O *package* Matplotlib – uma 1ª visão

- No mundo da ML a visualização gráfica da informação desempenha um papel vital
  - *É através da visualização gráfica que, muitas vezes, se conseguem identificar rapidamente caraterísticas e relações importantes entre os dados*
- O Matplotlib é atualmente o package de visualização científica mais popular do Python (segue o estilo das apresentações gráficas do MATLAB)
  - *a sua interface não é das mais simples, mas trata-se de uma biblioteca poderosa para criação de uma grande variedade de gráficos de boa qualidade*
- De forma a se usar uma interface estilo MATLAB na produção dos gráficos, começa-se por importar o módulo pyplot do Matplotlib, como plt:

```
import matplotlib.pyplot as plt
```

  - *Depois, criando, por exemplo, um array NumPy unidimensional com os dados da abcissa (50 valores igualmente espaçados entre 0 e 2π)*

```
import numpy as np
x=np.linspace(0,2*np.pi)
```

  - *e um segundo array com os dados da ordenada (função seno)*

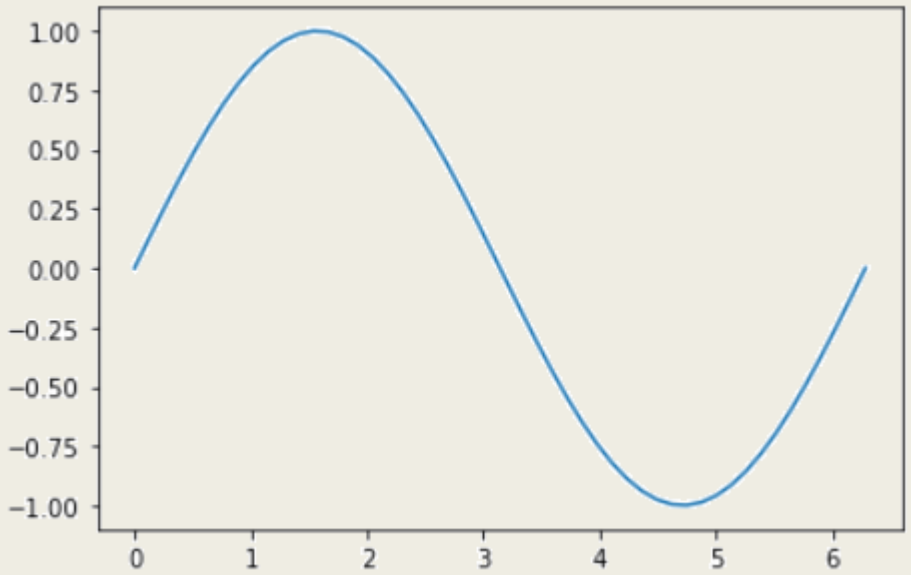
```
y=np.sin(x)
```

  - *é possível desenhar facilmente o respetivo gráfico com função plot() do módulo plt*

```
plt.plot(x,y)
```

O *package* Matplotlib – uma 1ª visão

```
plt.plot(x,y)
```



- Clicando no gráfico, dentro do ambiente de programação, tem-se acesso a algumas operações básicas, como ampliar e converter para outros formatos.

Nesta breve apresentação, mostrou-se um exemplo muito simples de criação de um gráfico no Matplotlib

- *muitos outros tipos de gráficos mais elaborados podem ser explorados consultando a galeria online do Matplotlib*
  - https://matplotlib.org/gallery.html

Desvendando um pouco mais o Matplotlib

- Com o Matplotlib conseguem-se gerar facilmente gráficos 2D complexos e de boa qualidade,
  - *e quando integrado no Jupyter Notebook torna-se numa ferramenta ainda mais poderosa no mundo da ML*
- O seu módulo pyplot, por sua vez, fornece-nos uma interface para a própria biblioteca Matplotlib
  - *torna a produção de gráficos uma tarefa mais simples, disponibilizando-nos recursos para controlarmos estilos de linhas, propriedades de fonte, formatação de eixos, legendas, tamanho e posicionamento dos gráficos, etc*
  - *juntamente com o NumPy, proporciona-nos um ambiente alternativo ao MatLab, com muitos comandos e funcionalidades similares*
- Como é com o pyplot que vamos interagir, é esse módulo que deveremos importar
  - *o pyplot é importado como plt, por convenção, podendo assumir essa importação duas formas alternativas:*

```
import matplotlib.pyplot as plt
```

```
from matplotlib import pyplot as plt
```

