Representações gráficas

Gráficos são formas de se representar visualmente dados ou valores numéricos, para torná-los mais claros e informativos.

Geralmente são utilizados para mostrar padrões, tendências ou comparar informações.

São utilizados em diversas áreas de estudo (matemática, estatística, geografia, economia, história, etc.).

Nessa aula veremos como criar gráficos simples a partir de funções e, principalmente, a partir de bases de dados, em ambos os casos usando Python.

> **De modo algum iremos cobrir nessa aula (ou nessa disciplina) tudo que existe sobre criação de gráficos e muito menos como fazer isso usando Python. Existe muito a ser explorado sobre esses assuntos além do que veremos nesse notebook.**

Gráficos de funções matemáticas

Vamos começar criando alguns gráficos a partir de funções matemáticas.

Python oferece a biblioteca `Matplotlib`, que tem diversas funções prontas para fazer gráficos, e a biblioteca `Numpy`, que tem diversas funções matemáticas prontas.

A opção `%matplotlib notebook` define como o gráfico vai aparecer. Existem outras opções, como `%matplotlib inline`, que você pode explorar depois.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib notebook
```

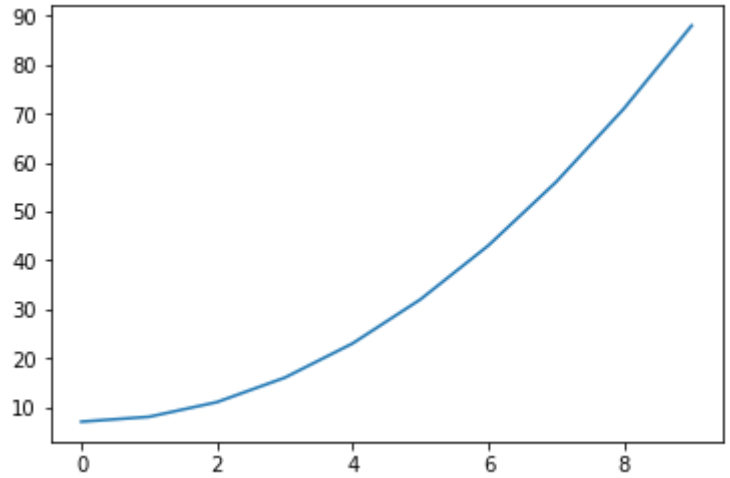
Basicamente, todo gráfico tem um eixo x, que fica na parte horizontal, e um eixo y, que fica na parte vertical.

Vamos ver no exemplo a seguir como gerar um gráfico para a função $x^2 + 7$, para valores de x variando entre 0 e 9, de uma em uma unidade.

A função para criar o gráfico é simplesmente chamada de `plt.plot`:

In [2]:
```
x = np.arange(0, 10, 1)
y = x**2 + 7
plt.plot(x,y)
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x27f617aed68>]
```



Veja que a curva do gráfico ficou com um leve aspecto de "quebrada".

Isso ocorreu porque pedimos um intervalo de 1 unidade entre os pontos.

Vamos suavizar o aspecto da curva pedindo intervalos menores, por exemplo de 0.1 unidade:

In [ ]:
```
x = np.arange(0, 10, 0.1)
y = x**2 + 7
plt.plot(x,y)
```

Podemos usar funções pré-existentes, oferecidas pelo `numpy` .

Por exemplo, temos várias funções trigonométricas como a função para calcular o seno de números:

In [ ]:
```
x = np.arange(0, 10, 0.1)
y = np.sin(x)
plt.plot(x,y)
```

Esse site (https://docs.scipy.org/doc/numpy-1.13.0/reference/ufuncs.html#available-ufuncs) tem uma lista das várias funções que o numpy oferece.

No exemplo a seguir usamos a função de logaritmo.

In [ ]:
```
x = np.arange(1, 100, 0.1)
y = np.log(x)
plt.plot(x,y)
```

Note que a função `arange()` não funciona apenas para números positivos.

A seguir temos o gráfico da função $3x^2 - 5x + 10$:

In [ ]:
```
x = np.arange(-5, 5, 0.1)
y = 3 * x**2 - 5*x + 10
plt.plot(x,y)
```

Podemos ainda personalizar nossos gráficos, colocando nome nos eixos e no próprio gráfico:

In [ ]:
```
plt.xlabel("tempo (s)")
plt.ylabel("sinal")
plt.title("Sinal lido a cada instante de tempo")

x = np.arange(0, 10, 0.1)
y = np.cos(x)
plt.plot(x,y)
```

Às vezes é útil ter uma grade no gráfico.

Com ela podemos ver, por exemplo, em que pontos o gráfico cruza o eixo x (definindo, assim, as *raízes* da função).

A seguir temos a função $2x - 4x^3 + 9$, cuja raiz é, aproximadamente, $x = 1.4372$.

