1 Visualização de Dados

Nas aulas anteriores nós trabalhamos a construção de distribuições de frequências, usando tabelas, dos conjuntos de dados crus que nós analisamos.

Nas próximas aulas nós veremos como essas distribuições podem ser representadas visualmente, por meio de gráficos ao invés de tabelas.

Gráficos são ferramentas essenciais na análise de dados e seu uso, tipicamente, causa impactos nos leitores de suas análises muito maiores que apresentação de medidas e/ou tabelas. Arrisco dizer que, sem gráficos, os dados não estão completamente analisados.

Existem diversas bibliotecas em Python para visualização de dados: a matplotlib (https://matplotlib.org/), biblioteca básica de geração de gráficos em Python, que possibilita a criação de uma enorme gama de gráficos com alta qualidade de acabamento e com grande possibilidade de customização; a seaborn (https://seaborn.pydata.org/), biblioteca derivada da `matplotlib`, especializada em gráficos estatísticos, com funções que facilitam a interação do usuário para a geração de gráficos, tornando o trabalho mais simples do que usando a `matplotlib` diretamente; a plotly (https://plot.ly/) e a bokeh (https://bokeh.pydata.org/en/latest/), que permitem construir, de forma relativamente simples, gráficos interativos e *dashboards* (O que é um dashboard? (http://marketingpordados.com/analise-de-dados/o-que-e-dashboard-%F0%9F%93%8A/)).

Para a nossa disciplina, nós faremos uso "indireto" da `matplotlib`, chamando-a por meio da `pandas`, que possui algumas funções internas que chamam diretamente funções da `matplotlib` (ver documentação aqui (http://pandas.pydata.org/pandas-docs/stable/visualization.html#bar-plots)). Eventualmente, usaremos funções das outras bibiotecas apresentadas aqui.

Além disso, nos basearemos bastante nos exemplos disponíveis no site Python Graph Gallery (https://python-graph-gallery.com/), um excelente repositório (julgo ser o melhor) para exemplos de códigos para gerar diversos tipos de gráficos em python e no The Data Visualisation Catalogue (https://datavizcatalogue.com), repositório com a descrição de funcionalidades e construção de vários tipos de gráficos.

Mesmo com essas funções chamando indiretamente cada função de geração de gráficos, é interessante, para certas configurações que possam surgir, criamos o hábito de importar a biblioteca `matplotlib`, especificamente o submódulo `pyplot`, onde se encontram todas as funções responsáveis pel geração dos gráficos.

Assim,

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Aqui vale um comentário importante.

Para que o gráfico apareça, tipicamente, temos que usar o método `plt.show()` logo abaixo do gráfico. Essa função, obviamente, mostra o gráfico em uma janela (quando executamos o código `.py` em uma IDE convencional ou quando executamos comandos em um terminal) ou *inline*, quando usamos um notebook.

Acontece que notebooks herdaram comandos especiais do IPython, e temos uma opção mais elegante para não ficar escrevendo `plt.show()` o tempo inteiro: basta colocar, logo após importar a biblioteca, o comando `%matplotlib inline` . Feito isso, não existe mais a necessidade de escrever o `plt.show()` após cada gráfico gerado.

Vale salientar também que versões mais recentes do Jupyter Noteboob trazem o comando por padrão, não sendo necessário, nesses casos, explicitá-lo.

