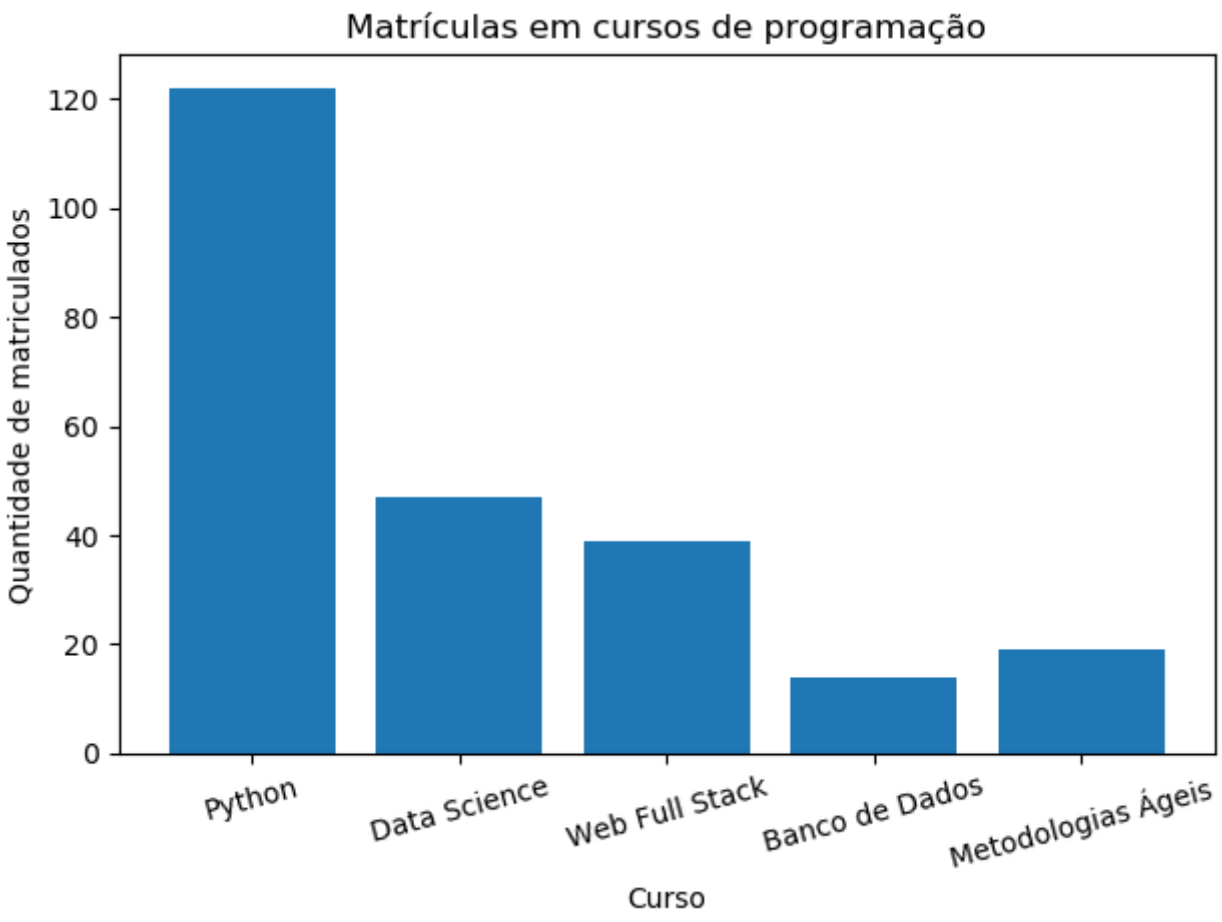
Outros tipos de gráfico

Vimos como gerar gráficos de linha utilizando o matplotlib e aprendemos a fazer diversas personalizações me nossos gráficos. Porém, podemos criar outros tipos de gráfico: gráficos de barras, gráficos de setores (pizza) e histogramas.

1. Gráficos de barras

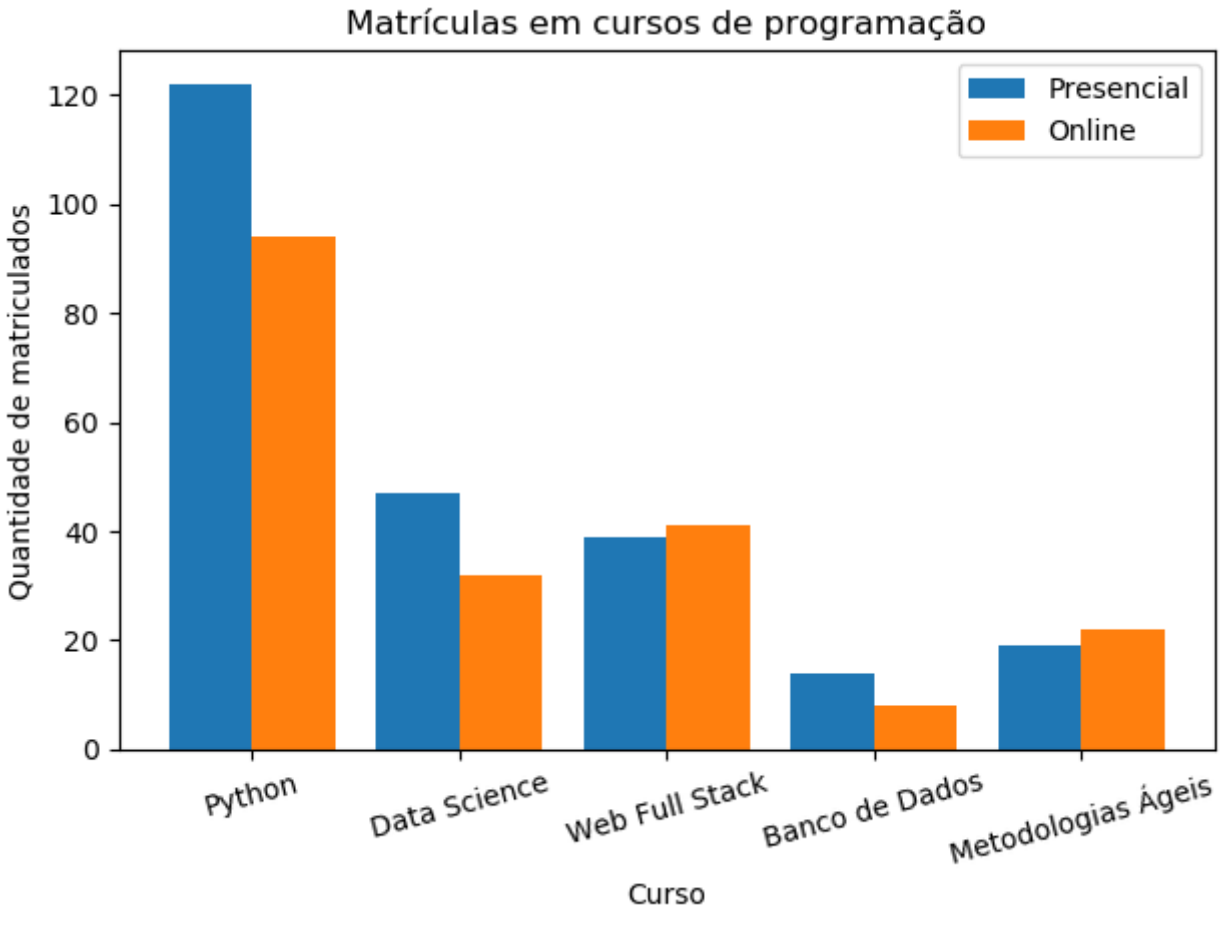
Para criar um gráfico de barras simples, o procedimento é bastante parecido com os gráficos de linha. Teremos uma coleção de dados para o eixo X (normalmente as diferentes categorias que gostaríamos de representar), outra coleção de dados para o eixo Y (normalmente os valores de cada categoria) e podemos fazer ajustes no nome dos eixos, título, utilizar *subplots* etc. Porém, ao invés de usar `plt.plot`, utilizaremos `plt.bar`. Vejamos um exemplo:

```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

cursos = ['Python', 'Data Science', 'Web Full Stack', 'Banco de Dados', 'Metodologias Ágeis']
matriculados = [122, 47, 39, 14, 19]

plt.bar(cursos, matriculados)
plt.title('Matrículas em cursos de programação')
plt.xlabel('Curso')
plt.ylabel('Quantidade de matriculados')
plt.xticks(rotation=15)
plt.tight_layout()

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```



1.1. Comparando subcategorias

No exemplo acima, estamos comparando o número de matriculados em diferentes cursos de uma escola de programação. Imagine, porém, que a escola abrisse uma modalidade online para os mesmos cursos, e nós desejássemos exibir lado-a-lado o número de matriculados para a versão online e a presencial de cada curso. O resultado desejado seria a imagem abaixo:



Para atingir esse resultado, nós usaremos o `plt.bar` 2 vezes: em uma passaremos a lista com o número de matriculados nos cursos presenciais, e na outra, o número de matriculados nos cursos online.

Porém, se fizermos isso diretamente, os gráficos ficarão sobrepostos. Precisamos fazer com que as barrinhas da série presencial fiquem um pouco deslocadas para a esquerda, e as da série online fiquem deslocadas um pouco para a direita. Portanto, teremos que gerar uma lista de valores para o eixo X do primeiro gráfico e outra lista, um pouco deslocada em relação a essa, para o segundo gráfico. Podemos usar uma "fórmula" pronta para gerar essas listas:

```
[t*x + l*n for x in range(d)]
```

Onde `t` é o número total de séries que temos, `l` é a largura de cada barrinha (o padrão é 0.8) e `n` é a série cujo eixo X estamos gerando. Não precisa se preocupar em decorar isso - guarde essa "fórmula" para usar sempre que necessário!

Assim, podemos plotar o gráfico da seguinte maneira:

```
plt.bar([t*x + l*1 for x in range(d)], presencial)
plt.bar([t*x + l*2 for x in range(d)], online)
```

Onde `presencial` e `online` são as listas com as quantidades de alunos matriculados por categoria.

Agora precisamos fazer com que as "marquinhas" no eixo X apareçam bem no meio das barras, e o texto de cada marquinha seja o nome do curso. Para isso usaremos uma outra fórmula (com as mesmas variáveis definidas anteriormente) e a função `plt.xticks`, que recebe a lista de valores do eixo X onde as marquinhas devem ser exibidas, e um parâmetro opcional `labels` que recebe uma lista com os textos a serem exibidos:

```
ticks = [t*x + l*t - l*t/2 + l/t for x in range(d)]
plt.xticks(ticks, labels=cursos)
```

Vejamos o código completo:

```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

cursos = ['Python', 'Data Science', 'Web Full Stack', 'Banco de Dados', 'Metodologias Ágeis']
presencial = [122, 47, 39, 14, 19]
online = [94, 32, 41, 8, 22]

# A lista de valores  de "x" para cada série é dada por: [t*x + l*n for x in range(d)]
# onde:
# n: número da série atual
# t: número total de séries
# d: número de pontos
# l: largura de cada barra (o padrão é 0.8)
t = 2 # temos 2 séries: online e presencial
d = 5 # cada série tem 5 pontos (no caso, 5 cursos)
l = 0.8

plt.bar([t*x + l*1 for x in range(d)], presencial)
plt.bar([t*x + l*2 for x in range(d)], online)
```

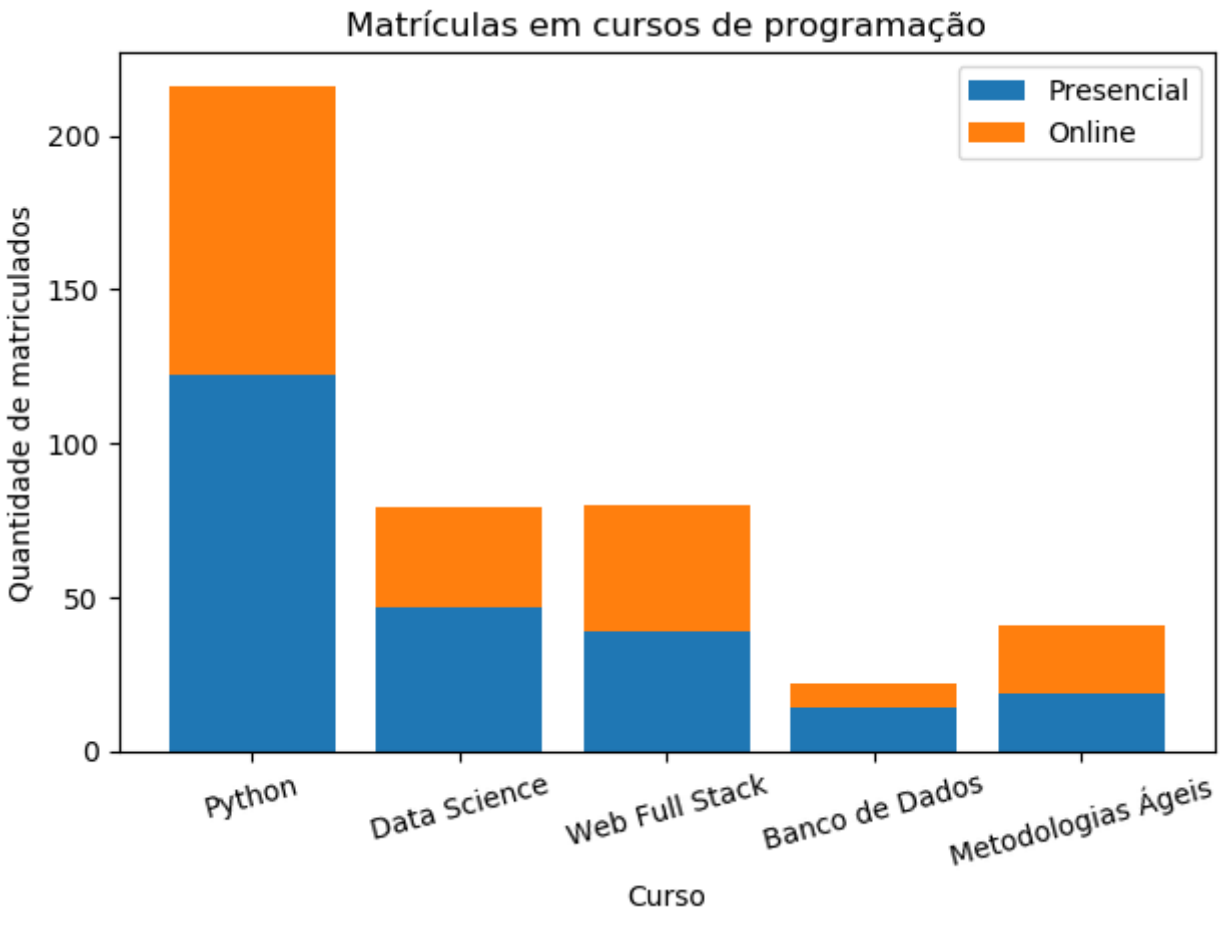
```
# Agora queremos que a "marquinha" fique bem no centro de cada bloco de barras...
# Mas não queremos que apareça números, e sim o nome dos cursos, por isso usaremos o parâmetro "labels"
ticks = [t*x + l*t - l*t/2 + l/t for x in range(d)]
plt.xticks(ticks, labels=cursos, rotation=15)
plt.legend(['Presencial', 'Online']) # exibindo a legenda indicando as categorias
plt.title('Matrículas em cursos de programação')
plt.xlabel('Curso')
plt.ylabel('Quantidade de matriculados')
plt.tight_layout()

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```

1.2. Empilhando categorias

O gráfico que vimos acima é útil para comparar explicitamente as categorias. Ele evidencia bem a divisão entre matriculados na modalidade presencial ou online. Mas em contrapartida ele sacrifica um pouco a visão do total de matriculados em cada curso. Se ao invés de colocarmos as informações lado-a-lado colocássemos empilhadas, essa informação seria mais visível.