In [ ]:
```
plt.grid(True)
x = np.arange(-5, 5, 0.1)
y = 2*x - 4 * x**3 + 9
plt.plot(x,y)
```

Novamente, é importante você perceber que **é possível fazer muito mais** usando matplotlib e numpy.

O que fizemos acima deve apenas servir de ponto de partida para você.

Por exemplo, apenas plotamos gráficos do tipo *linha*, sendo que existem várias outros tipos de gráficos. Veremos mais sobre isso a seguir.

Gráficos usando bases de dados

Agora vamos criar gráficos a partir de bases de dados.

O uso de gráficos torna a interpretação e/ou análise dos dados mais rápida e objetiva.

Antes de verificar como fazer gráficos com Python, vamos aprender algumas coisas sobre gráficos de maneira geral.

Tipos de gráficos

Existem vários tipos de gráficos, sendo que cada um deles representa melhor um certo tipo de informação.

Compreendê-los e escolher o formato ideal é essencial para realizar a leitura corretamente.

Você pode dar uma olhada [nesse site (http://posgraduando.com/exemplos-de-graficos/)](http://posgraduando.com/exemplos-de-graficos/) depois para ter uma ideia de como *não* usar certos tipos de gráficos.

> As figuras das seções a seguir não são de minha autoria e não foram geradas pelo Python.

Gráfico de linhas

Revela tendências e progressos ao longo do tempo. Você deve usá-los quando estiver traçando um conjunto de dados contínuos.

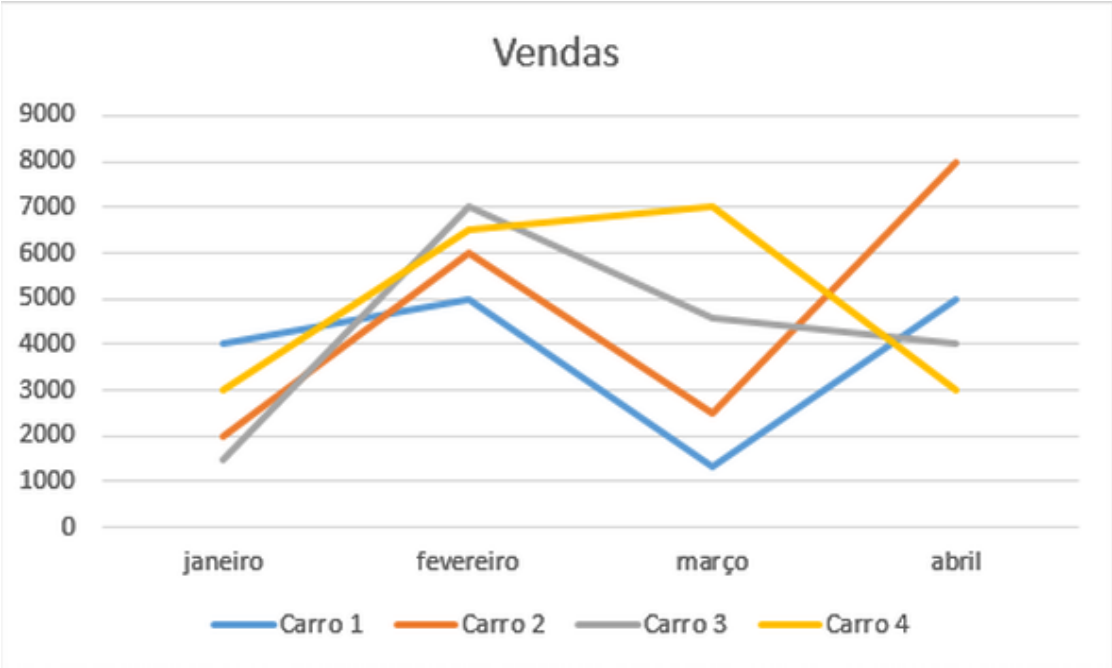
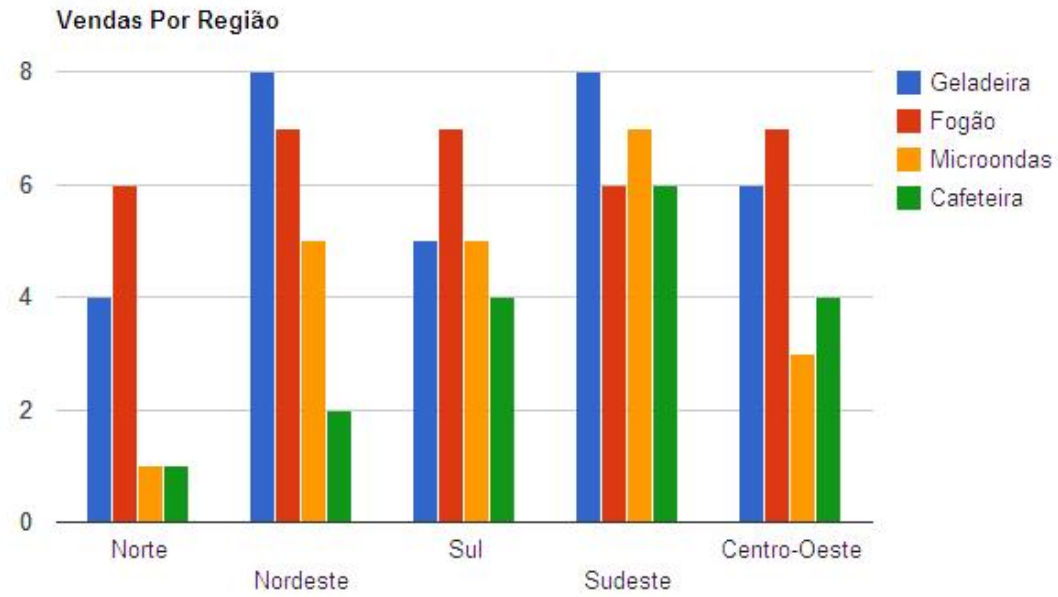


Gráfico de barras

Usado para comparar quantidades. As barras podem aparecer na vertical ou na horizontal e, quanto maior o comprimento de uma barra, maior o valor que representa.



