> Uma parte fundamental do kit de ferramentas de um(a) cientista de dados é a visualização de dados. Embora seja muito fácil criar visualizações, o mesmo não pode ser dito quanto a produção de **boas** visualizações.

Visualizando Dados

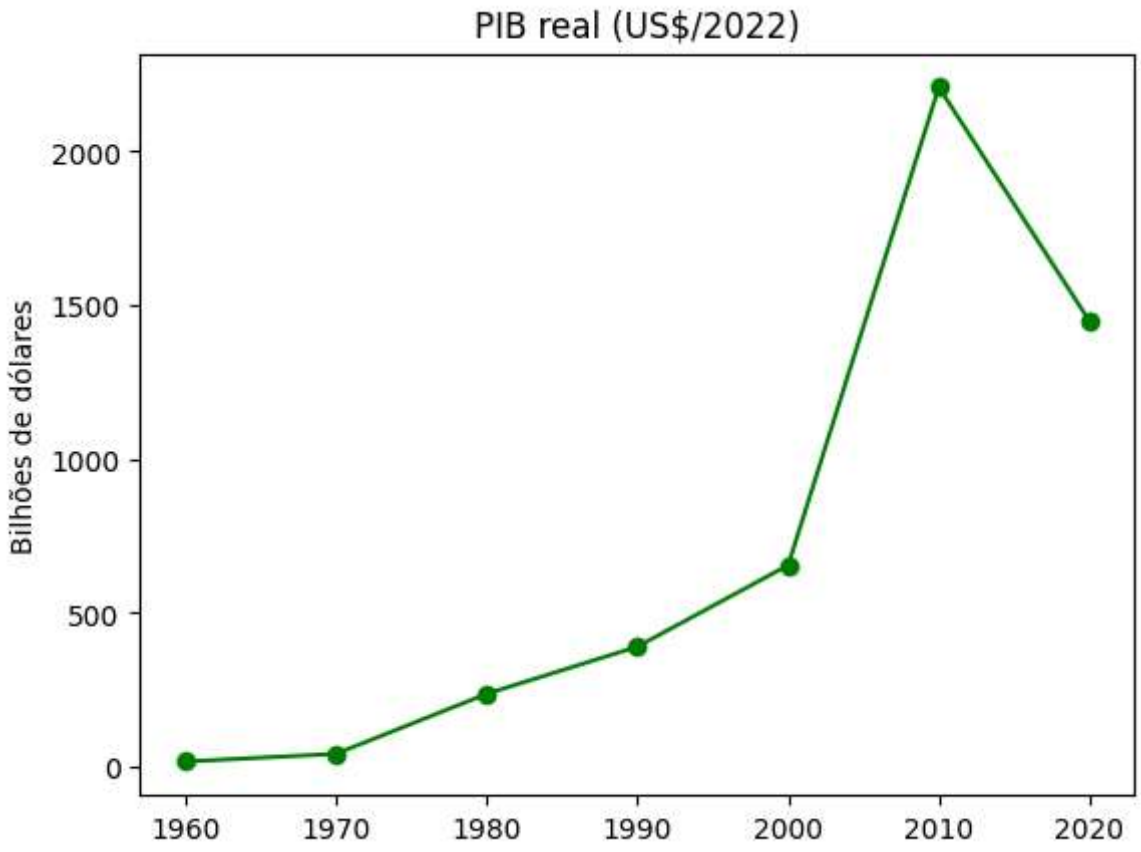
Existem dois usos principais da visualização de dados:

- Exploração de dados;
- Comunicação de dados.

Nesta aula, nos concentraremos em desenvolver as habilidades necessárias para começar a explorar nossos próprios dados e produzir as visualizações que usaremos no restante do curso.

Existe uma grande variedade de ferramentas de visualização de dados. Utilizaremos a amplamente utilizada biblioteca `matplotlib`. Ela é útil para a criação de simples gráficos de barras, linhas ou de dispersão. Por exemplo, fazer gráficos é fácil, como mostra a figura abaixo:

In [ ]:
```
from matplotlib import pyplot as plt

anos = [1960, 1970, 1980, 1990, 2000, 2010, 2020]
pib = [17.03, 42.33, 237.39, 390.73, 655.45, 2208.84, 1448.57]

# Cria um grafico de linha,
# 'anos' no eixo horizontal
# 'pib' no eixo vertical
plt.plot(anos, pib,
         color='green',     # cor = 'verde'
         marker='o',        # marcador dos pontos = 'o'
         linestyle='solid') # estilo da linha = 'solido' (ou continuo)

# Adiciona um titulo
plt.title("PIB real (US$/2022)")

# Adiciona um rotulo a eixo y
plt.ylabel("Bilhões de dólares")

# Exibe o grafico
plt.show()
```