In [ ]:

```
%matplotlib inline
```

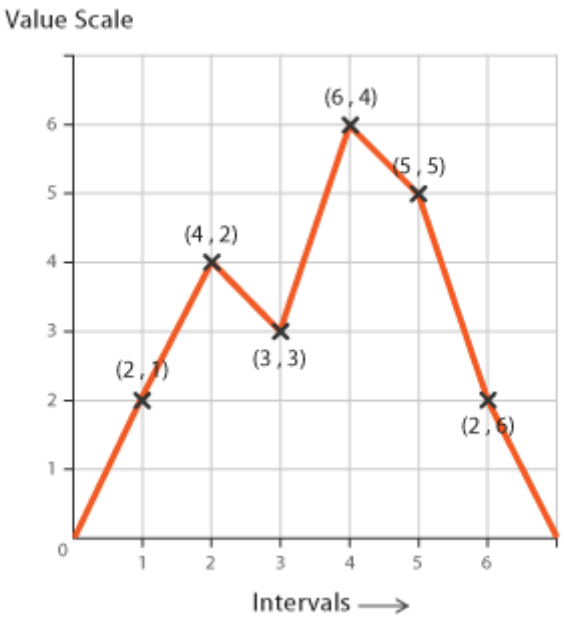
Caso queira que apareça uma janela fora do notebook para visualização do gráfico (opção interessante quando temos muitos gráficos dentro do notebook), basta substituir `inline` por `qt` no comando acima, escrevendo

In [ ]:

```
%matplotlib qt
```

1.1 Gerando um primeiro gráfico: o gráfico em linha

O gráfico mais básico que podemos gerar em qualquer biblioteca de visualização de dados é um gráfico em linha. Basicamente, gráficos em linha são gráficos feitos a partir de um plano cartesiano, interligando pares de ponto nesse plano, como ilustrado na figura abaixo.



Para plotar um gráfico em linha, a biblioteca `matplotlib` tem a função `plot` , cujos argumentos básicos são os conjuntos de valores de x e y, tipicamente numéricos.

Vejamos um exemplo:

In [ ]:

```
x = np.linspace(1,20,100)
y = x**2
plt.plot(x,y)
```

Exercise 1 Dê uma olhada na documentação da função plot (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html), e veja como modificar o tipo de linha, a cor dela, acrescentar um *grid* e um título ao seu gráfico e aos eixos.

Para gráficos estatísticos, podemos usar, ao invés de chamar diretamente a `matplotlib`, podemos usar o método `plot` da `pandas` que, basicamente, chama indiretamente a função da `matplotlib`, mas de uma forma muito mais integrável às estruturas de dados da `pandas`.

Como um exemplo básico, podemos transformar a função criada no exemplo anterior em uma `series` e, então, plotar seu gráfico.

In [ ]:

```
ts = pd.Series(y)
ts.plot()
```

Obviamente, obteremos o mesmo resultado.

Exercise 2 As configurações feitas no exercício anterior podem ser usadas também no método **plot** da **pandas**?

Resposta:

Ver a documentação disponível aqui (https://pandas.pydata.org/pandas-docs/stable/visualization.html#general-plot-style-arguments).

No caso, podemos fazer

In [ ]:

```
ts.plot(style='')
```

No contexto de estatística, o gráfico em linha é bastante utilizado para mostrar evolução de um conjunto de dados, tipicamente de dados que variam ao longo do tempo. Conjuntos de dados assim são chamados de séries temporais.

O exemplo a seguir cria uma série temporal com dados sintéticos (criados artificialmente), e plota o gráfico da soma cumulativa desses valores.

Os dados, basicamente, são 1000 valores aleatórios gerados com uma distribuição gaussiana, alocados para uma janela de tempo de 1000 dias a contar de 01/01/2000 (a `pandas` tem um conjunto de métodos bem interessantes para se trabalhar com datas e tempo).

In [ ]:

```
ts = pd.Series(np.random.randn(1000),
               index=pd.date_range('1/1/2000',
                                   periods=1000)).cumsum()
ts
```

In [ ]:

```
ts.plot()
plt.show()
```

O mesmo método `plot` também pode ser usado diretamente em um dataframe, gerando uma curva diferente para cada uma das variáveis (colunas) disponíveis no conjunto de dados.