O `plt.bar` possui o parâmetro opcional `bottom`. Ele recebe uma lista de valores indicando a partir de qual altura as barras devem ser desenhadas. Podemos mandar desenhar uma categoria (por exemplo, a categoria "presencial" de nosso exemplo), e em seguida, mandamos desenhar a outra categoria passando como `bottom` as alturas da categoria anterior.

```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

cursos = ['Python', 'Data Science', 'Web Full Stack', 'Banco de Dados', 'Metodologias Ágeis']
presencial = [122, 47, 39, 14, 19]
online = [94, 32, 41, 8, 22]

# primeiro desenhamos o presencial:
plt.bar(cursos, presencial)
# agora desenhamos o online acima do presencial:
plt.bar(cursos, online, bottom = presencial)
plt.xticks(rotation = 15)
plt.legend(['Presencial', 'Online'])
plt.title('Matrículas em cursos de programação')
plt.xlabel('Curso')
plt.ylabel('Quantidade de matriculados')
plt.tight_layout()

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```



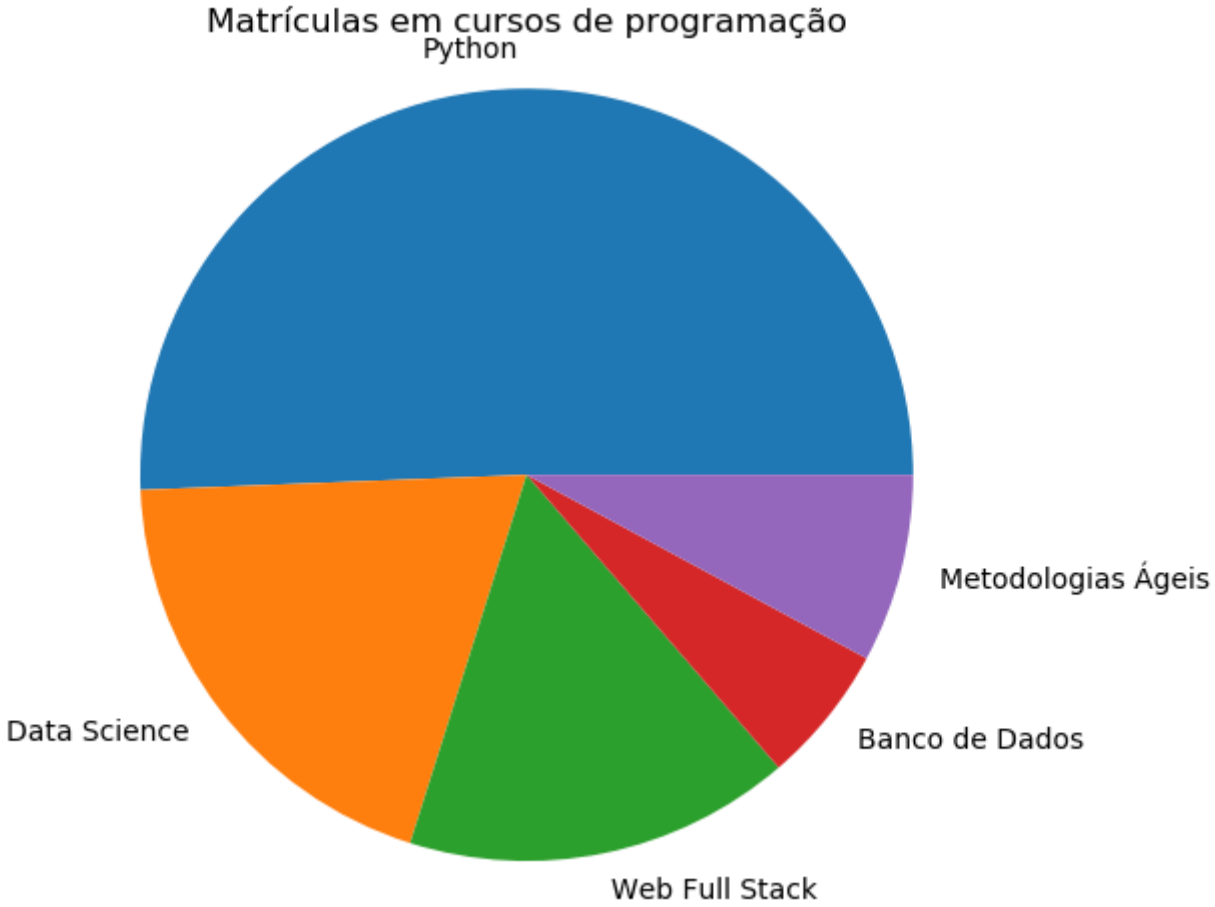
2. Gráficos de setores

Os gráficos de setores, popularmente conhecidos por gráficos de pizza (ou *pie charts*, "gráficos de torta" em inglês) são bastante interessantes quando ao invés de comparar os valores totais de cada categoria nós preferimos enfatizar o quanto aquela categoria representa do total.

Utilizaremos o `plt.pie` para gerar esses gráficos, e ele é bastante simples de usar. Basta passar uma coleção com os valores, e opcionalmente, uma coleção de *labels*. Normalmente utilizaremos também a linha `plt.axis('equal')` para deixar o gráfico plano e redondinho - sem ela, o matplotlib aplica um "efeito 3D" que na maioria dos casos é indesejado e acaba distorcendo um pouco o gráfico.

Vejamos um gráfico de pizza com os dados do primeiro exemplo deste capítulo:

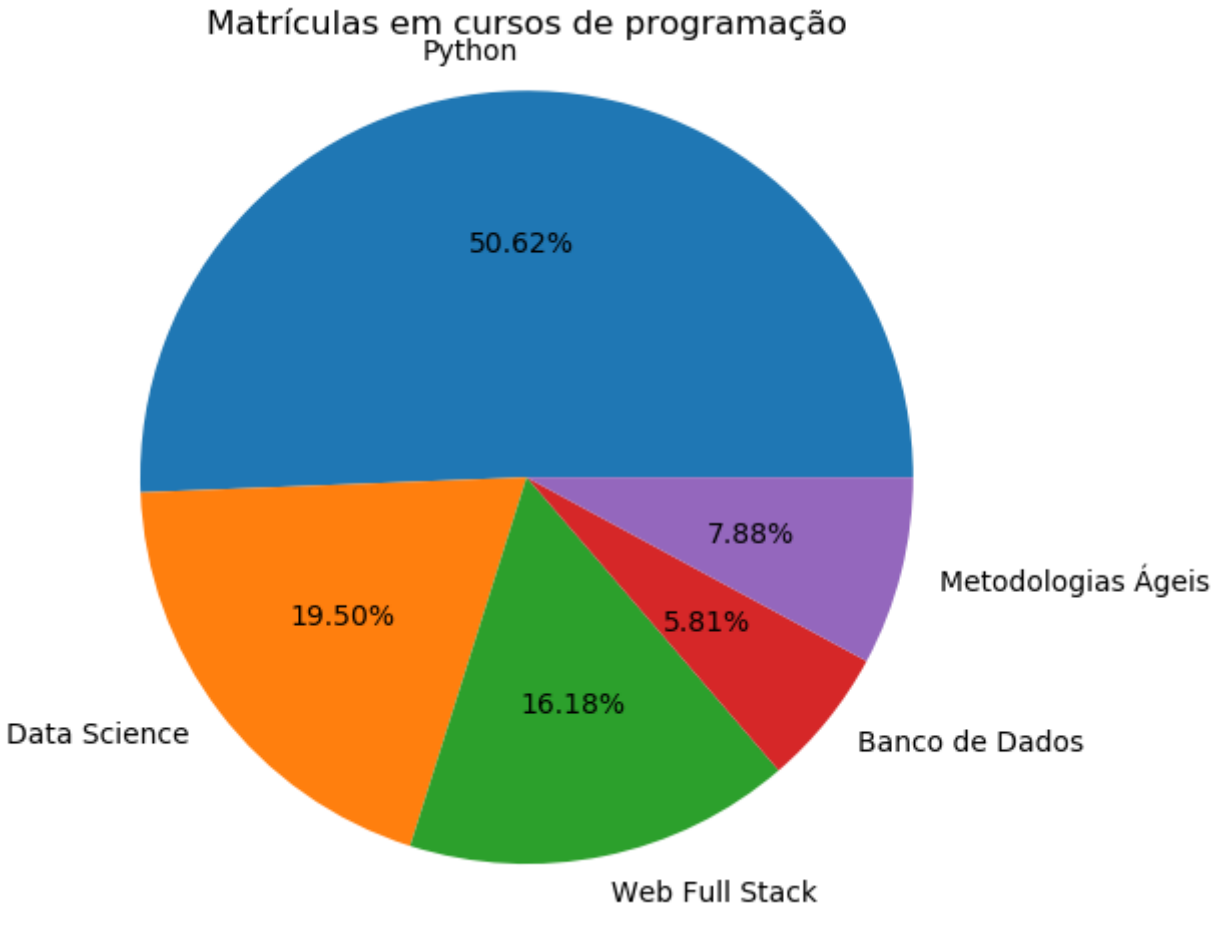
```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