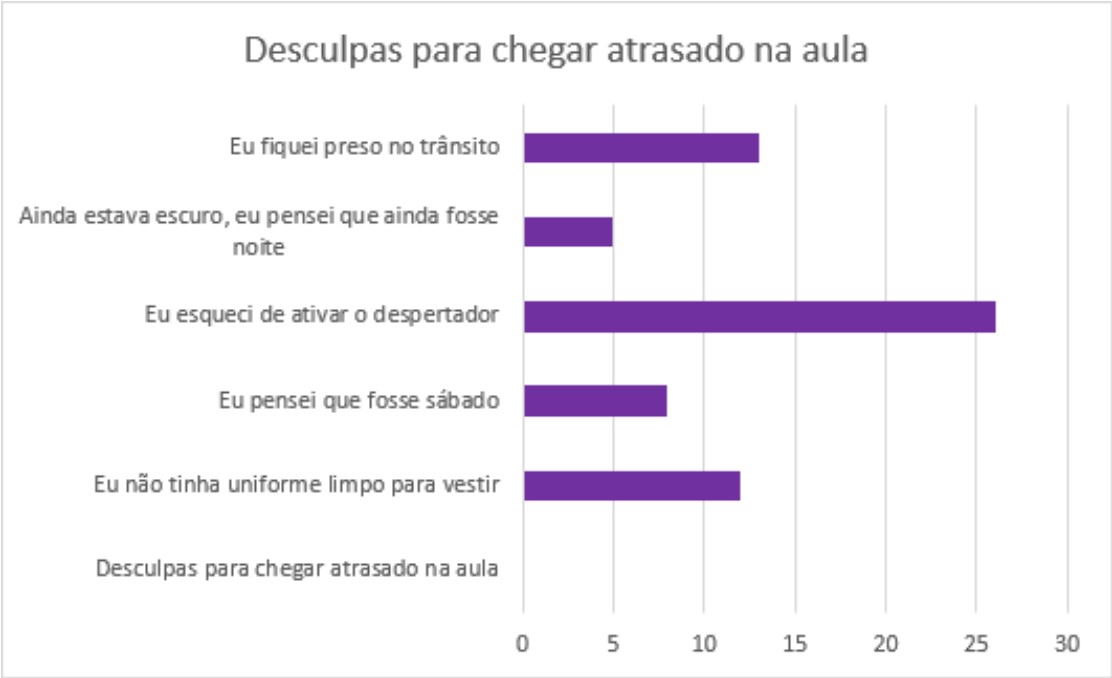
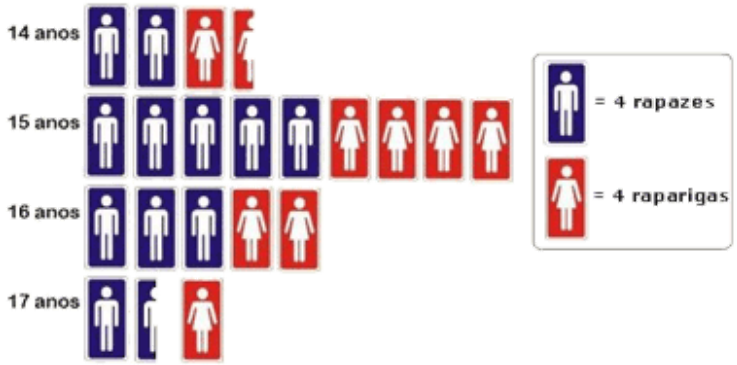


Gráfico de barras empilhadas

Em um gráfico de barras empilhadas, as barras são divididas em segmentos de barra coloridos que são posicionados em cima uns dos outros. A altura total de uma barra mostra o valor numérico de uma determinada categoria e as alturas dos segmentos de barra representam a contribuição de diferentes componentes para aquele valor.