Gráfico de linhas

- Um gráfico de linhas pode ser gerado usando simplesmente duas listas, a 1ª com os valores para o eixo dos x e a 2ª com os valores para o eixo dos y

```
x=list(range(20))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
y=[v%5 for v in x]
```

```
[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

```
plt.plot(x,y)

plt.title("Resto da divisão por 5") # título
plt.xlabel("Dividendo") # nome do eixo do x
plt.ylabel("Resto") # nome do eixo do y
```

Tal como se exemplifica, o título, nome dos eixos e outras especificações, devem ser indicadas separadamente, mas junto à função plot() – principal responsável pela produção do gráfico

Estilos de gráficos predefinidos

- Para que não percamos muito tempo a configurar os gráfico, o Matplolib disponibiliza-nos um conjunto de estilos predefinidos
  - *um desses estilos é o 'ggplot', que se baseia no package de visualização da linguagem R*
  - *Para escolhermos um estilo, usamos a função use() do módulo style:*

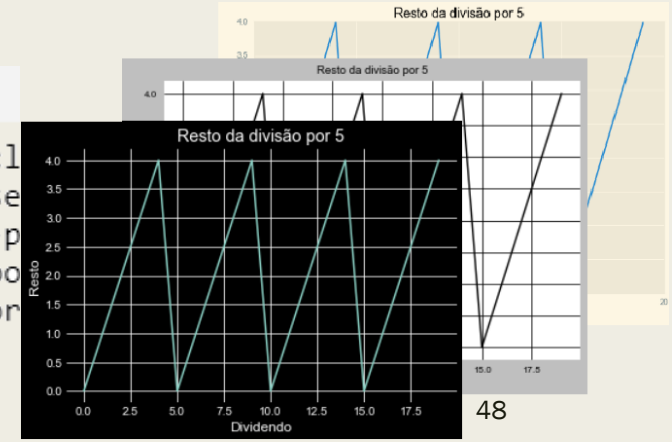
```
from matplotlib import style
style.use("ggplot")
```

  - *Caso tivéssemos escolhido este estilo antes da produção do gráfico, o mesmo teria o aspeto que se apresenta.*

- Muitos outros estilos predefinidos estão disponíveis

```
print(style.available)
```

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'cl
'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'se
'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-p
d', 'seaborn-deep', 'seaborn-muted', 'seaborn-noteboo
rn-pastel', 'seaborn-poster', 'seaborn-talk', 'seabor
'seaborn-whitegrid', 'tableau-colorblind10']
```



Sobreposição de funções num mesmo gráfico

- Invocando a função plot() sucessivas vezes, várias funções vão sendo desenhadas no mesmo gráfico
  - *Neste tipo de gráficos, com funções sobrepostas, há quase sempre a necessidade de adicionar legendas que permitam distinguir as funções desenhadas*
- De forma a ilustrarmos a sobreposição de funções, criemos uma segunda função que passe a ter como resultado não o resto, mas o quociente da divisão inteira por 5

```
z=[v//5 for v in x]
```

```
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3]
```

- Invocando a função plot() para ambas as funções y(x) e z(x), obtém-se a sobreposição esperada