cursos = ['Python', 'Data Science', 'Web Full Stack', 'Banco de Dados', 'Metodologias Ágeis']
matriculados = [122, 47, 39, 14, 19]

plt.pie(matriculados, labels = cursos)
plt.axis('equal') # deixando o gráfico redondinho
plt.title('Matrículas em cursos de programação')

plt.tight_layout() # deixando as informações bem centralizadas na tela

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```



2.1. Incluindo percentuais

Podemos incluir em nosso gráfico o percentual que cada fatia representa no total utilizando o parâmetro opcional `autopct`. Esse parâmetro recebe uma string indicando se os valores devem ser arredondados para inteiro (`%d%%`) ou para número com *n* casas após a vírgula (`%.nf%%`, substituindo n pelo número de casas).

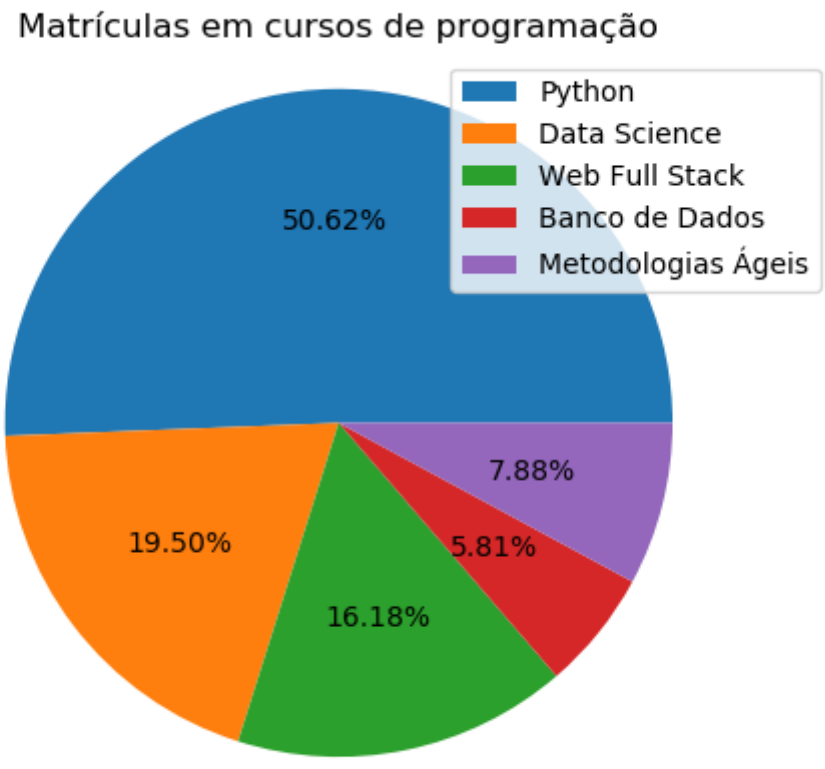
Vamos redesenhar o gráfico acima incluindo o percentual com 2 casas após a vírgula. Basta acrescentar o parâmetro `autopct='%.2f%%'` à chamada de `plt.pie` do exemplo anterior.



2.2. Legendas

Dependendo de quantos setores o seu gráfico possui, escrever o nome das categorias próximo de cada setor como fizemos nos dois exemplos anteriores pode poluir bastante o visual. Uma alternativa é não passar o parâmetro *labels*, e ao invés disso chamar `plt.legends` (como fizemos em outros tipos de gráfico) para que a caixinha com a legenda de cores seja exibida ao lado do gráfico.

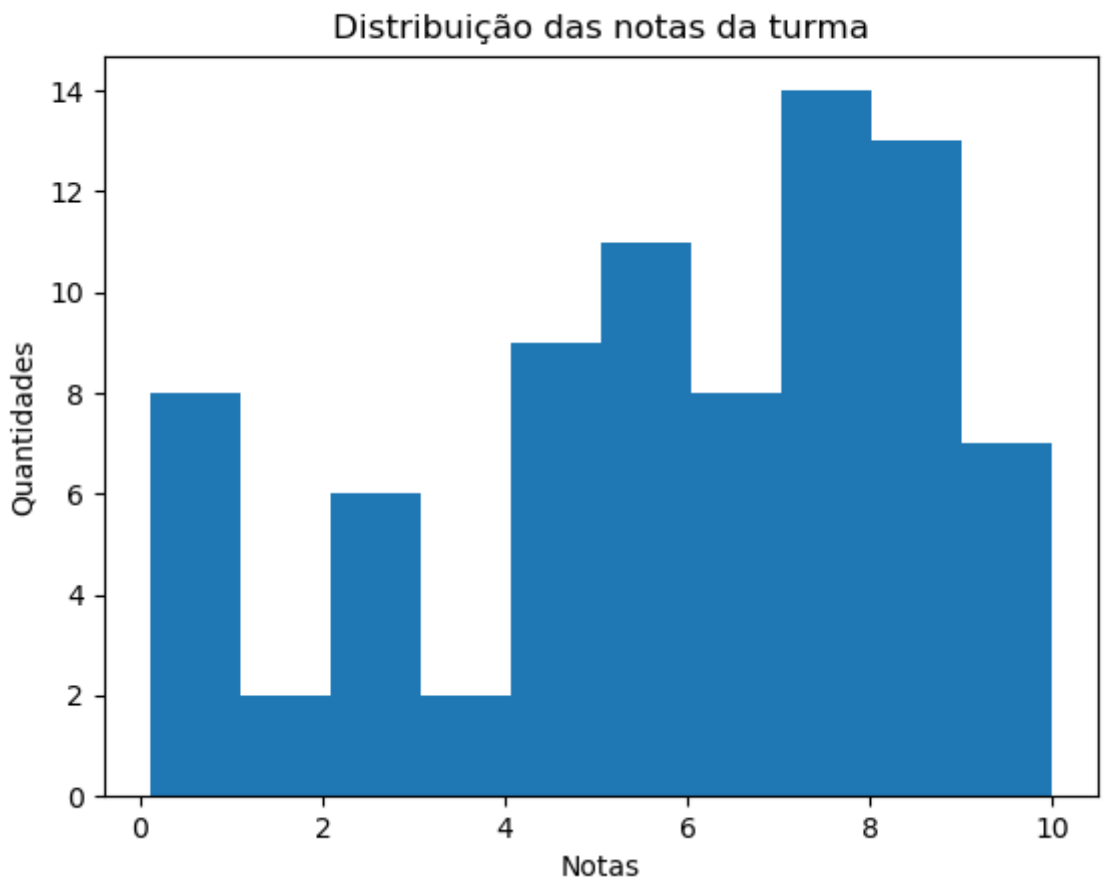
```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
#%matplotlib inline

cursos = ['Python', 'Data Science', 'Web Full Stack', 'Banco de Dados', 'Metodologias Ágeis']
matriculados = [122, 47, 39, 14, 19]

plt.pie(matriculados, autopct='%.2f%%')
plt.axis('equal') # deixando o gráfico redondinho
plt.title('Matrículas em cursos de programação')
plt.legend(cursos)

# incluir a linha abaixo se não estiver no Jupyter
plt.show()
```



3. Histogramas

Outro gráfico que nos interessa é o histograma. Ele nos dá a ideia de *distribuição* dos nossos dados. Ele é uma espécie de gráfico de barras que nos mostra quantos pontos de nosso conjunto se encaixam em cada intervalo de valores.