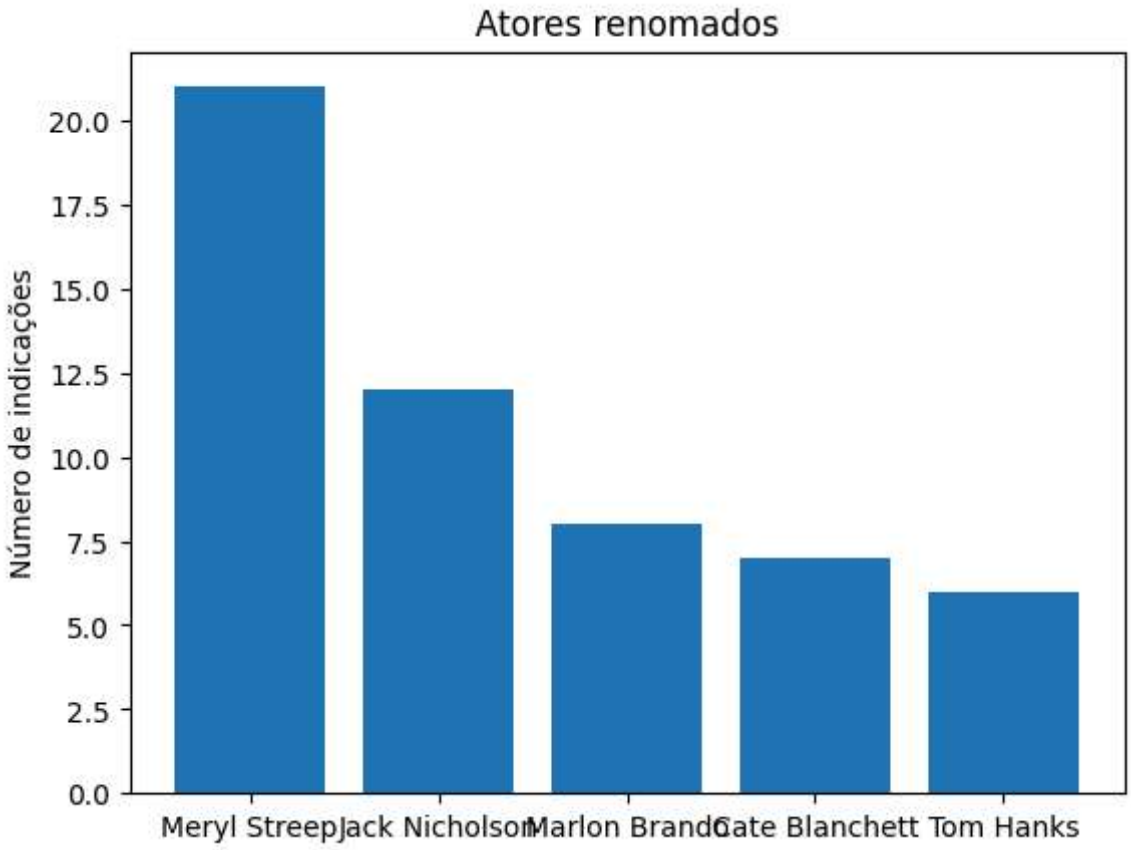


Há muitas maneiras de personalizar seus gráficos com, por exemplo, rótulos dos eixos, estilos de linha e marcadores de ponto. Iremos abordar algumas delas em nossos exemplos práticos.

Gráfico de Barras

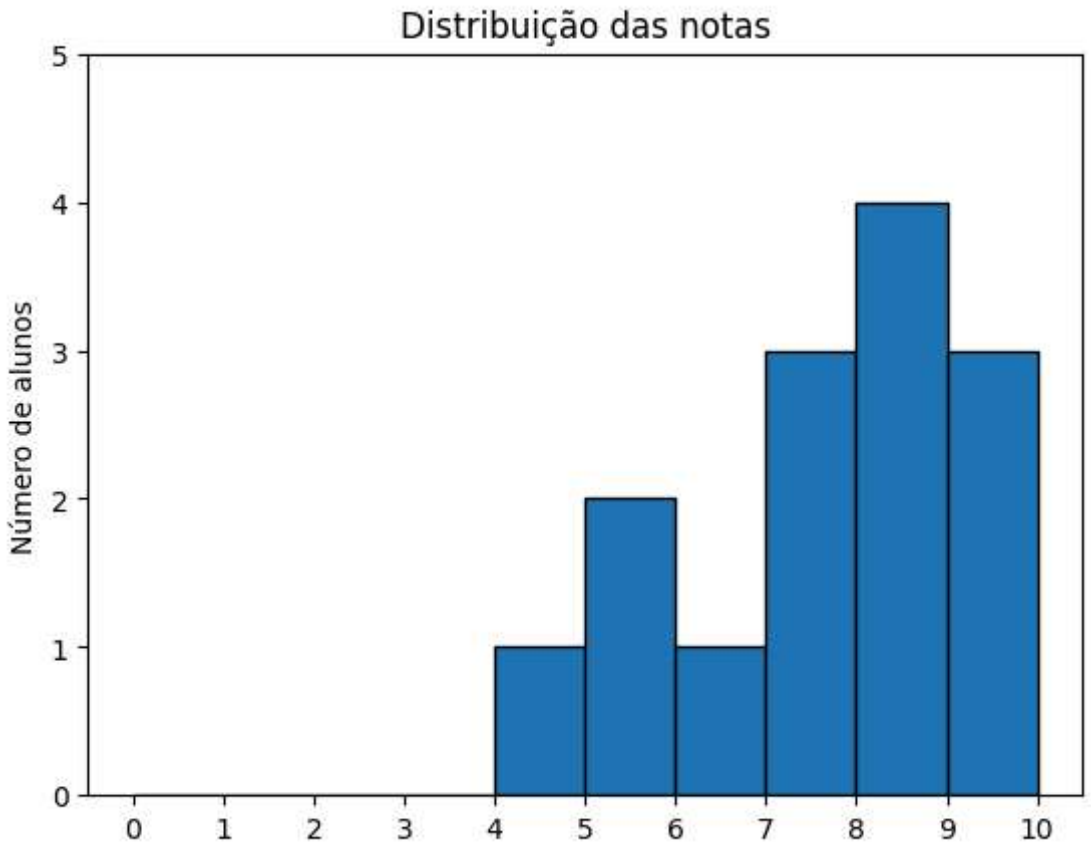
Um gráfico de barras é uma boa opção quando você deseja mostrar como alguma quantidade varia entre um conjunto discreto de itens. Por exemplo, a figura abaixo mostra quantas vezes alguns atores foram indicados ao Oscar:

In [ ]:
```
atores = ["Meryl Streep", "Jack Nicholson", "Marlon Brando", "Cate Blanch
num_indicacoes = [21, 12, 8, 7, 6]

# Cria um grafico de barras,
# com cordenadas horizontais [0, 1, 2, 3, 4]
# e alturas = 'num_indicacoes'
plt.bar(range(len(num_indicacoes)), num_indicacoes)

plt.title("Atores renomados")      # adiciona um titulo
plt.ylabel("Número de indicações") # rotula o eixo vertical

# rotula o eixo horizontal com os nomes dos atores
# no centro das barras
plt.xticks(range(len(atores)), atores)

plt.show()
```