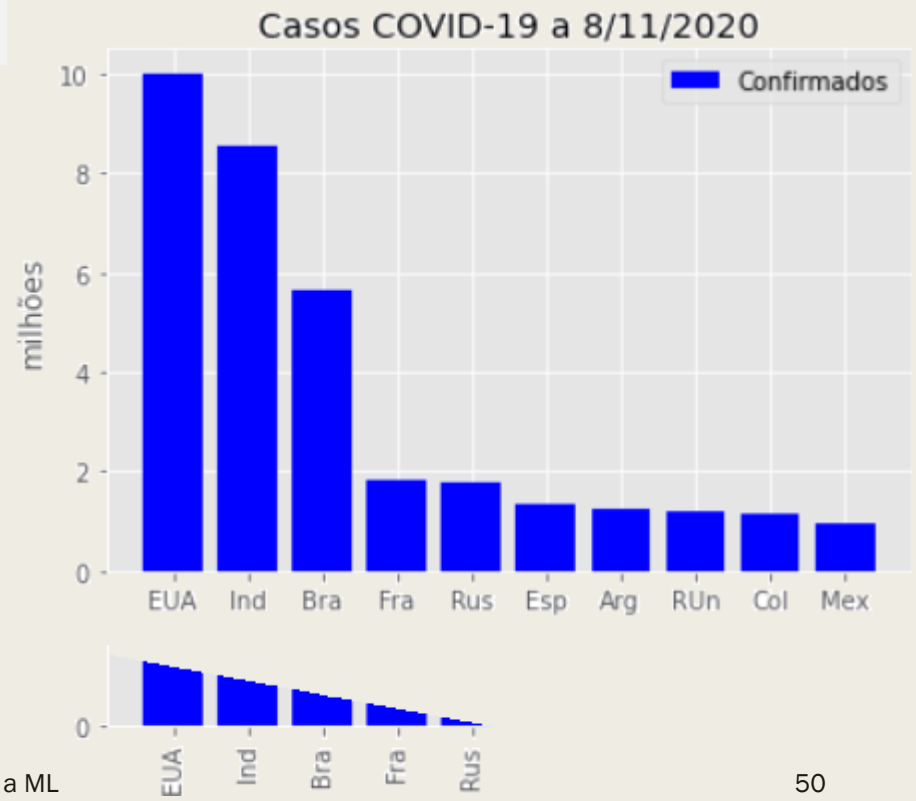
```
plt.plot(x,y, label='Resto')
plt.plot(x,z, label='Quociente')

plt.title("Divisão inteira por 5")
plt.xlabel("Dividendo")

plt.legend()
```

Para a adição de legendas usa-se a função legend() e o parâmetro label da função plot()

Gráfico de barras

- Os gráficos de barras são úteis para comparar dados, não para representar funções
  - *Por exemplo, pretendendo-se comparar o número de contágios COVID-19 a 8/nov/2020, entre os 10 países mais afetados, um gráfico de barras, gerado pela função bar(), será o indicado*

```
paises=['EUA', 'Ind', 'Bra', 'Fra', 'Rus', 'Esp', 'Arg', 'RUn', 'Col', 'Mex']
casos=[10.02, 8.55, 5.66, 1.84, 1.76, 1.33, 1.24, 1.2, 1.14, 0.97]
plt.bar(paises,casos, color = 'b', label='Confirmados') #'b'- blue
plt.title("Casos COVID-19 a 8/11/2020")
plt.ylabel("milhões")
plt.legend()
```



No presente exemplo houve o cuidado de usar para os rótulos do eixo do x apenas as 3 iniciais do nome dos países