In [ ]:

```
df = pd.DataFrame(np.random.randn(1000, 4),
                  index=ts.index,
                  columns=list('ABCD')).cumsum()
df.plot()
```

Nós exploraremos dados de séries temporais em alguns exemplos ao longo do curso. Nesse momento, precisamos conhecer alguns gráficos básicos, bastante comuns, que servirão de base para construção de gráficos mais elaborados ou como comparativo para outras análises gráficas.

Em aulas anteriores, nós falamos sobre distribuições de frequência de variáveis qualitativas e quantitativas, vimos como fazer essas distribuições e, conforme dito na aula, foi destacado que classificar variáveis tem grande importância, pois os métodos de análise dos dados mudam de acordo com o tipo de variável.

Isso não é diferente para gráficos. Existem gráficos que são exclusivamente usados para dados categóricos, outros para dados numéricos. Essa será nossa primeira separação.

Os gráficos gerados nas sessões seguintes são, basicamente, representações gráficas das distribuições de frequência estudadas anteriormente. Cada parte ou representação desses gráficos diz respeito a uma categoria, ou classe numérica vista nas tabelas de distribuição de frequência.

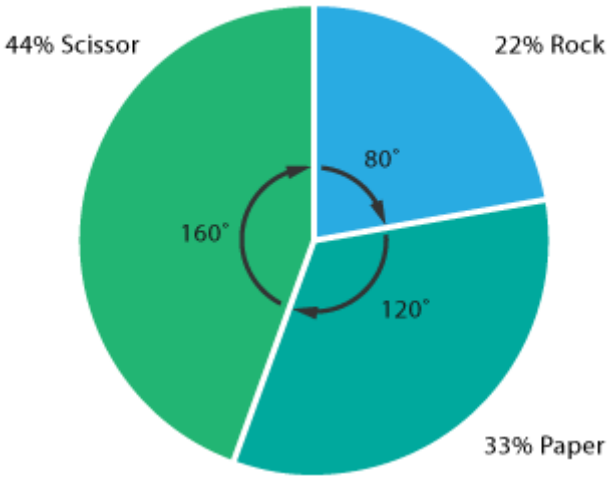
1.2 Gráficos básicos para variáveis qualitativas

1.2.1 Gráficos em setores

O gráfico em setores (ou pizza, ou torta, ou setorizado) é um tipo bastante comum de gráfico, extensivamente usado em ambientes empresariais e escritórios para apresentar resultados.

Basicamente, o gráfico representa cada proporção da frequência relativa como uma "fatia" ou setor, obtida pela divisão do círculo pelo ângulo associado à essa proporção. Assim como a soma de todas as frequências relativas deve somar 100% (se representada de forma percentual), a soma de todos os ângulos de cada setor deve somar $360°$.

O gráfico abaixo ilustra a anatomia desse tipo de gráfico. Logo em seguida, nós executamos esse mesmo exemplo usando o método `pie` , da `Pandas` .



Data			
Rock	Paper	Scissor	TOTAL
2	3	4	9
To calculate percentages			
2/9=22%	3/9=33%	4/9=44%	100%
Degrees for each "pie slice"			
(2/9) x 360 = 80°	(3/9) x 360 = 120°	(4/9) x 360 = 160°	360°

Primeiro, criaremos uma `Series` com resultados (escolhidos por mim) que condizem com os da figura acima.

In [ ]:

```
var = pd.Series(['Rock','Paper','Rock','Scissor','Scissor',
                 'Paper', 'Paper', 'Scissor', 'Scissor'])
var
```

Podemos, então, obter as frequências absoluta e relativa desse conjunto de dados, inclusive em forma percentual,

In [ ]:

```
a = var.value_counts()
a
```

In [ ]:

```
var.value_counts(normalize=True)*100
```

e podemos, então, gerar o gráfico, como se segue, o argumento `figsize` , do método `pie` , basicamente tem a função de deixar o gráfico em proporção.

In [ ]:

```
a.plot.pie(figsize=(6, 6))
# plt.title("Faltou Spock")
```

Exercise 3 Você deve ter percebido que o gráfico foi aplicado sobre a distribuição de frequência (a) e não o conjunto de dados crus (var). Explique porque isso é necessário.

