com baixo teor de gordura ou sem gordura).
* **pulse:** a frequência cardíaca obtida (em batimentos por minuto).
* **time:** tempo decorrido após o início do exercício.
* **kind:** tipo de exercício (repouso, caminhada ou corrida).

Poderíamos estar interessados ​​na forma de distribuição das pulsações para os diferentes tipos de atividade após 30 minutos de ter realizado a mesma:

| plt.figure() df\_30\_min = df\_ejercicio[df['time'] == '30 min'] ax = sns.displot(data=df\_30\_min, x='pulse', kind='kde', hue='kind', fill=True) ax.set(xlabel='Frequência Cardíaca', ylabel='Densidade', title='Distribuição das pulsações') |
| --- |

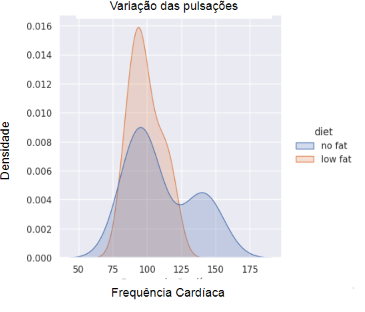


Sem surpresa, as pessoas que correram acabaram com o pulso mais alto do que aquelas que não correram.

**Observações:**

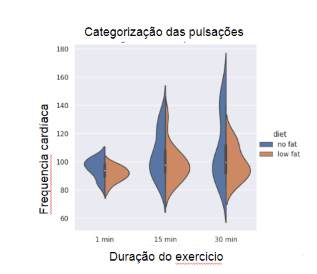
* Em primeiro lugar, o método sns.displot recebe como parâmetro 'dados' que recebe um dataframe Pandas e depois recebe os nomes das colunas a serem apresentadas a partir desse dataframe.
* O parâmetro *hue* determina com base em qual coluna de **variáveis ​​categóricas** o agrupamento deve ser executado. Nesse caso, com base no tipo de exercício.
* O parâmetro fill adiciona sombreado às distribuições.
* El parâmetro kind='kde' afirma que queremos uma estimativa da distribuição de probabilidade das amostras. Se fizermos kind = 'hist', obteremos um histograma.

Se, em vez disso, quisermos comparar aquelas pessoas que têm uma dieta com baixo teor de gordura (categoria de low fat) com aquelas que não consomem gordura (categoria no fat), temos apenas que alterar o parâmetro de matiz anterior para a coluna 'diet'.



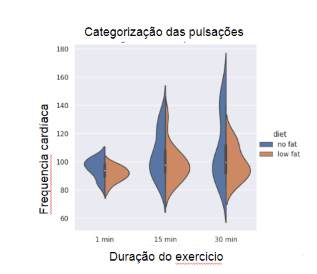
Outra forma de visualizar distribuições de acordo com categorias é pelo método *sns.catplot():*

| ax = sns.catplot(data=df\_exercicio, kind='violin', x='time', y='pulse', hue='diet', split=True) ax.set(xlabel='Duração de exercício', ylabel='Frequência cardíaca', title='Categorização da distribuição de pulsações') |
| --- |



Ou ainda, para diferentes tipos de exercício:

| ax = sns.catplot(data=df\_exercicio, kind='violin', x='kind', y='pulse', hue='diet', split=True) ax.set(xlabel='Duração do exercício', ylabel='Frequência cardíaca', title='Categorização da distribuição de pulsações') |
| --- |



Desta forma, fizemos uma trajetória pelas principais ferramentas de visualização que o Python oferece.