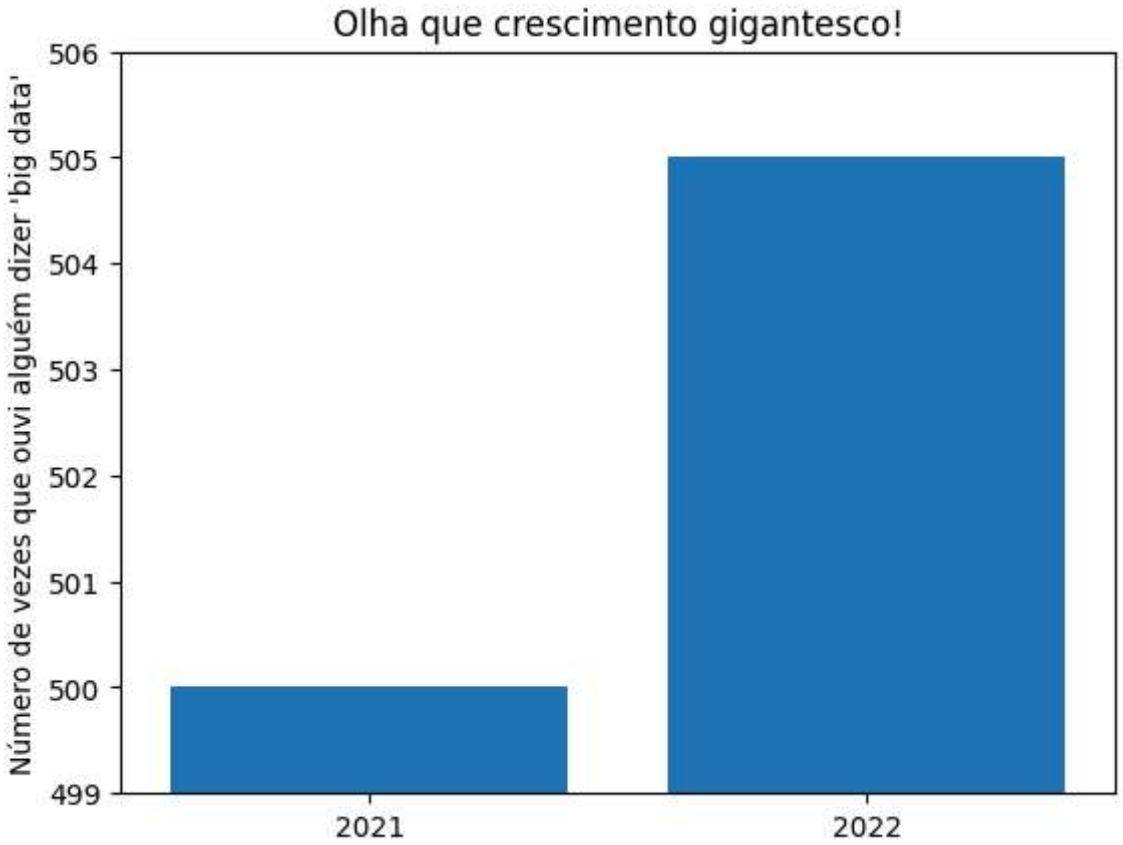
Um gráfico de barras também pode ser uma boa opção para plotar histogramas de valores numéricos agrupados para explorar visualmente como os valores são distribuídos:

In [ ]:
```
notas = [8.3, 9.5, 9.1, 8.7, 7.0, 5.7, 8.5, 8.2, 10, 6.7, 7.3, 7.7, 4.8,

plt.hist(notas, bins = range(11), edgecolor=(0, 0, 0))

plt.axis([-.5, 10.5, 0, 5]) # eixo horizontal de -.5 ate 10.5,
                            # eixo vertical de 0 ate 5

plt.xticks(range(11)) # rotulos do eixo horizontal em 0, 1, ..., 10
plt.ylabel("Número de alunos")
plt.title("Distribuição das notas")
plt.show()
```



Seja criterioso ao usar `plt.axis`. Ao criar gráficos de barras, não é recomendado o eixo vertical iniciar em algum valor diferente de 0, pois esta é uma das formas de ludibriar as pessoas:

In [ ]:
```
mencoes = [500, 505]
anos = [2021, 2022]

plt.bar(anos, mencoes, 0.8)
plt.xticks(anos)
plt.ylabel("Número de vezes que ouvi alguém dizer 'big data'")

# Eixo vertical enganoso partindo de 499
plt.axis([2020.5, 2022.5, 499, 506])
plt.title("Olha que crescimento gigantesco!")
plt.show()
```



Na figura a seguir, usamos eixos mais condizentes e o crescimento parece muito menos impressionante:

In [ ]:
```
mencoes = [500, 505]
anos = [2021, 2022]

plt.bar(anos, mencoes, 0.8)
plt.xticks(anos)
plt.ylabel("Número de vezes que ouvi alguém dizer 'big data'")

# Eixo vertical enganoso partindo de 0
plt.axis([2020.5, 2022.5, 0, 550])
plt.title("Crescimento não tão gigantesco assim")
plt.show()
```