O método `pie` também pode ser usado diretamente em `DataFrames`, como ilustra o exemplo abaixo.

In [ ]:

```
df = pd.DataFrame({'massa': [0.330, 4.87 , 5.97],
                   'raio': [2439.7, 6051.8, 6378.1]},
                   index=['Mercúrio', 'Vênus', 'Terra'])
df
```

In [ ]:

```
df.plot.pie(y='massa', figsize=(5, 5))
```

Usando o argumento `subplots=True` é possível gerar mais de um gráfico em setores do mesmo `DataFrame`.

In [ ]:

```
df.plot.pie(figsize=(10, 5), subplots=True)
```

Exercise 4 É possível gerar gráficos em setores de um DataFrame inteiro, sem selecionar uma coluna especificamente? Justifique sua resposta.

Exercise 5 Quais as desvantagens de se usar gráficos em setores? Pesquise nas referências disponíveis no começo desse notebook.

1.2.1.1 Uma variação mais elegante: o gráfico de rosca

Os gráficos em setores às vezes são criticados por induzirem os leitores a focar nas áreas proporcionais das fatias entre si e no gráfico como um todo. Isso dificulta a visualização das diferenças entre as fatias, especialmente quando você tenta comparar vários gráficos juntos.

Um gráfico de rosca, de certa forma, remedia esse problema ao não enfatizar o uso da área. Em vez disso, os leitores se concentram mais em ler o comprimento dos arcos, em vez de comparar as proporções entre as fatias.

Além disso, os gráficos de rosca são mais eficientes em termos de espaço do que os Gráficos de pizza, porque o espaço em branco dentro de um gráfico de rosca pode ser usado para exibir informações dentro dele.

Para criar um gráfico de rosca, basicamente nós criaremos um círculo branco e, depois, sobreporemos ele ao gráfico em setores convencional, usando os métodos gcf (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.gcf.html) e gca

(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.gca.html) da `matplotlib` .

In [ ]:

```
circulo=plt.Circle( (0,0), 0.7, color='white')
```

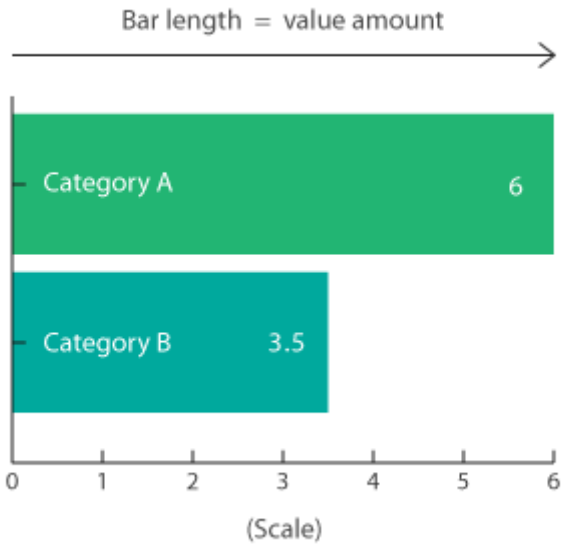
In [ ]:

```
df.plot.pie(y='massa', figsize=(5, 5))
p=plt.gcf()
p.gca().add_artist(circulo)
```

Exercise 6 Como alterar as cores e a ordem das fatias de um gráfico em setores?

1.2.2 Gráficos em barras para uma variável

O gráfico de barras clássico usa barras horizontais ou verticais (gráfico de colunas) para fazer comparações entre as categorias de uma dada variável, seja usando a frequência absoluta, seja a relativa. Um eixo do gráfico mostra as categorias específicas sendo comparadas e o outro eixo representa uma escala de valores discretos, como mostra a figura abaixo.