No histograma abaixo, podemos facilmente verificar que em uma turma, 8 alunos tiraram entre 0 e 1, 2 alunos tiraram entre 1 e 2, 6 alunos tiraram entre 2 e 3, e assim sucessivamente.



Para gerar esse histograma, basta usar `plt.hist`, que recebe o conjunto de dados e faz tudo automaticamente.

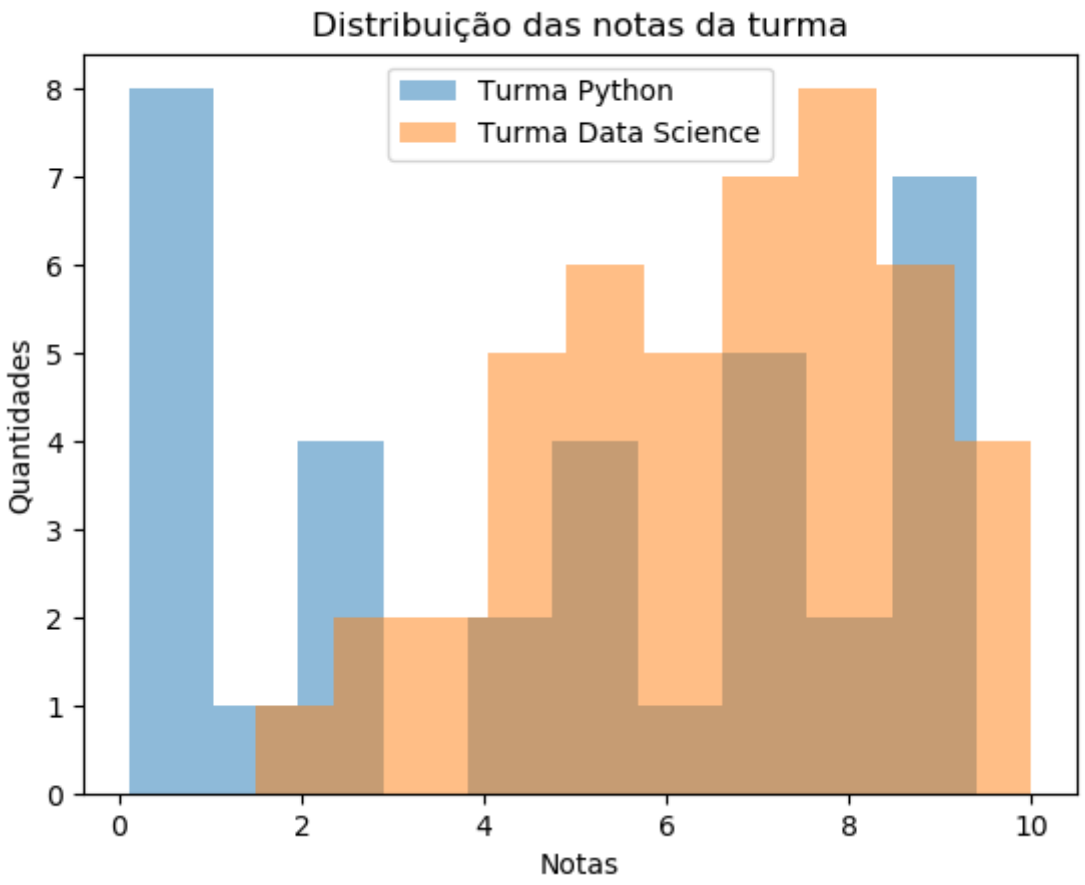
```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

notas = [2.3,2.1,1.6,2.6,0.1,0.8,0.2,0.2,8.8,7.1,
7.4,8.3,6.4,7.0,7.5,8.8,1.0,0.7,0.6,0.7,9.3,9.4,
8.9,8.9,4.3,9.3,7.8,7.3,5.3,4.2,5.2,5.5,5.4,2.2,
4.0,5.3,4.2,4.8,2.6,1.5,7.4,3.0,8.4,5.9,7.8,9.0,
7.5,8.0,8.4,8.1,8.3,9.0,7.9,7.3,7.3,8.7,5.5,5.3,
4.0,5.8,5.3,4.3,7.8,7.0,9.6,10.0,9.4,9.5,8.0,6.8,
6.2,8.1,5.0,4.5,4.6,6.0,6.6,5.0,6.1,7.0]

plt.hist(notas)
plt.title('Distribuição das notas da turma')
plt.xlabel('Notas')
plt.ylabel('Quantidades')

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```

3.1. Comparando distribuições

Podemos comparar dados diferentes chamando o `plt.hist` sucessivas vezes passando diferentes conjuntos. Porém, como eles ficarão sobrepostos, um deles pode prejudicar a visualização do outro. O parâmetro `alpha`, que varia de 0 a 1, pode deixar o gráfico mais transparente, de modo que a gente possa visualizar ambas as curvas sem dificuldade.

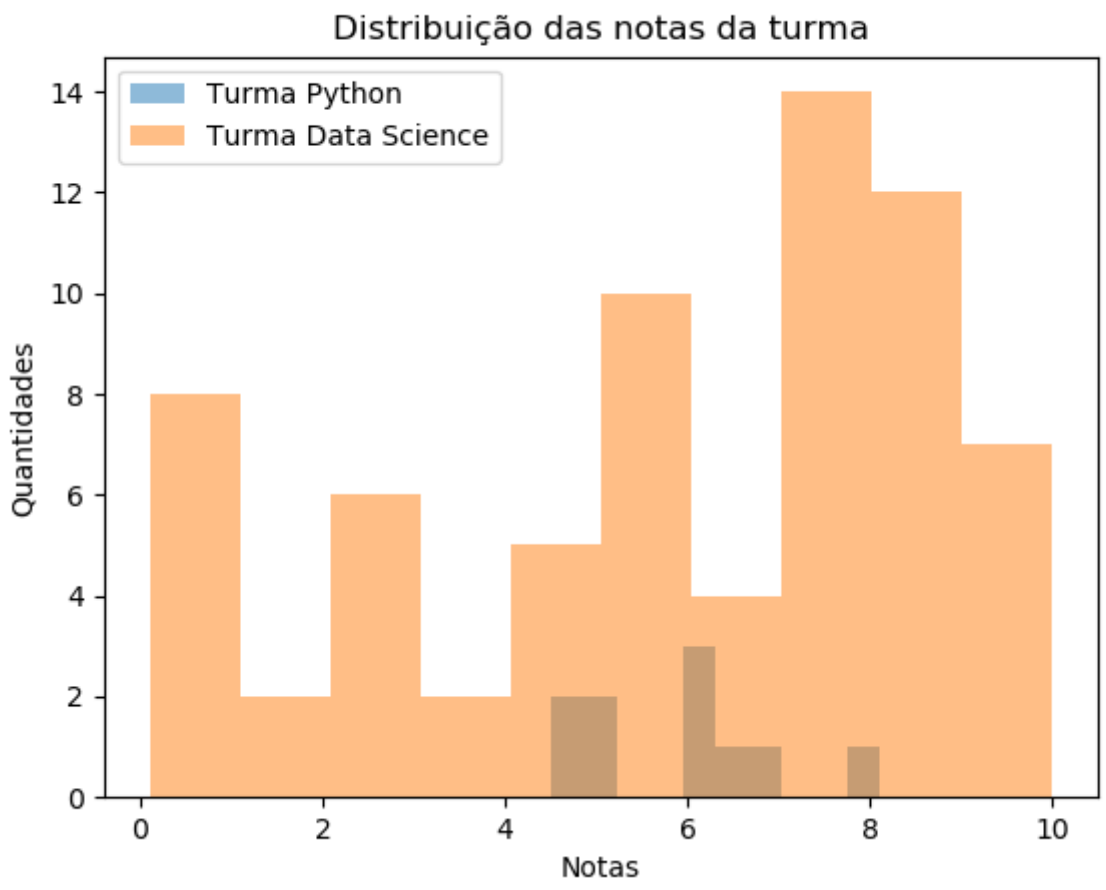
```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