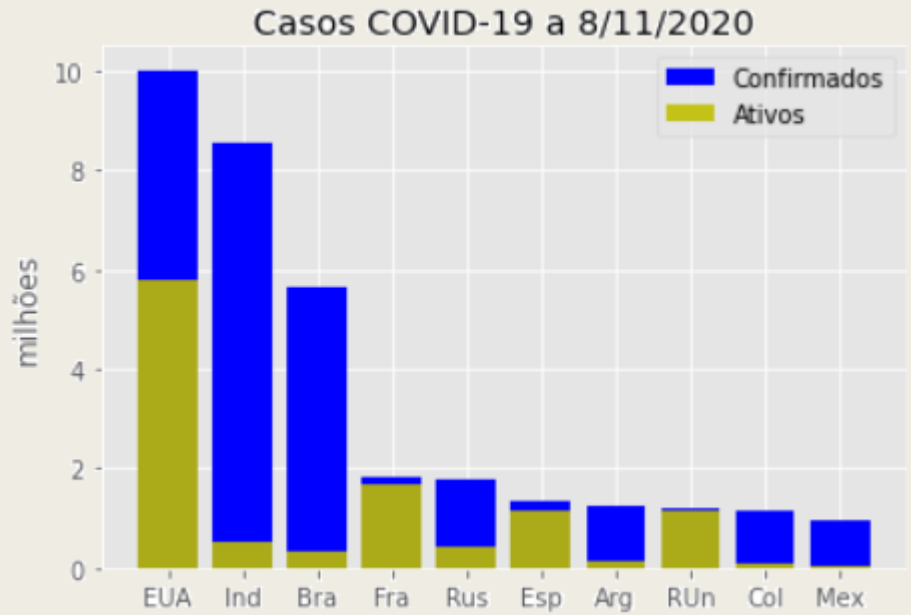
- Mas, não raras as vezes, torna-se difícil incluir todos os rótulos no eixo do x
- Nesses casos, a solução passaria por dispor verticalmente esses rótulos através da função xticks()

```
plt.xticks(rotation='vertical')
```

Sobreposição de gráficos de barras

- Tal como com os gráficos de linhas, é também possível sobrepor 2 ou mais gráficos de barras
  - *Continuando com o exemplo anterior, pode ser interessante mostrar o gráfico dos números de casos ainda ativos em cada país, sobreposto ao gráfico de casos confirmados*
    - com essa sobreposição, conseguimos uma fácil perceção da proporção dos contágios que se mantêm ativos em cada país
  - *Para se obter essa sobreposição, basta acrescentar ao código anterior o código que se segue, responsável por gerar o 2º gráfico de barras:*

```
ativos=[5.82, 0.51, 0.35, 1.66, 0.41, 1.14, 0.15, 1.14, 0.07, 0.05]
plt.bar(paises,ativos, color = 'y', label='Ativos', alpha = 0.9)
plt.legend()
```



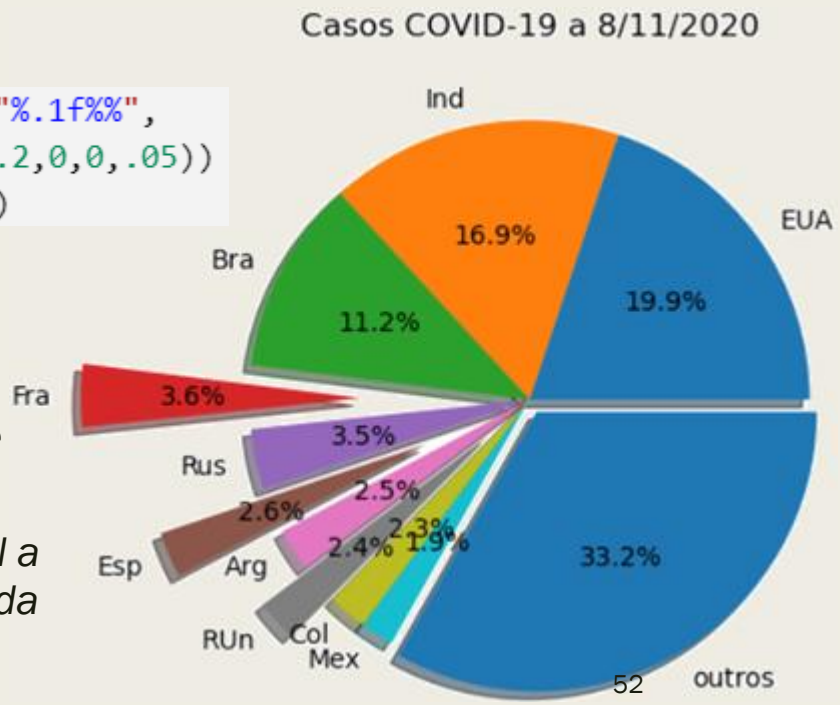
O parâmetro alpha regula o nível de opacidade, entre 0 e 1. Revela-se particularmente útil quando algumas das barras do 2º gráfico superem as do 1º (nesse caso, se não baixássemos o alpha para um nível de transparência aceitável, as 1as barras ficariam completamente tapadas pelas 2as)