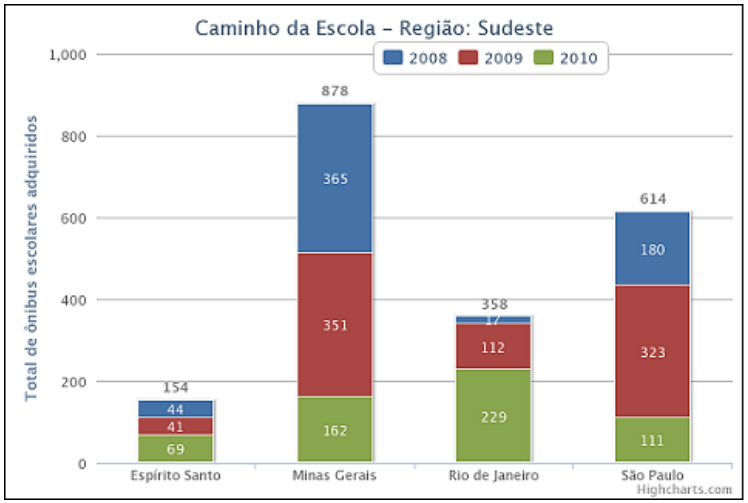
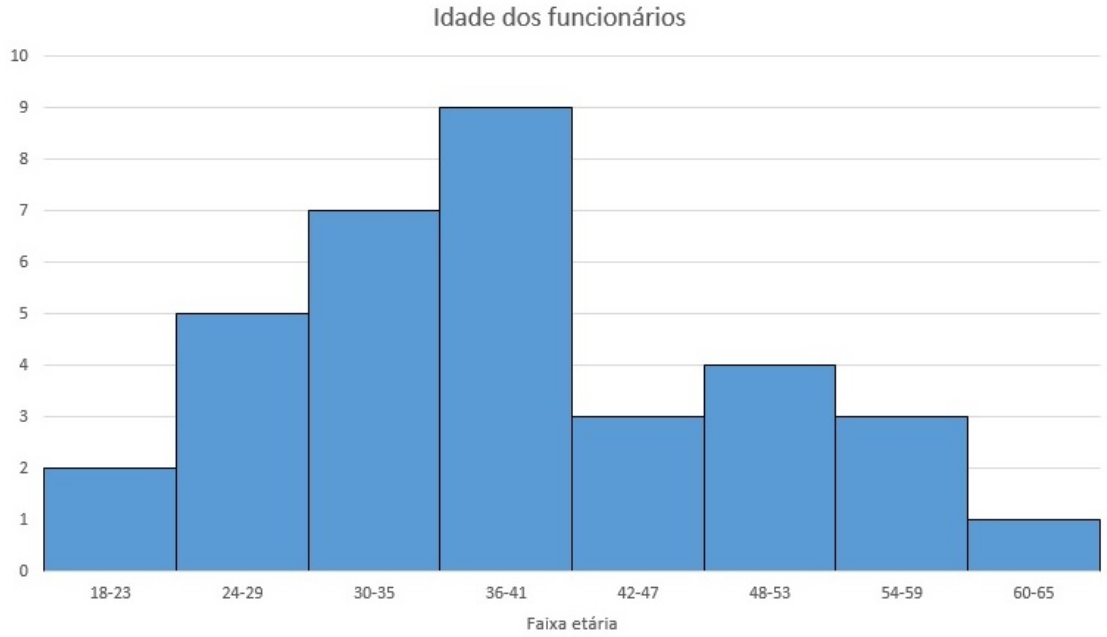


Gráfico de pizza

Serve para mostrar como as categorias representam parte de um todo. A soma total de todas as categorias precisa ser igual a 100%.

Histograma

Utilizado para indicar a distribuição dos dados. Apresenta diversas barras verticais, sendo que base de cada barra representa uma classe e a altura de cada barra representa a quantidade ou a frequência absoluta com que o valor da classe ocorre no conjunto de dados. De certa forma se assemelha ao gráfico de barras, mas em geral não apresenta espaço entre as barras.



Gráficos em Python com o Pandas

Finalmente, vamos aprender como utilizar o Python para criar gráficos a partir de uma base de dados.

Como na aula passada, vamos utilizar o Pandas para nos auxiliar na manipulação das bases de dados.

In [ ]:
```
import pandas as pd
from IPython.display import display
```

Vamos utilizar quatro arquivos nessa aula.

O arquivo `pib-percapita.csv` contém informações sobre o produto interno bruto (PIB) por habitante, para várias cidades brasileiras com dados do ano 2009.

O arquivo `temperatura-SA.csv` contém informações sobre a temperatura, chuva, nebulosidade e duração do sol em Santo André por hora entre os dias 14 e 28 de julho de 2018.

O arquivo `populacao-urbana.csv` contém a população por país entre os anos 3700 A.C. e 2000 D.C.