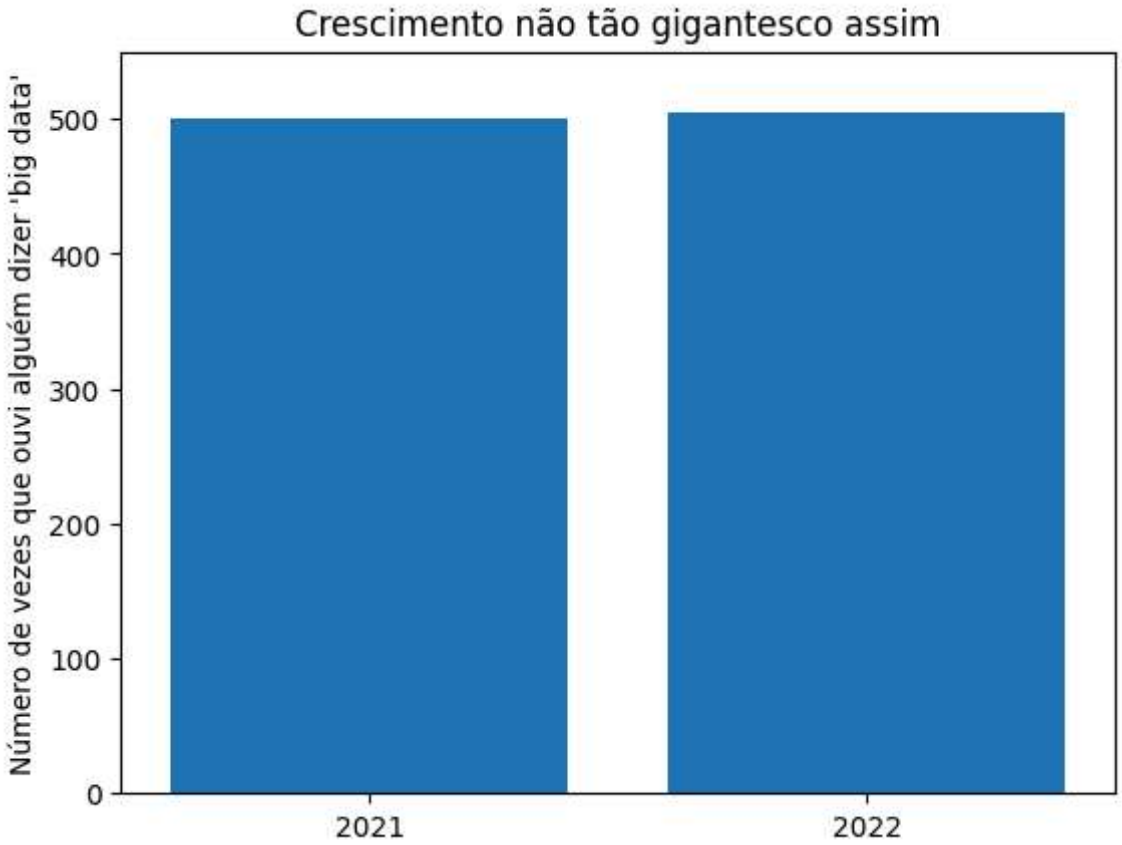
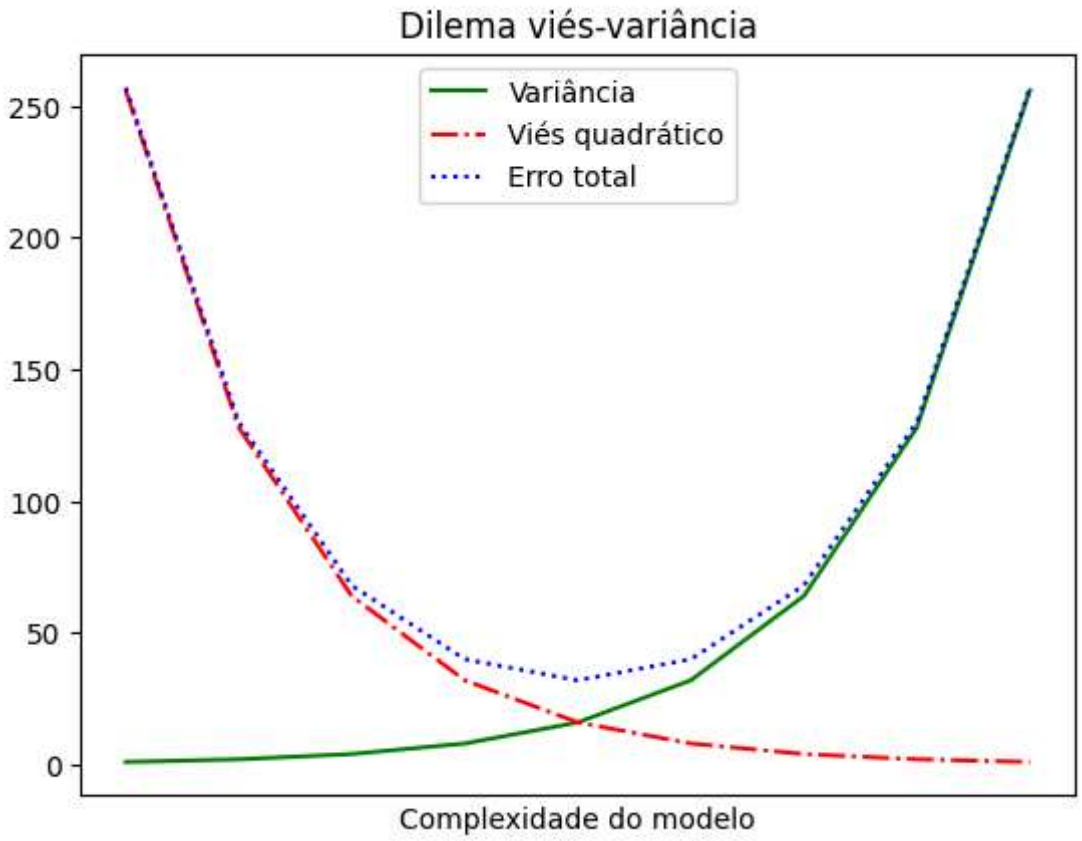


Gráfico de Linhas

Como já visto, podemos fazer gráficos de linhas usando `plt.plot`. Eles são uma boa opção para mostrar tendências, conforme ilustrado abaixo:

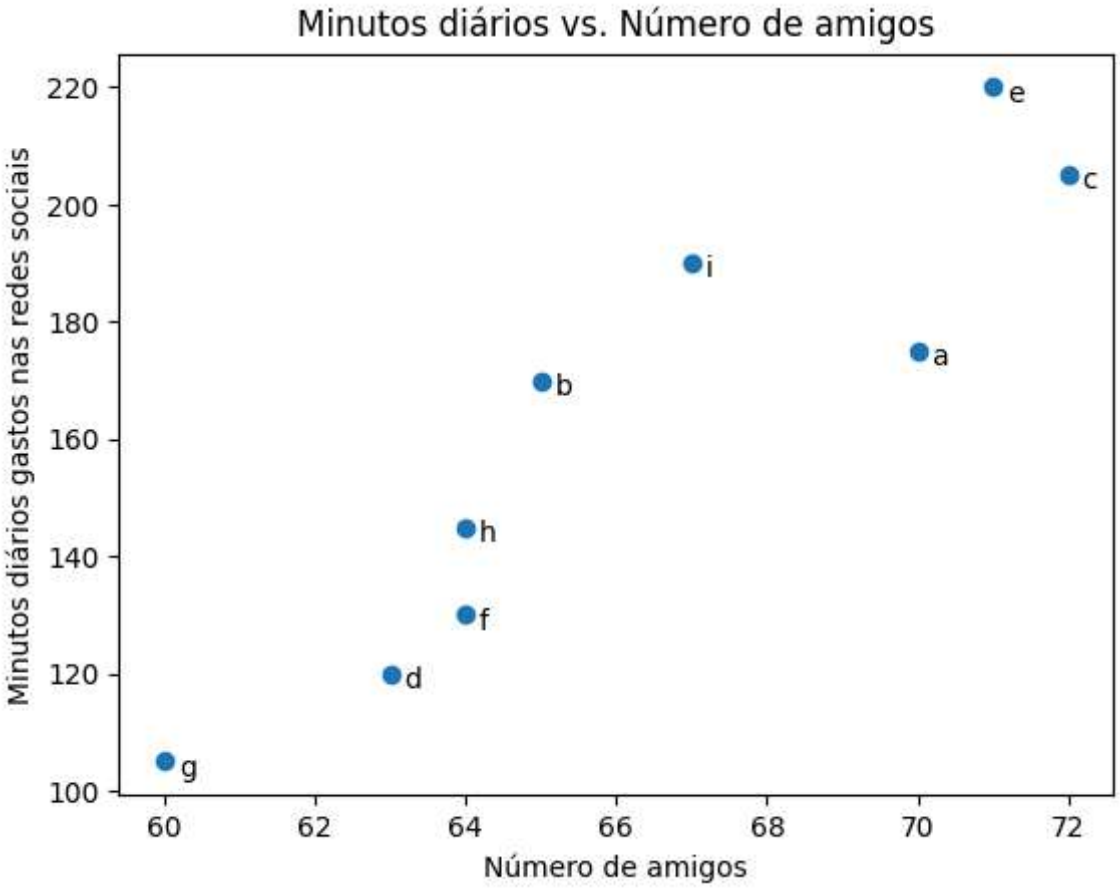
In [ ]:
```
variancia = [1, 2, 4, 8, 16, 32, 64, 128, 256]
vies_quadratico = [256, 128, 64, 32, 16, 8, 4, 2, 1]
erro_total = [x + y for x, y in zip(variancia, vies_quadratico)]

# Podemos utilizar plt.plot varias vezes
# para mostrar multiplas series em um mesmo grafico
plt.plot(variancia, 'g-', label='Variância')     # linha verde continua
plt.plot(vies_quadratico, 'r-.', label='Viés quadrático') # linha vermelh
plt.plot(erro_total, 'b:', label='Erro total')  # linha azul tracejada

# Como atribuímos rotulos a cada serie,
# inserimos uma legenda (loc=9 significa "top center")
plt.legend(loc=9)
plt.xlabel("Complexidade do modelo")
plt.xticks([])
plt.title("Dilema viés-variância")
plt.show()
```



Gráficos de Dispersão

Um gráfico de dispersão é a escolha correta para visualizar a relação entre dois conjuntos de dados pareados. Por exemplo, a figura abaixo ilustra a relação entre o número de amigos que seus usuários têm e o número de minutos que eles passam nas redes sociais todos os dias:

In [ ]:
```
amigos = [70, 65, 72, 63, 71, 64, 60, 64, 67]
minutos = [175, 170, 205, 120, 220, 130, 105, 145, 190]
rotulos = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

plt.scatter(amigos, minutos)

# rotulando cada ponto
for rotulo, contagem_amigos, contagem_minutos in zip(rotulos, amigos, min
    plt.annotate(rotulo,
                 # rotula cada ponto
                 xy=(contagem_amigos, contagem_minutos),
                 xytext=(5, -5),  # ligeiramente deslocado
                 textcoords='offset points')

plt.title("Minutos diários vs. Número de amigos")
plt.xlabel("Número de amigos")
plt.ylabel("Minutos diários gastos nas redes sociais")
plt.show()
```



Explorando Dados

Vamos recuperar os exemplos de obtenção de dados feitos na aula anterior e visualizar dados reais.

Explorando Dados Unidimensionais

Com os dados sobre Estados disponibilizados pela API do IBGE, vamos demonstrar algumas informações através de gráficos:

In [ ]:
```
!pip install ibge
```

Feito isso, encontramos os dados sobre os Estados brasileiros da seguinte forma:

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from ibge.localidades import *