Gráfico circular

- Os gráficos circulares são úteis para representar proporções de um todo
  - *por isso, o somatório dos vários valores a representar deve perfazer 100%*
- Para ilustramos a construção deste tipo de gráfico, podemos continuar com o exemplo dos contágios do COVID-19
  - *De facto faz sentido usarmos o gráfico circular para melhor percebermos que proporção do total de contágios cabe a cada país*
  - *E uma vez que neste tipo de gráfico representamos todo o universo de casos, devemos começar por adicionar mais um elemento à serie de países, que represente todos os casos do resto do mundo (16.74 milhões)*

```
paises.append('outros')
casos.append(16.74)
```

```
plt.pie(casos, labels=paises, autopct="%.1f%%",
shadow=True, explode=(0,0,0,.6,0,.4,0,.2,0,0,.05))
plt.title("Casos COVID-19 a 8/11/2020")
```



- O gráfico circular é gerado pela função pie()
  - *Com o parâmetro 'autopct' indicamos, através duma string de formatação, de que forma é mostrado o valor de cada fatia*
  - *Com o parâmetro 'explode', indicamos qual a percentagem (de 0 a 1) de destaque de cada fatia.*

Gráfico de dispersão

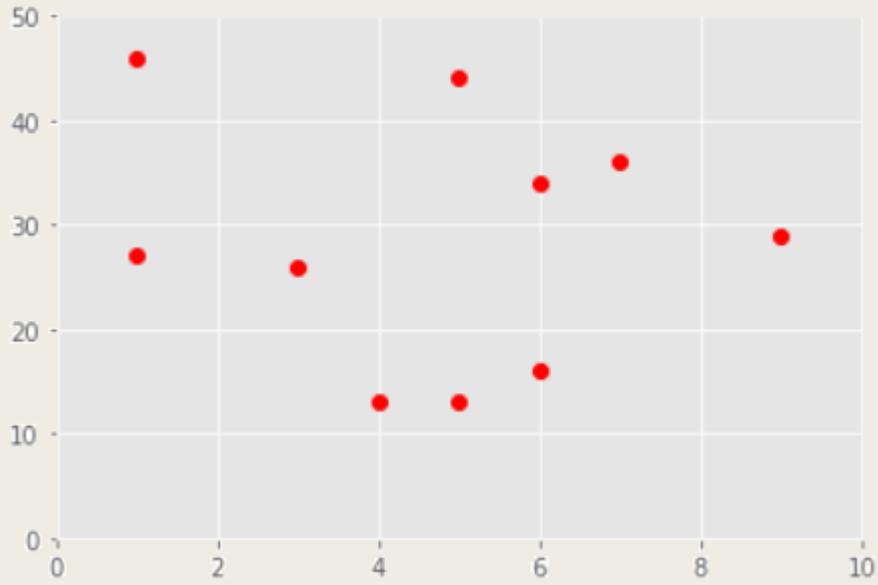
- Um gráfico de dispersão usa pontos (ou outro tipo de marcações) para representar num plano bidimensional as coordenadas X e Y expressas por duas variáveis, ilustrando de que forma estão as duas relacionadas
  - *Este tipo de gráfico é também produzido no Matplotlib pela função plot(), bastando para isso passar-lhe como parâmetro uma string com a formatação que pretendemos para o marcador*
- Para ilustrar esse tipo de gráfico, comecemos por criar as duas coordenadas x e y

```
x=np.random.random_integers(1,9,10); print('Abcissas:', x)
y=np.random.random_integers(1,49,10); print('Ordenadas:', y)
```

```
Abcissas: [1 4 5 1 7 6 9 6 3 5]
Ordenadas: [46 13 44 27 36 16 29 34 26 13]
```

E invoquemos então a função plot()

```
plt.plot(x,y,'or')
  # or: red (r) circle (o) marker
plt.axis([0,10,0,50])
  # [xmin, xmax, ymin, ymax]
```