turmaPy = [2.3,2.1,1.6,2.6,0.1,0.8,0.2,0.2,8.8,7.1,
7.4,8.3,6.4,7.0,7.5,8.8,1.0,0.7,0.6,0.7,9.3,9.4,
8.9,8.9,4.3,9.3,7.8,7.3,5.3,4.2,5.2,5.5,5.4,2.2]
turmaDS = [4.0,5.3,4.2,4.8,2.6,1.5,7.4,3.0,8.4,5.9,7.8,9.0,
7.5,8.0,8.4,8.1,8.3,9.0,7.9,7.3,7.3,8.7,5.5,5.3,
4.0,5.8,5.3,4.3,7.8,7.0,9.6,10.0,9.4,9.5,8.0,6.8,
6.2,8.1,5.0,4.5,4.6,6.0,6.6,5.0,6.1,7.0]

plt.hist(turmaPy, alpha=0.5)
plt.hist(turmaDS, alpha=0.5)
plt.title('Distribuição das notas da turma')
plt.xlabel('Notas')
plt.ylabel('Quantidades')
plt.legend(['Turma Python', 'Turma Data Science'])
```

```
# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```

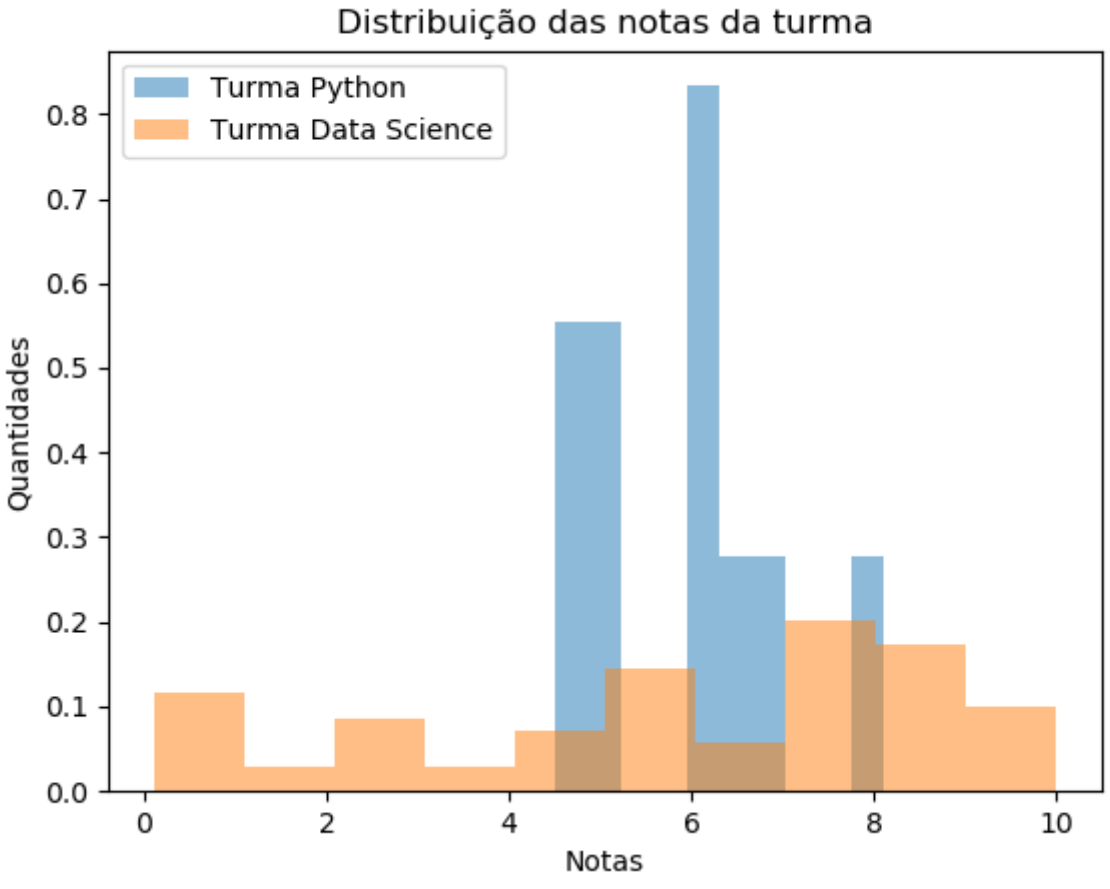


3.2. Normalizando os dados

O que aconteceria se uma das turmas do exemplo anterior tivesse uma quantidade muito inferior de alunos do que a outra? Digamos que 5 alunos fossem de Data Science e o restante todo fosse de Python. Como isso impactaria nosso histograma?

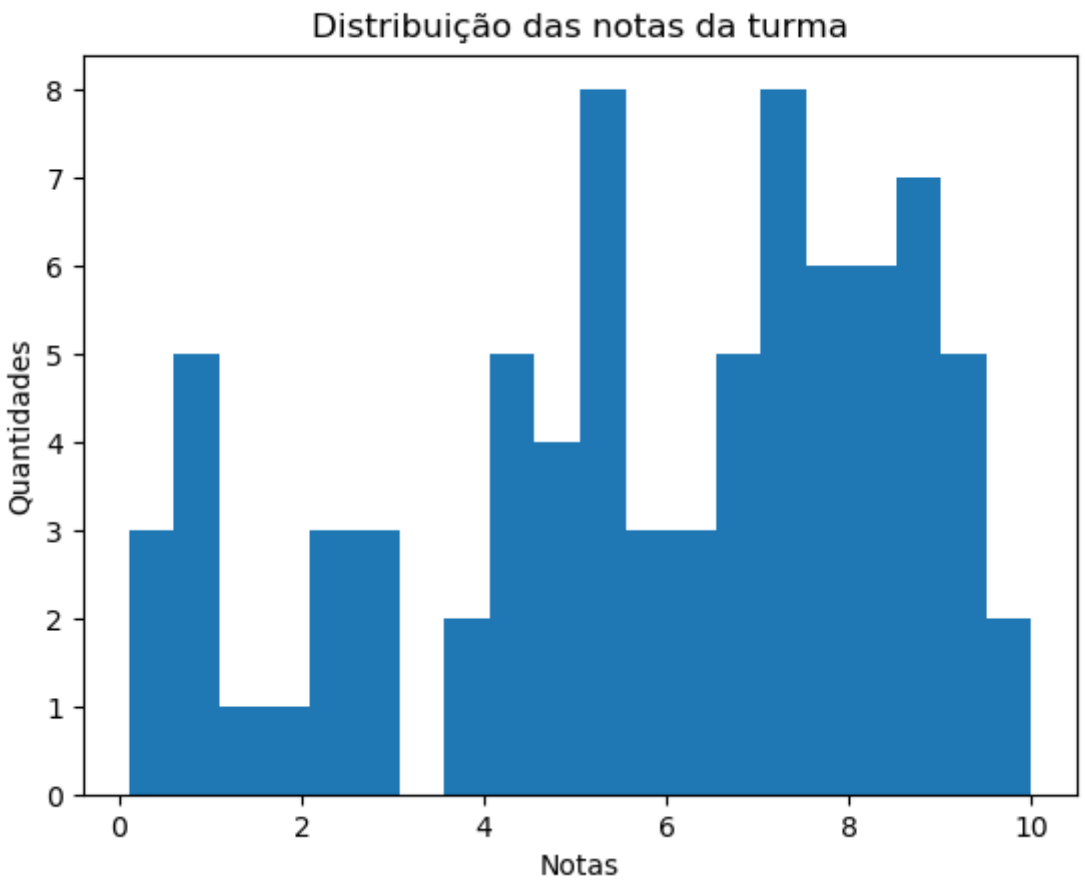


Difícil comparar a distribuição das turmas assim, né? Podemos passar o parâmetro `normed=True` em nossas chamadas a `plt.hist`. Isso fará com que os dados sejam *normalizados*: a área sob as curvas será necessariamente igual a 1, independentemente do tamanho dos nossos conjuntos. Assim fica mais fácil comparar o desempenho de 2 turmas:

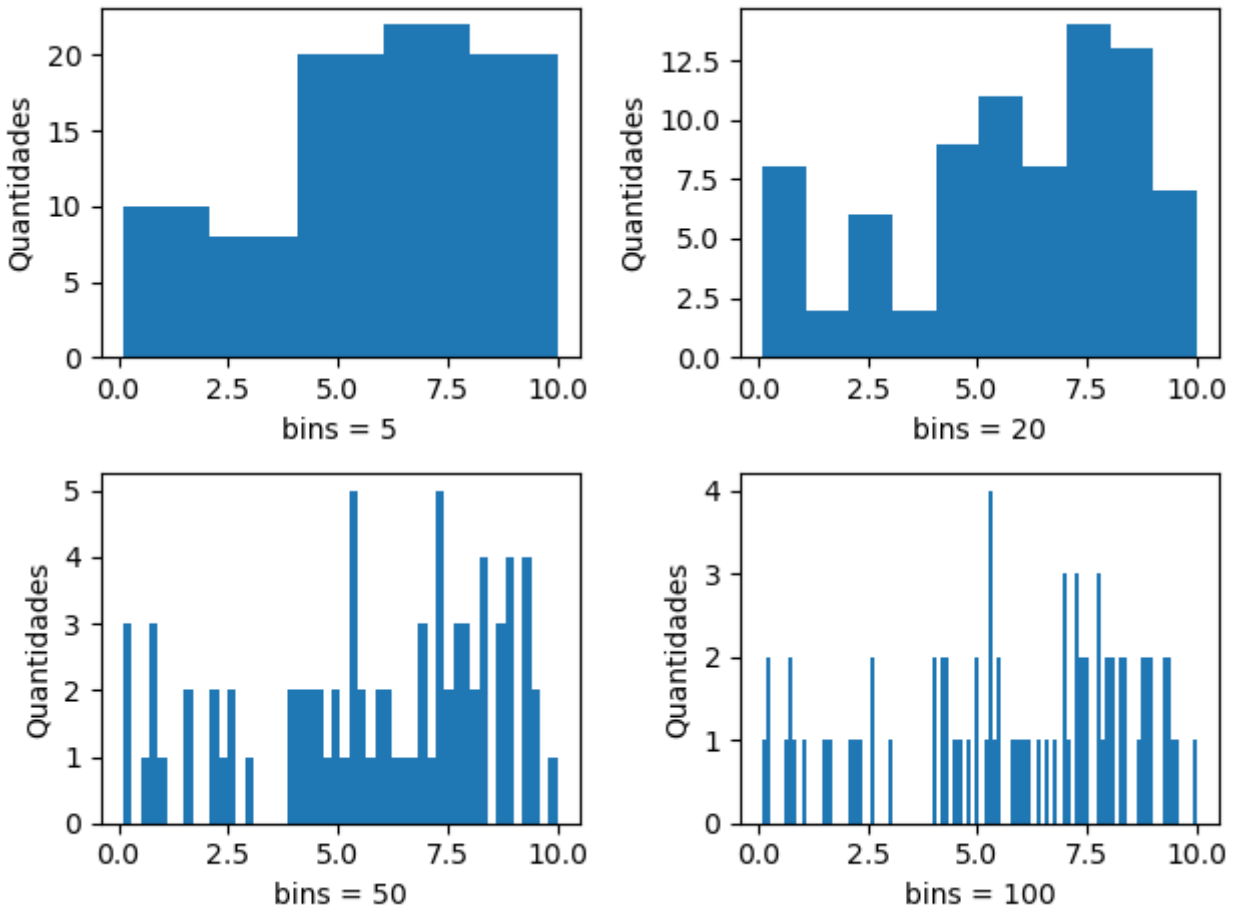


3.3. Alterando os intervalos

Podemos também alterar as subdivisões adotadas em nosso histograma passando o parâmetro opcional `bins`. Nossos exemplos estão utilizando o valor padrão de 10 subdivisões, o que faz nossas notas serem agrupadas nos intervalos 0-1, 1-2, 2-3, 3-4, ... 9-10. O que acontecerá se definirmos que gostaríamos de 20 *bins* ao invés de 10? Faça o teste no primeiro exemplo passando o parâmetro `bins=20` em `plt.hist`.



> Importante: o número de *bins* pode alterar bastante a forma do gráfico. Uma escolha inadequada de subdivisões para a sua base de dados pode acabar dando muita ênfase para informações pouco relevantes. Os quatro gráficos abaixo foram obtidos a partir do mesmo conjunto de



dados:

4. Erros

Nem sempre nossos dados são confiáveis, e é desejável que nossos gráficos reflitam até que ponto podemos confiar neles.

Considere, por exemplo, pesquisas eleitorais. Sempre que uma pesquisa de intenção de voto é divulgada, o jornal informa a margem de erro da pesquisa. Essa margem de erro pode ser uma barrinha ou uma área sombreada indicando a região dentro da qual aquele valor pode variar.

Vejamos um exemplo:

Semana de campanha	Candidato A	Candidato B
Semana 1	51%	49%
Semana 2	48%	52%
Semana 3	47%	53%
Semana 4	46%	54%

Se a margem de erro da pesquisa é de 2 pontos percentuais para mais ou para menos, a partir de que ponto podemos afirmar com segurança qual candidato está na frente?

4.1. Erro em barras

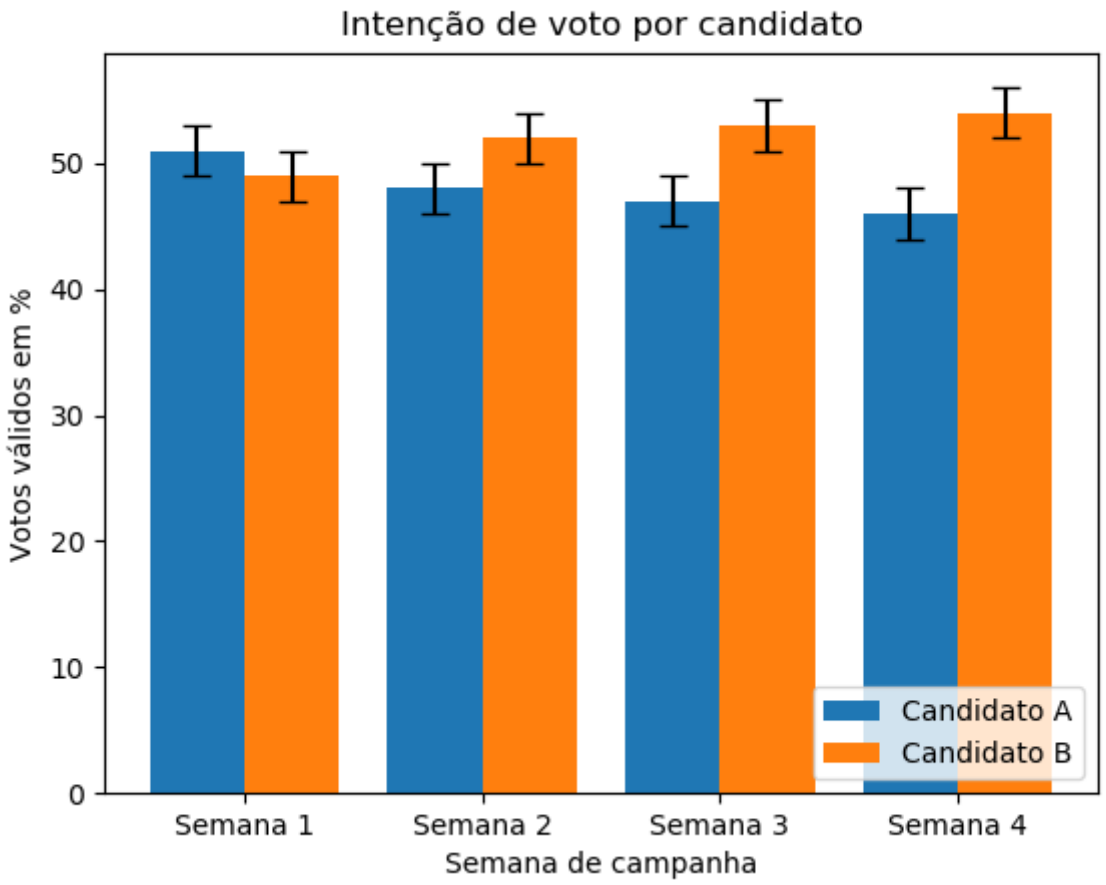
Para acrescentar uma barrinha de erro, modificaremos passaremos para a função que desenhou o gráfico o parâmetro opcional `yerr`. Podemos ajustar a largura de suas bases através do parâmetro `capsize`.