estados = Estados()
df_estados = pd.json_normalize(estados.json())
```

Vamos obter a quantidade de Estados em cada região do país:

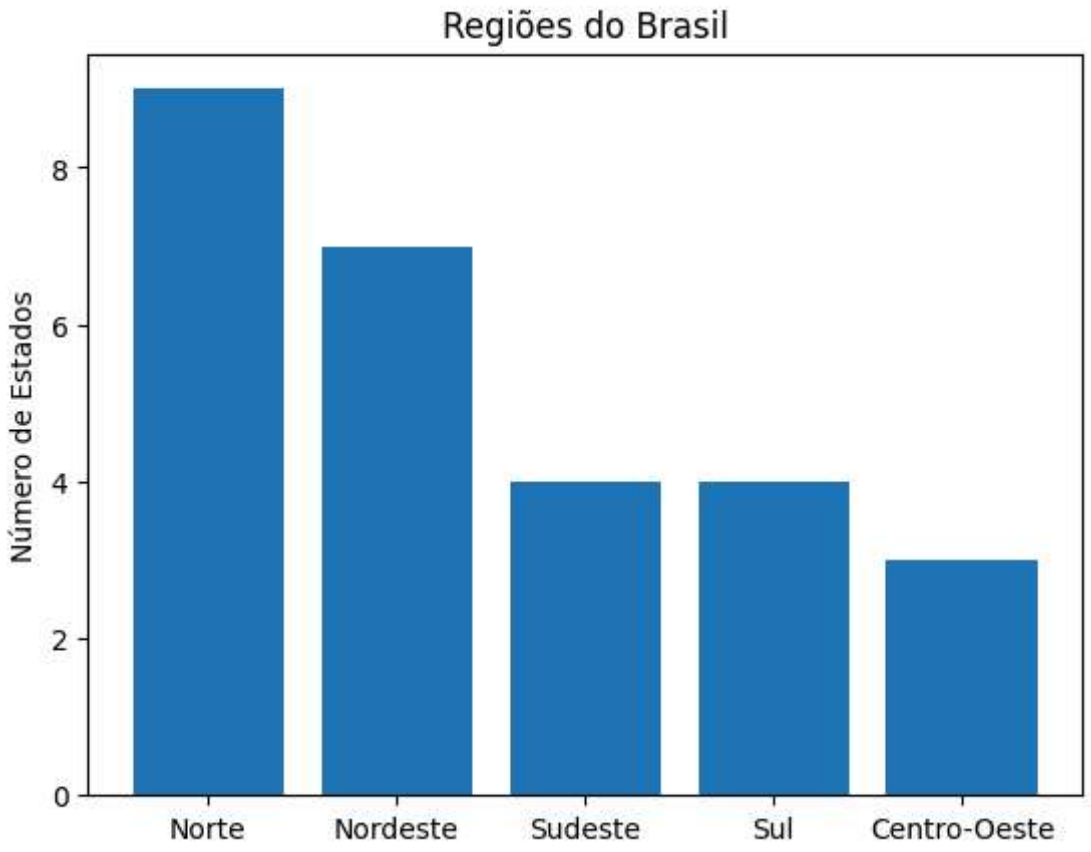
In [ ]:
```
# quantidade de regioes
df_estados["regiao.sigla"].nunique()
```

```
# nome de cada regiao
df_estados["regiao.nome"].unique()

# numero de vezes que cada regiao aparece
# ou seja, o numero de Estados em cada regiao
print(df_estados["regiao.nome"].value_counts())
```

```
regiao.nome
Nordeste        9
Norte           7
Sudeste         4
Centro-Oeste    4
Sul             3
Name: count, dtype: int64
```

Com as informações dos Estados, vamos analisar graficamente a quantidade em cada região:

In [ ]:
```
regioes = ['Nordeste', 'Norte', 'Sudeste', 'Centro-Oeste', 'Sul']
num_estados = [9, 7, 4, 4, 3]

# Cria um grafico de barras,
# com cordenadas horizontais [0, 1, 2, 3, 4]
# e alturas = 'num_estados'
plt.bar(range(len(num_estados)), num_estados)

plt.title("Regiões do Brasil")      # adiciona um titulo
plt.ylabel("Número de Estados")     # rotula o eixo vertical

# rotula o eixo horizontal com os nomes dos atores
# no centro das barras
plt.xticks(range(len(regioes)), regioes)

plt.show()
```



Fica claro que certas regiões têm mais Estados que as outras!

> Embora o Distrito Federal não seja um Estado, estamos o considerando como só para fins didáticos!

Apesar do trabalho manual que tivemos neste exemplo. Certas análises gráficas já estão embutidas na biblioteca `Pandas`. Vamos recuperar os dados sobre o valor *real* do salário mínimo como vimos em aulas anteriores. Lembrando que também precisamos instalar essa biblioteca virtualmente:

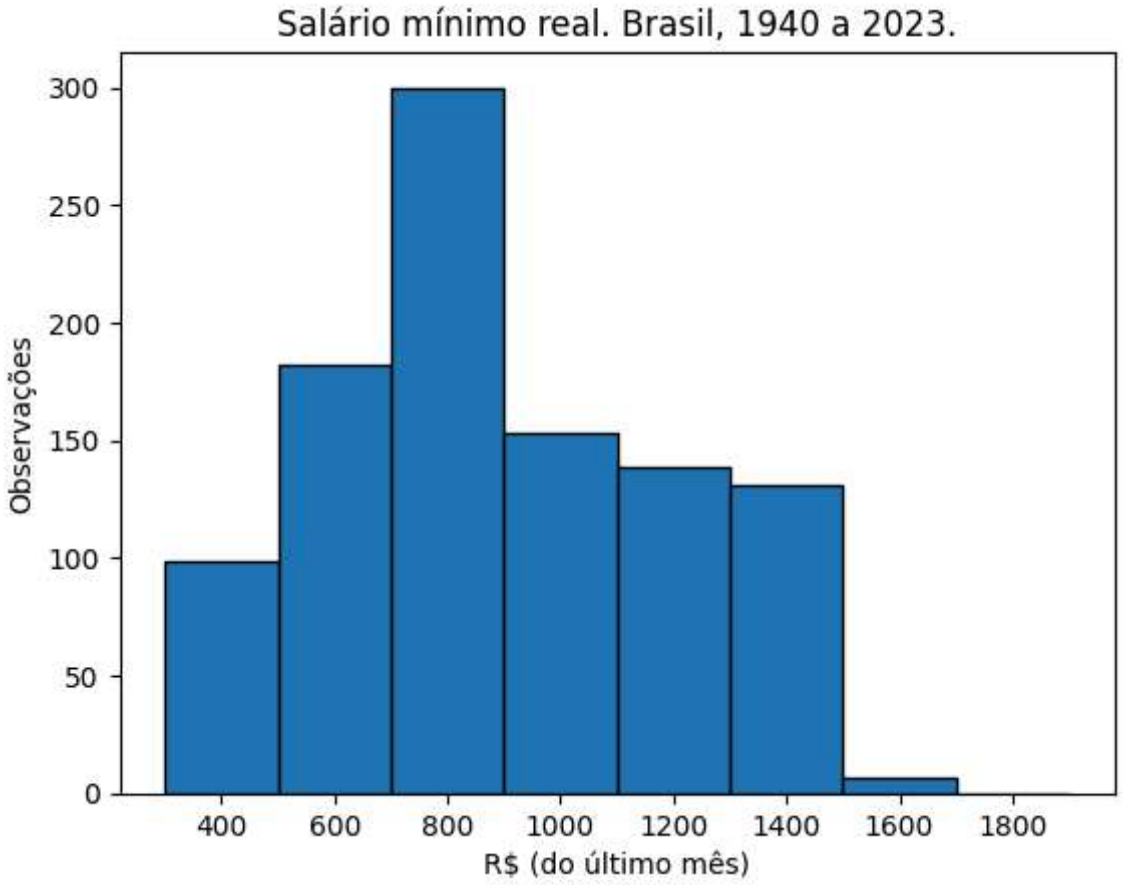
In [ ]:
```
!pip install ipeadatapy
```

Gerando um belo gráfico:

In [ ]:
```
import ipeadatapy