O arquivo `agua.csv` contém informações sobre abastecimento de água nas residências do Brasil que foram visitadas no Censo de 2010, por município.

In [ ]:
```
pib = pd.read_csv("pib-percapita.csv", sep=";")
temperatura = pd.read_csv("temperatura-SA.csv", sep=";")
populacao = pd.read_csv("populacao-urbana.csv", sep=";")
agua = pd.read_csv("agua.csv", sep=";")
```

In [ ]:
```
display(pib.head())
```

In [ ]:
```
display(temperatura.head())
```

In [ ]:
```
display(populacao.head())
```

In [ ]:
```
display(agua.head())
```

Importante

Lembre-se de que as tabelas são do tipo DataFrame.

Assim, `pib`, `temperatura` e `populacao` são variáveis do tipo DataFrame.

Importante 2

A informação mais importante que você precisa guardar é que o Pandas **compara colunas de um único DataFrame** entre si.

Assim, tanto os valores do eixo x quanto do eixo y do gráfico devem estar armazenados em colunas.

Isso significa que se não houver uma coluna com os valores que você deseja mostrar no gráfico, você deve de alguma forma criar uma coluna nova e utilizá-la.

Importante 3

Esse notebook não vai ter campos de "Faça você mesmo" especificamente.

No entanto, você deve modificar os comandos dos exemplos que serão dados e testar as diferentes possibilidades.

Funções do Pandas para plotagem de gráficos

Vamos precisar de apenas quatro funções para conseguir plotar gráficos usando o Pandas: `plot()`, `hist()`, `groupby()` e `agg()`.

plot()

A função `plot()` pode ser aplicada sobre uma tabela (dataframe) e é a que de fato cria um gráfico. Seus possíveis parâmetros são:

- `x`: para indicar a coluna que será usada no eixo x (se não especificado, o índice da tabela será usado)
- `y`: para indicar a lista de colunas que serão usadas no eixo y
- `kind`: para indicar o tipo do gráfico, que pode ser `line` (linha), `bar` (barra vertical), `barh` (barra horizontal), `pie` (pizza). Existem ainda outros tipos
- `title`: para indicar o título do gráfico
- `grid`: para indicar se deve ou não ser impressa uma grade no gráfico
- `stacked`: para, no caso dos gráficos de barra, indicar se é barra empilhada ou não

- `figsize` : para indicar o tamanho da figura
- `logx` : para indicar se a escala do eixo x deve ser logarítmica
- `logy` : para indicar se a escala do eixo y deve ser logarítmica

Veremos a seguir vários exemplos utilizando essa função.

> *Lembre-se de brincar com os exemplos: por exemplo, trocando o tipo de gráfico ou as colunas que estão sendo utilizadas.*

Exemplo de plot(): Variação da temperatura em Santo André no dia 21 de julho de 2018

Temos no arquivo que foi aberto na variável `temperatura` uma coluna `Hora` e uma coluna `Temperatura` . Vamos compará-las a seguir:

In [ ]:
```
condicao = temperatura["Dia"] == 21
temperatura[condicao].plot(x="Hora", y=["Temperatura"], kind="line", title="Temperatura em SA (21/07/2018)")
```

Podemos perceber que a maior temperatura do dia (pouco mais de 24 graus) aconteceu por volta das 11h.

Exemplo de plot(): Valor do PIB per capita por cidade do Acre, em 2009

In [ ]:
```
condicao = pib["UF"] == "ACRE"
pib[condicao].plot(x="Cidade", y=["PIB percapita"], kind="bar")
```

Note que a cidade Bujari é que teve maior PIB per capita no Acre em 2009, ficando sua capital Rio Branco em segundo lugar.

Faça um teste agora e troque o estado "ACRE" no comando acima por, sei lá, "MINAS GERAIS".

Como são muitas as cidades, o gráfico vai ficar ilegível!

Exemplo de plot(): Valor do PIB e do PIB per capita por cidade do Acre, em 2009

Como as colunas PIB e PIB per capita se referem a coisas com mesma unidade de medida, faz sentido usar um mesmo gráfico para observá-las:

In [ ]:
```
condicao = pib["UF"] == "ACRE"
pib[condicao].plot(x="Cidade", y=["PIB", "PIB percapita"], kind="bar")
```

Mas note que agora o grafo ficou ilegível também (não dá para ver as colunas de PIB per capita).

Podemos transformar a escala do gráfico para logarítmica:

In [ ]:
```
condicao = pib["UF"] == "ACRE"
pib[condicao].plot(x="Cidade", y=["PIB", "PIB percapita"], kind="bar", logy=True)
```

Como se explica o fato de Rio Branco ter o maior PIB e ao mesmo tempo não ter o maior PIB per capita (e sim o segundo maior PIB per capita)? E Bujari, que nem tem o segundo maior PIB?