Vejamos como ficaria o gráfico de barras da nossa pesquisa agora com a barra de erro inclusa:

```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

candidatoA = [51, 48, 47, 46]
candidatoB = [49, 52, 53, 54]
datas = ['Semana 1', 'Semana 2', 'Semana 3', 'Semana 4']

# t: número total de séries
# d: número de pontos
# l: largura de cada barra (o padrão é 0.8)
t = 2 # temos 2 séries: candidatoA e candidatoB
d = 4 # temos 4 semanas
l = 0.8

# yerr = 2: variação de mais ou menos 2 no eixo Y
# capsize = 5: ajustando a largura da base da barrinha de erro
plt.bar([t*x + l*1 for x in range(d)], candidatoA, yerr=2, capsize=5)
plt.bar([t*x + l*2 for x in range(d)], candidatoB, yerr=2, capsize=5)

ticks = [t*x + l*t - l*t/2 + l/t for x in range(d)]
plt.xticks(ticks, labels=datas)
plt.legend(['Candidato A', 'Candidato B'], loc=4)
plt.title('Intenção de voto por candidato')
```

```
plt.xlabel('Semana de campanha')
plt.ylabel('Votos válidos em %')

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```

> O parâmetro `loc` em plt.legend força uma posição específica para a posição onde a legenda ficará no gráfico. Consulte os valores possíveis em https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.legend.html



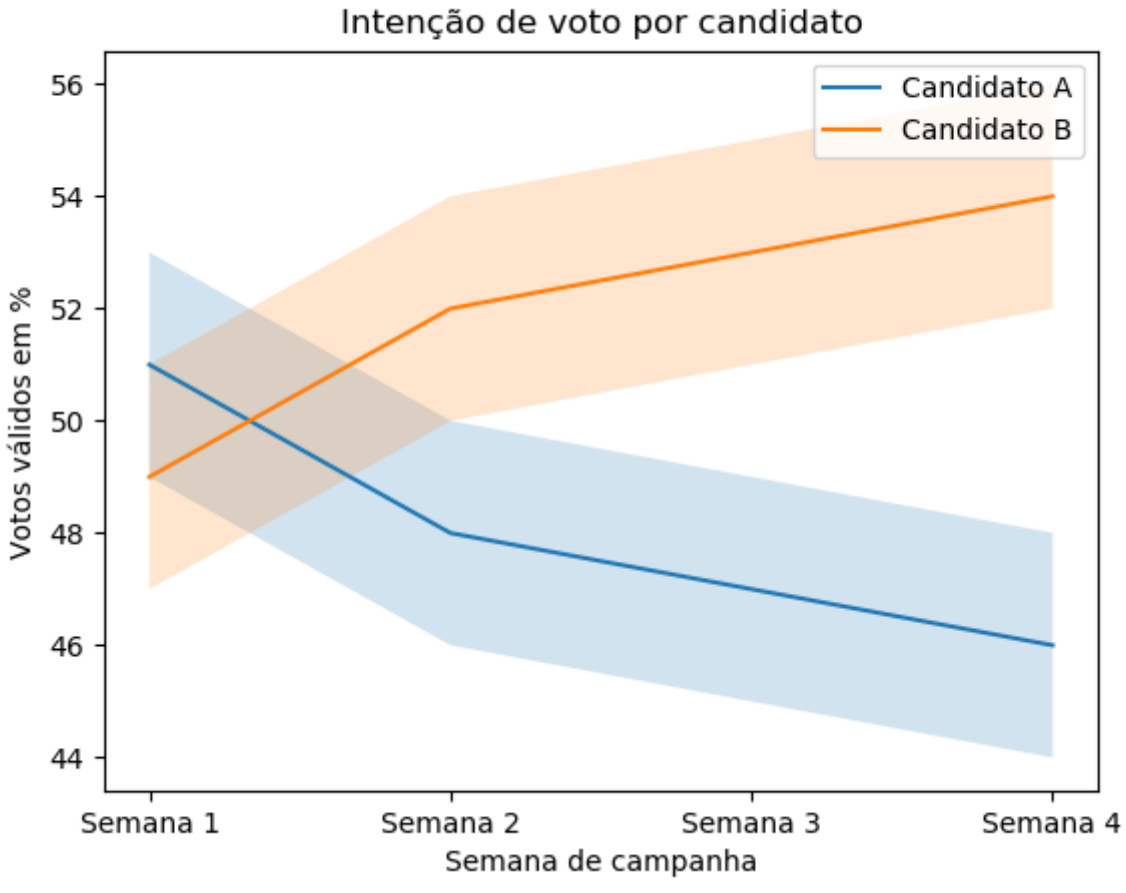
Podemos ver que as barras de erro dos candidatos se interceptam nas 2 primeiras semanas, caracterizando um empate técnico. Só podemos afirmar com convicção a liderança do Candidato B a partir da 3ª semana.

4.2. Erro em linhas

Caso optássemos por mostrar os resultados das pesquisas como séries temporais, uma opção seria mostrar o erro como uma zona sombreada ao redor da linha.

A função `plt.fill_between` faz o sombreamento. Ela espera 3 parâmetros: os valores de x, os valores inferiores da margem de erro para y e os valores superiores da margem de erro para y. Devemos calcular nós mesmos esses valores. O parâmetro opcional `alpha` é um número de 0 a 1 indicando o nível de transparência dessa região de erro.

```
import matplotlib.pyplot as plt
# remover a linha abaixo se não estiver no Jupyter
%matplotlib inline

candidatoA = [51, 48, 47, 46]
candidatoB = [49, 52, 53, 54]
datas = ['Semana 1', 'Semana 2', 'Semana 3', 'Semana 4']

plt.plot(candidatoA)
plt.plot(candidatoB)

# calculando as margens de erro para o candidato B
candidatoBsup = [x+2 for x in candidatoB]
candidatoBinf = [x-2 for x in candidatoB]
# calculando as margens de erro para o candidato A
candidatoAsup = [x+2 for x in candidatoA]
candidatoAinf = [x-2 for x in candidatoA]

# preenchendo o erro
plt.fill_between(datas, candidatoAinf, candidatoAsup, alpha=0.2)
plt.fill_between(datas, candidatoBinf, candidatoBsup, alpha=0.2)

plt.legend(['Candidato A', 'Candidato B'])
plt.title('Intenção de voto por candidato')
plt.xlabel('Semana de campanha')
plt.ylabel('Votos válidos em %')

# incluir a linha abaixo se não estiver no Jupyter
#plt.show()
```



E novamente podemos tirar a mesma conclusão sobre o empate técnico nas duas primeiras semanas.