As tabelas de barras são diferenciadas dos histogramas, pois não exibem desenvolvimentos contínuos durante um intervalo. Os dados discretos do gráfico de barras são dados categóricos e, portanto, respondem à pergunta "quantos?" em cada categoria.

No `pandas` , usamos o método `bar` para gerar gráficos em barras, como mostra o exemplo a seguir.

In [ ]:

```
df = pd.DataFrame({'velocidade': [0.1, 17.5, 40, 48, 52, 69, 88],
                   'tempo de vida': [2, 8, 70, 1.5, 25, 12, 28]},
                   index=['caracol', 'porco', 'elefante', 'coelho', 'girafa', 'coio
df
```

In [ ]:

```
df['velocidade'].plot.bar()
```

Exercise 7 Gerer um gráfico em setores da mesma variável do gráfico acima e compare os dois gráficos, enfatizando, principalmente, a transmissão da informação para o leitor.

Uma grande falha nos gráficos de barras é que a rotulagem se torna problemática quando há um grande número de barras. Por esse motivo, existe uma variação do método `bar` , o `barh` que gera o gráfico com as barras dispostas horizontalmente. Assim, o exemplo anterior pode aparecer como

In [ ]:

```
df['velocidade'].plot.barh()
```

Foi visto também que, quando queremos investigar a relação entre duas variáveis categóricas, construir tabelas de contingência são uma excelente escolha para verificar essas relações. Um exemplo disso é o próprio `DataFrame` desse exemplo que relaciona as espécimes animais com duas de suas características.

Até agora, todos os exemplos de gráficos em barra foram feitos para um só atributo, nesse contexto.

No caso de um `DataFrame` , para fazer isso, podemos plotar os gráficos de cada atributo lado a lado, usando o argumento `subplots=True` , que possibilita, na mesma figura, ter dois gráficos.

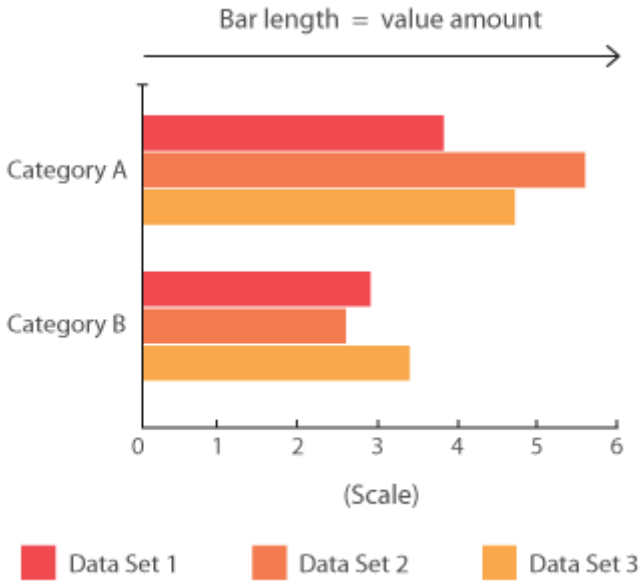
In [ ]:

```
df.plot.bar(subplots=True)
```

Exercise 8 Explique o resultado obtido quando fazemos **df.plot.bar(y="velocidade", x='tempo de vida')**. E se invertermos x e y?

1.2.3 Gráficos em barras para várias variáveis: barras agrupadas e empilhadas

Esse arranjo de gráficos lado a lado nem sempre ajuda na comparação entre os atributos plotados. Para tal objetivo, pode-se fazer uso de um gráfico de barras agrupado.



Essa variação de um gráfico de barras é usada quando duas ou mais séries de dados são plotadas lado-a-lado e agrupadas em categorias, todas no mesmo eixo. Se quisermos representar uma tabela de contingência por gráficos em barras, por exemplo, agrupar as barras de todos os atributos considerados é uma excelente opção.