In [ ]:
```
condicao = (pib["UF"] == "ACRE") & ((pib["Cidade"] == "Rio Branco") | (pib["Cidade"] == "Bujari"))
display(pib[condicao])
```

Exemplo de plot(): Cidades que concentravam a população do Japão no ano de 1500

In [ ]:
```
condicao = (populacao["País"] == "Japan") & (populacao["Ano"] == 1500)
populacao[condicao].plot(x="Cidade", y="População", kind="pie")
```

Ao plotar um gráfico de pizza, o Pandas automaticamente considera que os nomes de cada parte "da pizza" são os índices. Por isso temos esses números sendo mostrados na legenda do gráfico acima.

Para arrumar isso, vamos indicar que o índice deve ser o nome das cidades:

In [ ]:
```
condicao = (populacao["País"] == "Japan") & (populacao["Ano"] == 1500)
# vamos criar um dataframe novo, para não alterar nada do anterior:
populacao2 = populacao[condicao]
populacao2.set_index("Cidade", inplace=True)
populacao2.plot(y="População", kind="pie")
```

Para fins estéticos, vamos tirar a legenda de cima do gráfico (usando a função `legend` abaixo) e também vamos alterar o tamanho do gráfico com o parâmetro `figsize`:

In [ ]:
```
condicao = (populacao["País"] == "Japan") & (populacao["Ano"] == 1500)
populacao2 = populacao[condicao]
populacao2.set_index("Cidade", inplace=True)
populacao2.plot(y="População", kind="pie", figsize=(5,5)).legend(bbox_to_anchor=(1.5, 1))
```

Exemplo de plot(): Residências do estado de Roraima e o recebimento de água

Do total de residências visitadas pelo Censo em cada município brasileiro, algumas possuem água em pelo menos um cômodo, outras possuem água no terreno ou propriedade e outras nem possuem água.

Vamos visualizar esses dados na forma de grafo de barras empilhadas (pois cada barra irá conter o total de residências daquele município).

In [ ]:
```
condicao = agua["Estado"] == "Roraima"
agua[condicao].plot(x="Município", y=["Água em pelo menos um cômodo", "Água no terreno ou propriedade", "Não tinham"],
                    kind="bar", stacked=True, figsize=(15,5))
```

Vamos omitir a cidade de Boa Vista, que teve um número muito maior de residências visitadas, para poder observar melhor o que acontece com os outros municípios:

In [ ]:
```
condicao = (agua["Estado"] == "Roraima") & (agua["Município"] != "Boa Vista")
agua[condicao].plot(x="Município", y=["Água em pelo menos um cômodo", "Água no terreno ou propriedade", "Não tinham"],
                    kind="bar", stacked=True, figsize=(15,5))
```

Exemplo de plot(): Variação da temperatura em SA entre os dias 20 e 25 de julho

Até agora vimos exemplos bem simples que plotam dados sobre uma única cidade, um único dia ou um único país.

Mas e se quisermos comparar dados entre vários dias, várias cidades ou vários países?

Vamos filtrar o dataframe `temperatura` para pegar os dados de temperatura entre os dias 20 e 25 de julho em Santo André e tentar plotar algum gráfico com isso:

In [ ]:
```
condicao = (temperatura["Dia"] >= 20) & (temperatura["Dia"] <= 25)
temperatura[condicao].plot(x="Hora", y="Temperatura", kind="line")
```

Veja como o gráfico acima não faz sentido nenhum.

Isso acontece porque o que o `plot()` faz é simplesmente colocar no gráfico os dados de todas as linhas de uma vez.

Quando tínhamos só o dia 21, só tinha um dado de temperatura para o horário das 10h, por exemplo. Agora temos 6 dados de temperatura para o horário das 10h (dos dias 20, 21, 22, 23, 24 e 25).

Tente modificar o tipo do gráfico para ver se o problema é resolvido.

É para resolver esse tipo de problema que vamos também estudar as funções `groupby()` e `agg()` mencionadas anteriormente.

hist()

A função `hist()` pode ser aplicada sobre uma tabela (dataframe) e é específica para criar histogramas.

Ela recebe uma lista de colunas que armazenam os dados sobre os quais queremos criar a distribuição.

Vejamos um exemplo a seguir.

Exemplo de hist(): Distribuição da população do estado de São Paulo em 2009

Vamos criar um histograma para verificar a distribuição da população em São Paulo no ano de 2009.

Cada classe (barra) do histograma vai indicar um intervalo de quantidade de pessoas e o tamanho da barra vai indicar quantas cidades estão naquela classe:

In [ ]:
```
condicao = pib["UF"] == "SÃO PAULO"
pib[condicao].hist("População (2009)")
```

Veja que o gráfico acima não informa nada: a capital de São Paulo tem *muito* mais habitantes do que as outras cidades, ficando sozinha na última barra (que começa no valor 1.0×10^7) enquanto que praticamente todas as outras cidades estão aglomeradas na primeira barra.