  - *Se não incluíssemos a string 'ro', seria desenhado um gráfico de linhas, o que, neste caso, não faria muito sentido*

Sobreposição de gráficos de dispersão

- Também os gráficos de dispersão podem ser sobrepostos, chamando sucessivas vezes a função plot() ou até mesmo através duma única invocação
  - *De forma a ilustramos esta segunda opção, criemos as coordenadas para dois gráficos adicionais*

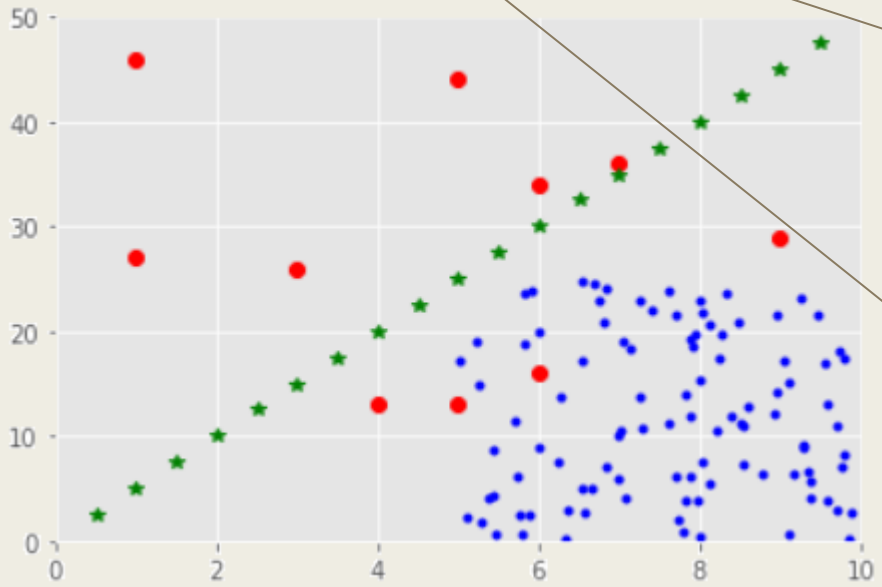
```
x1=np.arange(1,20)/2
y1=x1*5
x2=np.random.random(100)*5+5
y2=np.random.random(100)*25
```

100 valores aleatórios entre 5 e 10

100 valores aleatórios entre 0 e 25

  - *Para os três gráficos surgirem sobrepostos é suficiente passar para a função plot() as coordenadas e string do marcador de cada um deles, em sequência*

```
plt.plot(x,y,'or', x1,y1,'*g', x2,y2,'.b')
plt.axis([0,10,0,50])
```



'.b': pontos azuis

'*g': estrelas verdes

Existem muitas outras pequenas strings para representar outras formas e cores dos marcadores (consultar help(plt.plot))

Para fixar a escala dos eixos X e Y a valores específicos. Doutra forma, os limites dos eixos adaptar-se-iam aos valores apresentados.

Gráficos de dispersão com a função scatter()

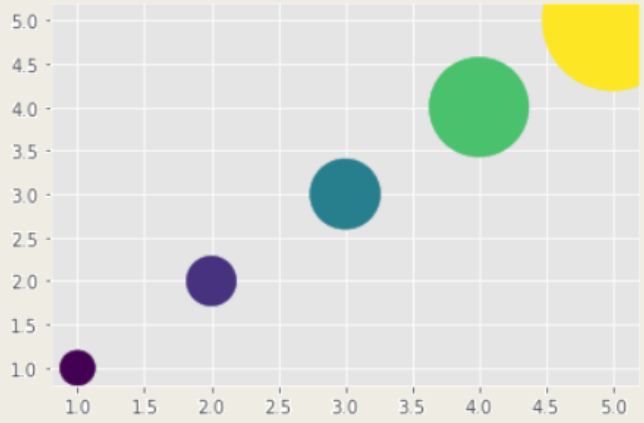
- Gráficos de dispersão mais sofisticados podem ser conseguidos com a função scatter()
  - *Usada quando se pretende marcadores de tamanho e cor variável*
  - *Enquanto a função scatter() desenha pontos sem linhas a interliga-los, a função plot() pode ou não desenhar linhas, dependendo dos argumentos que forem usados*
- A título de exemplo, imprimamos a série de coordenadas (x2,y2) com a função scatter()
  - *Para o efeito, começamos por definir, de forma aleatória, as sequências de cores e tamanhos a usar nos 100 marcadores*

```
cores=np.random.random(100)
area=(np.random.random(100)*30)**2

plt.scatter(x2,y2,c=cores,s=area,alpha=0.5)
```