Como um gráfico de barras, o comprimento de cada barra é usado para mostrar comparações numéricas discretas entre as categorias. Cada série de dados é atribuída a uma cor individual ou a um tom variável da mesma cor, a fim de distingui-las. Cada grupo de barras é então espaçado um do outro.

Gráficos de barras agrupadas geralmente são usados para comparar variáveis ou categorias agrupadas a outros grupos com essas mesmas variáveis ou tipos de categorias.

Para obter gráficos desse tipo, simplesmente temos de aplicar o método `bar` ao `DataFrame` , como se segue.

In [ ]:

```
df.plot.bar()
```

In [ ]:

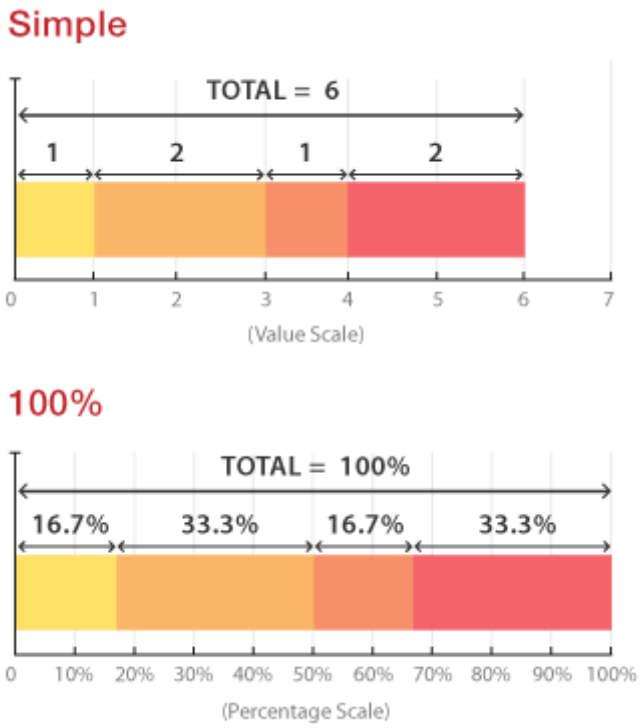
```
df.plot.barh()
```

Quando a quantidade de atributos é grande, esse formato dos gráficos dificulta a leitura, pois ou as barras se sobrepõe, ou sua espessura diminui a ponto de se ter dificuldade de identificar cada barra. Uma opção, nesses casos, é trabalhar com gráficos de barras empilhadas.

Ao contrário de um gráfico de barras agrupadas, que exibe suas barras lado a lado, os gráficos de barras empilhadas segmentam suas barras de vários conjuntos de dados uma sobre a outra. Eles são usados para mostrar como uma categoria maior é dividida em categorias menores e qual a relação de cada parte com a quantidade total. Existem dois tipos de gráficos de barras empilhadas:

- Gráficos de barras empilhadas simples: colocam cada valor para o segmento após o anterior. O valor total da barra é todos os valores do segmento adicionados juntos. Ideal para comparar as quantidades totais em cada grupo / barra segmentada.

- Gráficos de barras empilhadas percentuais: mostram a porcentagem de todo o grupo e são plotados pela porcentagem de cada valor para o valor total de cada grupo. Isso facilita a visualização das diferenças relativas entre as quantidades em cada grupo.



Para fazer um gráfico de barras empilhadas, basta inserir o argumento `stacked=True` ao método `bar` em um `DataFrame` , como mostrado a seguir.

In [ ]:

```
df.plot.bar(stacked=True)
```

Exercise 9 Repita o exemplo acima usando um gráfico em barras percentuais.

1.3 Gráficos básicos para variáveis quantitativas

Para variáveis quantitativas, *a priori*, veremos dois gráficos: o histograma e o gráfico de densidade. A medida que o curso evoluir, teremos mais alguns gráficos, como o box-plot, por exemplo, que servem para descrever a distribuição de dados numéricos.