Para consertar isso, normalizamos os valores aplicando a função de logarítmo (na base 10) sobre os dados da população.

Vamos criar uma nova coluna para armazenar esse valor para poder utilizá-la depois na plotagem:

In [ ]:
```
pib['População (log10)'] = np.log10(pib['População (2009)'])
display(pib.head())
```

Agora vamos criar o histograma com essa nova coluna:

In [ ]:
```
condicao = pib["UF"] == "SÃO PAULO"
pib[condicao].hist("População (log10)")
```

A informação que temos do gráfico acima é que, por exemplo, entre 20 e 30 cidades de São Paulo têm população entre aproximadamente $10^{2.8}$ e $10^{3.35}$ habitantes (primeira barra). A maioria das cidades (cerca de 450) de São Paulo têm população entre aproximadamente $10^{3.35}$ e $10^{4.55}$ habitantes (segunda a quarta barras). Uma ou duas cidades (provavelmente apenas a capital) têm população acima de $10^{6.5}$ habitantes.

Verificando:

In [ ]:
```
condicao = (pib["UF"] == "SÃO PAULO") & (pib["População (2009)"] >= 10**2.8) & (pib["População (2009)"] <= 10**3.35)
print(pib[condicao].shape)

condicao = (pib["UF"] == "SÃO PAULO") & (pib["População (2009)"] >= 10**3.35) & (pib["População (2009)"] <= 10**4.55)
print(pib[condicao].shape)

condicao = (pib["UF"] == "SÃO PAULO") & (pib["População (2009)"] >= 10**6.5)
print(pib[condicao].shape)
display(pib[condicao])
```

groupby()

A função `groupby()` pode ser aplicada sobre uma tabela (dataframe) e ela vai separar os dados em grupos baseados em algum critério.

Quando aplicada, ela não cria um novo dataframe, mas de certa forma cria vários, um para cada grupo.

Ela recebe como parâmetro apenas uma coluna ou lista de colunas e vai criar grupos de acordo com valores que se repetem nessas colunas.

Vamos ver alguns exemplos a seguir.

Exemplo de groupby(): Dados da temperatura em SA entre os dias 14 e 28 de julho

A tabela `temperatura` contém dados de hora em hora para todos os dias entre 14 e 28 de julho, em Santo André.

Podemos agrupar os dados da tabela por dia:

In [ ]:
```
temperatura_agrupado = temperatura.groupby("Dia")
type(temperatura_agrupado)
```

Quando usamos a função `groupby()` sobre uma tabela, ela não cria uma nova tabela (dataframe). Ela cria um dado diferente.

Como foi criado um grupo para cada dia, podemos visualizar cada grupo como uma tabela diferente:

In [ ]:
```
# visualizando apenas os dados do grupo do dia 21 de julho:
display(temperatura_agrupado.get_group(21))
```

Mas nós já sabíamos como ver apenas os dados do dia 21 de julho:

In [ ]:
```
condicao = temperatura["Dia"] == 21
display(temperatura[condicao])
```

Então para que serviu o agrupamento?

É que sobre os dados agrupados (e não apenas sobre um grupo), podemos aplicar funções para obter uma única tabela.

Para isso vamos precisar da última função que eu mencionei, a função `agg()`.

agg()

A função `agg()` serve para transformar os dados de uma coluna inteira em um único dado, de acordo com uma ou mais funções (por exemplo, `agg("max")` gera o maior valor da coluna, `agg("mean")` gera a média da coluna, etc.).

Ela pode ser aplicada sobre um dataframe ou então sobre os grupos criados pela `groupby()`.

Vejamos alguns exemplos para seu funcionamento ficar mais claro.

In [ ]:
```
# Vamos aplicar a função agg() sobre um dataframe primeiro:
display(temperatura.agg("max"))
```

Veja que ela criou uma única coluna, que tem o maior valor de cada coluna do dataframe `temperatura`. Por exemplo, o maior "Dia" é 28 porque o dataframe só tem dados entre os dias 14 e 28.

Agora, essa informação não parece tão útil.

Vamos então aplicá-la sobre os grupos que criamos acima:

In [ ]:
```
temperatura_agrupado = temperatura.groupby("Dia")
display(temperatura_agrupado.agg("max"))
```

Agora sim, para cada dia (veja a coluna mais à esquerda) temos apenas o valor máximo de cada uma das colunas.

Agora que temos uma tabela, podemos plotar um gráfico que faça sentido:

In [ ]:
```
temperatura_agrupado = temperatura.groupby("Dia")
temperatura_agrupado.agg("max").plot(y=["Temperatura"], kind="line", title="Temperatura máxima de 14 a 28 de julho")
```

De onde vemos que no dia 20 de julho tivemos a maior temperatura do período (cerca de 27 graus).