Estes gráficos, por vezes, também são designados *bubble charts, e percebe-se porquê…*

- Para um mais fácil entendimento, repare-se neste exemplo mais simples

```
x3=y3=[1, 2, 3, 4, 5]
cores=[.2, .3, .5, .7, .9]
areas=[400, 800, 1600, 3200, 6400]
plt.scatter(x3,y3,c=cores,s=areas)
```



Subplots (combinar sem sobrepor)

- Também é possível combinar sem sobrepor vários gráficos numa mesma figura (janela gráfica), desenhando-os lado a lado (na horizontal ou na vertical) em tamanho mais pequeno
  - *É a função subplot() que dimensiona e posiciona cada um desses pequenos gráficos*
  - *Mais concretamente, a função subplot() cria a subárea retangular onde irá surgir o próximo gráfico a ser gerado, com a dimensão e a posição especificadas através dum parâmetro formado por três dígitos:*
    - os dois primeiros estabelecem o número de linhas e colunas de uma grelha invisível que define as possíveis subáreas onde será impresso o próximo gráfico,
    - o terceiro dígito indica a posição da subárea dessa grelha onde irá surgir o próximo gráfico.
  - *Por exemplo, caso se começasse com a instrução*

```
subplot(235)
```

seleciona a 5ª subárea definida por uma grelha de 2 linhas e 3 colunas

o próximo gráfico iria aparecer com a dimensão e na posição indicadas na figura que se segue

- A mesma indicação poderia ser dada, passando cada um dos dígitos como parâmetro separado: suplot(2,3,5)
- Esta opção teria a vantagem de não limitar a grelha à dimensão 3x3

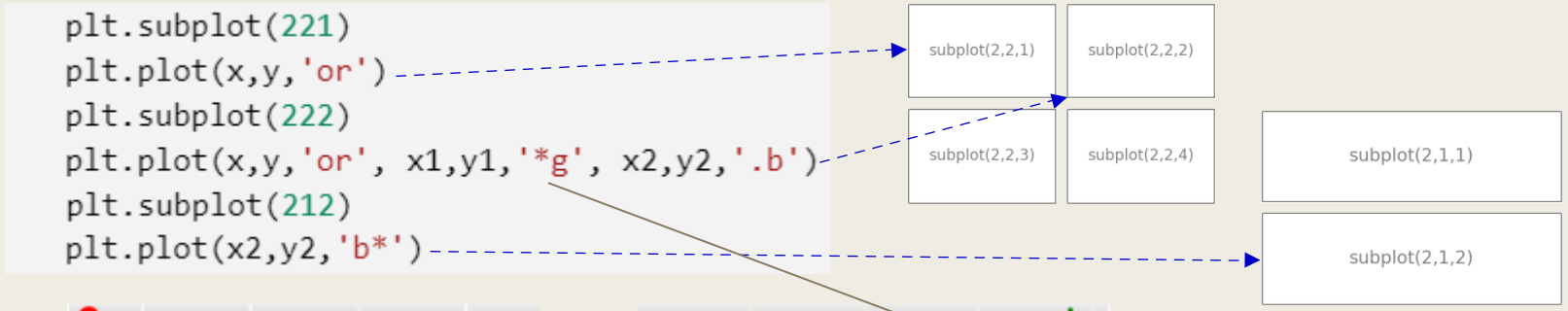
Subplots (possíveis subáreas)