Para exemplificar a criação desses gráficos usaremos o seguinte conjunto sintético de dados,

In [ ]:

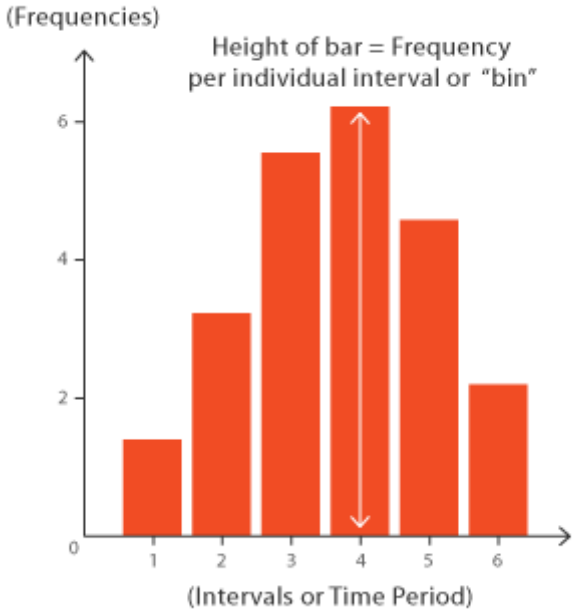
```
df2 = pd.DataFrame({'a': np.random.randn(1000) + 1, 'b': np.random.randn(1000),
                    'c': np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])

df2
```

1.3.1 Histogramas

Um histograma visualiza a distribuição de dados em um intervalo contínuo ou em determinado período de tempo. Cada barra em um histograma representa a frequência tabulada em cada intervalo/caixa (*bins*).

Os histogramas ajudam a estimar onde os valores estão concentrados, quais são os extremos e se existem lacunas ou valores incomuns. Eles também são úteis para dar uma visão aproximada da distribuição de probabilidade.



Para gerar histogramas a partir de `DataFrames` , temos o método `hist` , que pode ser usado como se segue:

In [ ]:
```
df2.plot.hist(bins=10)
```

In [ ]:
```
df2['a'].plot.hist(bins=10)
```

In [ ]:
```
df2['a'].plot.hist(bins=100, grid=True)
```

Exercise 10 Nós vimos, um pouco atrás no curso o uso da função *hist* da *matplotlib*. Gere os mesmos gráficos acima usando essa função e explique os valores que aparecem na saída dessa função.

Exercise 11 Gere histogramas de frequência relativa e acumulada, para os parâmetros dos exemplos acima.

1.3.2 Gráficos de densidade

Um gráfico de densidade visualiza uma estimação da distribuição de dados em um intervalo ou período de

tempo contínuo. Este gráfico é uma variação de um histograma que usa suavização de *kernel* (método para estimar o gráfico da densidade de probabilidade de uma distribuição) para plotar valores, permitindo distribuições mais suaves e contínuas. Os picos de um gráfico de densidade ajudam a exibir onde os valores são concentrados no intervalo.

Gráficos de densidade tipicamente não apresentam o problema que histogramas trazem com relação ao número de classes considerados na geração do gráfico. Poucas classes deixam a distribuição tipicamente uniforme, escondendo a real forma e muitas classes geram o problema de "vales" ou classes sem valores (ver exemplos abaixo). Esses problemas não ocorrem com gráficos de densidade.

In [ ]:

```
df2['a'].plot.hist(bins=4)
```

In [ ]:

```
df2['a'].plot.hist(bins=1000)
```

O método `kde` da `Pandas` permite criar gráficos de densidade a partir de `DataFrames`, como ilustrado abaixo.

In [ ]:

```
df2.plot.kde()
```

Exercise 12 Para cada uma das colunas do dataframe "df2", gere um gráfico que contenha o histograma e o gráfico de densidade sobrepostos. Dica: veja a função *distplot* da biblioteca Seaborn.

Exercise 13 Teste versões de todos os gráficos vistos aqui para o conjunto de dados **train.csv**.