Podemos usar uma lista de funções (e não apenas "max"):

In [ ]:
```
temperatura_agrupado = temperatura.groupby("Dia")
display(temperatura_agrupado.agg(["max", "mean", "min"]))
```

In [ ]:
```
temperatura_agrupado = temperatura.groupby("Dia")
temperatura_agrupado.agg(["max", "mean", "min"]).plot(y="Temperatura", kind="line")
```

No gráfico acima vemos que a temperatura mínima do período aconteceu no dia 18 de julho, sendo próxima a 12 graus, e que a amplitude nesse dia foi bem alta, já que a temperatura máxima foi de 26 graus (ou seja, uma variação de 14 graus no mesmo dia).

Principais funções para usar na função `agg()`: `max`, `mean`, `min`, `sum`.

Exemplos

Agora que já vimos as quatro funções que vamos precisar para criar gráficos com o pandas (`plot()` , `hist()` , `groupby()` e `agg()`), vamos ver a seguir vários exemplos de uso delas para realmente entender suas aplicações.

Estude cada exemplo com cuidado e crie suas próprias variações.

Vamos verificar quantas horas de sol ao todo cada dia de julho teve:

In [ ]:
```
# Primeiro agrupamos a tabela por dia e então agregamos os dados somando-os
temperatura_agrupada = temperatura.groupby("Dia").agg("sum")
# Assim, a coluna "Duração do sol (min)" tem o total de minutos de sol que o dia teve
# Então criamos uma coluna nova, para calcular esse mesmo tempo só que em horas:
temperatura_agrupada["Luminosidade (h)"] = temperatura_agrupada["Duração do sol (min)"]/60
# E plotamos o gráfico de barras:
temperatura_agrupada.plot(y="Luminosidade (h)", kind="bar")
```

A luminosidade foi zero nos dias 22 e 24 e bem perto disso no dia 25. O sol não nasceu nesses dias?

Provavelmente, a nebulosidade estava alta nesses dias:

In [ ]:
```
condicao = (temperatura["Dia"] == 22) | (temperatura["Dia"] == 24) | (temperatura["Dia"] == 25)
display(temperatura[condicao].groupby("Dia").agg("mean"))
```

Agora vamos verificar o PIB per capita por estado brasileiro:

In [ ]:
```
pib_agrupado = pib.groupby("UF").agg(["max", "mean", "min"])
pib_agrupado.plot(y="PIB percapita", kind="bar", figsize=(10,5))
```

Em qual estado você escolheria morar?

Como o valor máximo de PIB per capita de alguns estados está muito acima do normal, vamos verificar só os estados que tenham esse valor abaixo de 80000:

In [ ]:
```
pib_agrupado = pib.groupby("UF").agg(["max", "mean", "min"])
condicao = pib_agrupado["PIB percapita", "max"] < 80000
pib_agrupado[condicao].plot(y="PIB percapita", kind="bar", figsize=(10,5))
```

Vamos verificar a seguir a distribuição de população por estado brasileiro em 2009.

Primeiro agrupamos os dados por estados e fazemos a soma dos valores para poder obter o total da população por estado.

Em seguida aplicamos a função para plotar o histograma:

In [ ]:
```
pib.groupby("UF").agg("sum").hist("População (2009)")
```

Veja que apenas um estado tem população maior do que 3.5×10^7. Deve ser São Paulo, certo?

In [ ]:
```
pib_agrupado = pib.groupby("UF").agg("sum")
condicao = pib_agrupado["População (2009)"] > 3.5 * 10**7
display(pib_agrupado[condicao])
```

E quais são os 5 estados que estão na segunda faixa?

In [ ]:
```
pib_agrupado = pib.groupby("UF").agg("sum")
condicao = (pib_agrupado["População (2009)"] >= 0.4 * 10**7) & (pib_agrupado["População (2009)"] <= 0.88 * 10**7)
display(pib_agrupado[condicao])
```

In [ ]:
```
agua_agrupada = agua.groupby("Estado").agg("sum")
agua_agrupada.plot(y=["Água em pelo menos um cômodo", "Água no terreno ou propriedade", "Não tinham"],
                   kind="bar", stacked=True, figsize=(15,5))
```

Vamos remover os estados do Sudeste da plotagem para visualizar os outros dados melhor:

In [ ]:
```
condicao = (agua["Estado"] != "São Paulo") & (agua["Estado"] != "Rio de Janeiro") & (agua["Estado"] != "Minas Gerais") &
agua_agrupada = agua[condicao].groupby("Estado").agg("sum")
agua_agrupada.plot(y=["Água em pelo menos um cômodo", "Água no terreno ou propriedade", "Não tinham"],
                   kind="bar", stacked=True, figsize=(15,5)).legend(bbox_to_anchor=(1.2, 1))
```

Claramente, o Pará é o estado que possui mais residências sem água encanada dentre as residências visitadas pelo Censo.