# o codigo da serie de salario minimo na base
# e 'GAC12_SALMINRE12'
df_salmin = ipeadatapy.timeseries('GAC12_SALMINRE12')

# 'bins' representa o parametro de intervalos
# estamos definindo os intervalos:
# [300, 500), [500, 700), [700, 900), [900, 1100),
# [1100, 1300), [1300, 1500), [1500, 1700), [1700, 1900)
ax = (df_salmin['VALUE (R$ (do último mês))']
      .plot
      .hist(bins = range(300, 1901, 200), edgecolor = (0, 0, 0))
)
```

```
# rotulo do eixo horizontal
ax.set_xlabel("R$ (do último mês)")

# rotulo do eixo vertical
ax.set_ylabel("Observações")

# titulo
ax.set_title("Salário mínimo real. Brasil, 1940 a 2023.")
```

Out[ ]: `Text(0.5, 1.0, 'Salário mínimo real. Brasil, 1940 a 2023.')`



> Podemos observar que, na maioria das vezes, o salário mínimo deflacionado (ou real), valia aproximadamente R $1000 (com referência do último mês). Vemos também que em alguns momentos, 1500. Por outro lado, há também muitos momentos em que foi bastante desvalorizado, ficando abaixo dos R$ 700.

Duas Dimensões

Reconsidere o clássico conjunto de dados descrevendo o peso (em libras) e altura (em polegadas) de adultos saudáveis:

In [ ]:
```
import requests
from bs4 import BeautifulSoup

# retirando o texto do HTML de uma pag na web
url = requests.get('http://socr.ucla.edu/docs/resources/SOCR_Data/SOCR_Da
```

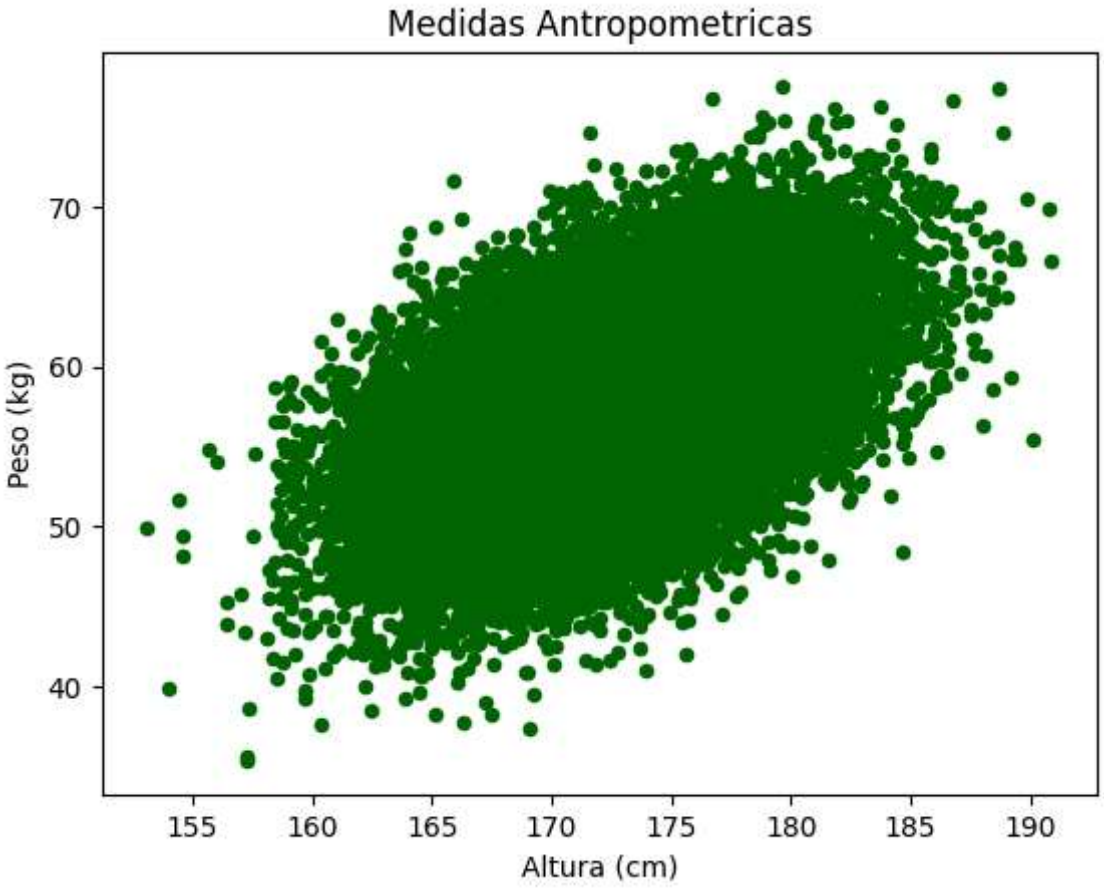
```
busca = BeautifulSoup(url,'lxml')
tabela = busca.find_all('table')

# tabela com as medidas antropometricas
# Height(inches) = Altura(polegadas)
# Weight(pounds) = Peso(libras)
medidas_antro = pd.read_html(str(tabela), header = 0)[0]

# removendo a primeira coluna
medidas_antro = medidas_antro.iloc[:, 1:3]

# alterando os nomes para portugues
medidas_antro.columns = ['altura', 'peso']

# transformando altura de polegadas para cm
medidas_antro["altura"] = medidas_antro["altura"] * 2.54

# transformando peso de libras para kg
medidas_antro["peso"] = medidas_antro["peso"] / 2.205
```

```
<ipython-input-14-90291976fded>:12: FutureWarning: Passing literal html to
'read_html' is deprecated and will be removed in a future version. To read
from a literal string, wrap it in a 'StringIO' object.
  medidas_antro = pd.read_html(str(tabela), header = 0)[0]
```

A imagem resultante pode ser vista abaixo:

In [ ]:
```
# grafico de dispersao
ax1 = medidas_antro.plot.scatter(x = 'altura',
                                 y = 'peso',
                                 c = 'DarkGreen')

# rotulo do eixo horizontal
ax1.set_xlabel("Altura (cm)")

# rotulo do eixo vertical
ax1.set_ylabel("Peso (kg)")

# titulo
ax1.set_title("Medidas Antropometricas")
```

Out[ ]:
```
Text(0.5, 1.0, 'Medidas Antropometricas')
```



Em geral, conforme a altura da pessoa aumenta, seu peso também aumenta (o que faz todo sentido físico). Embora, a relação não seja perfeita para todos e envolva outros aspectos que analisaremos mais a frente. Também podemos calcular algumas estatísticas interessantes... Mas isso também veremos nas próximas aulas!

Muitas Dimensões

Vamos retornar ao exemplo financeiro que vimos anteriormente:

In [ ]:
```
!pip install yfinance
```

Vamos analisar graficamente suas evoluções ao longo do ano de 2022:

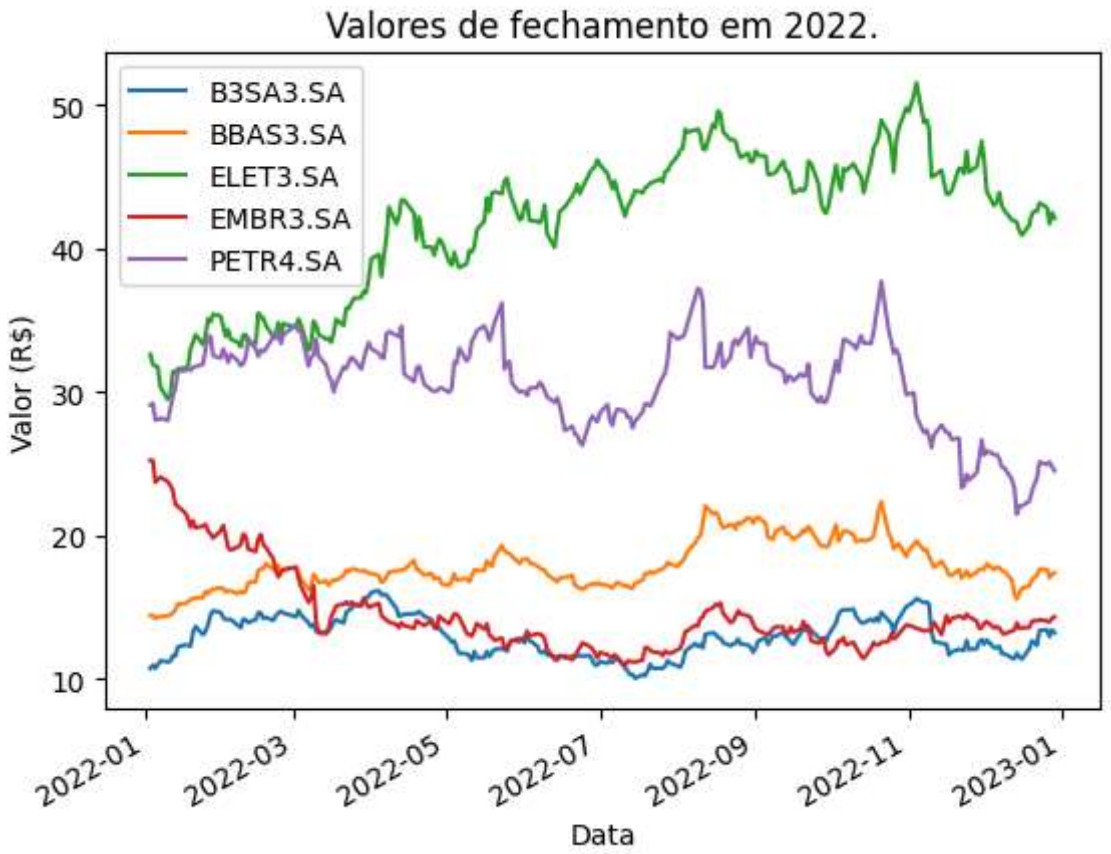
In [ ]:
```
import yfinance as yf

# B3SA3.SA: B3 S.A. - Brasil, Bolsa, Balcão
# BBAS3.SA: Banco do Brasil S.A.
# ELET3.SA: Centrais Elétricas Brasileiras S.A. - Eletrobrás
# EMBR3.SA: Embraer S.A.
# PETR4.SA: Petróleo Brasileiro S.A. - Petrobras
acoes_br = yf.download("B3SA3.SA BBAS3.SA ELET3.SA EMBR3.SA PETR4.SA",
                       start = "2022-01-01",
                       end = "2022-12-31")

# separando somente os valores de fechamento
acoes_br = acoes_br.iloc[:, 5:10]

# renomeando
```

```
acoes_br.columns = ['B3SA3.SA', 'BBAS3.SA',     'ELET3.SA',     'EMBR3.SA
acoes_br.index.name = 'data'
```

A figura abaixo exibe a evolução dos valores de fechamento de diversão ações do mercado brasileiro em 2022.

In [ ]:
```
ax = acoes_br.plot()

ax.set_xlabel("Data")
ax.set_ylabel("Valor (R$)")
ax.set_title("Valores de fechamento em 2022.")
```

Out[ ]:
```
Text(0.5, 1.0, 'Valores de fechamento em 2022.')
```



Isso é o suficiente para você começar a fazer visualização de seus dados. Aprenderemos muito mais sobre visualização ao longo do curso.

Referências Adicionais

- A galeria do matplotlib lhe dará uma boa ideia dos tipos de gráficos que você pode fazer (e como fazê-los);
- Seaborn permite que você produza visualizações ainda mais bonitas (e mais complexas);
- Bokeh é uma biblioteca que traz visualizações em estilo interativo para o Python.