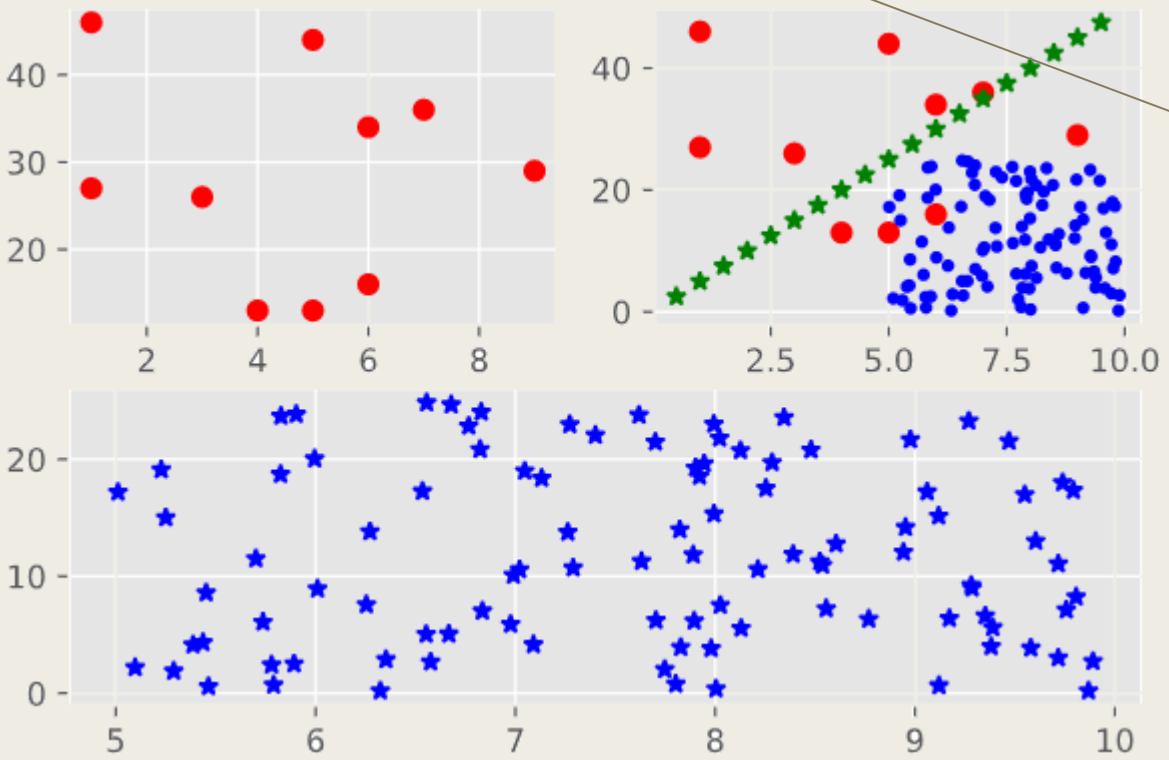
- Seguem-se ilustrações das subáreas selecionadas noutros possíveis *subplots*

Subplots (exemplificando)

- Com as sequências de valores produzidas anteriormente, vamos exemplificar o uso da função subplot(), desenhando 3 gráficos combinados numa mesma figura

```
plt.subplot(221)
plt.plot(x,y,'or')
plt.subplot(222)
plt.plot(x,y,'or', x1,y1,'*g', x2,y2,'.b')
plt.subplot(212)
plt.plot(x2,y2,'b*')
```





Repare-se que nada impede que num dos *subplots* sejam impressos vários gráficos sobrepostos

A função axes() é similar à subplot()

- permite, no entanto, posicionar os subgráficos em qualquer posição, permitindo inclusive sobreposição

A biblioteca Seaborn

- A Seaborn é uma biblioteca complementar de visualização de dados baseada na Matplotlib
  - *Fornece-nos um conjunto de ferramentas para contruir gráficos estatísticos em Python*
  - *É mais um excelente auxílio para a exploração e compreensão dos dados*
  - *Encontra-se perfeitamente integrada com as estruturas de dados do Pandas – as suas funções de criação de gráficos operam diretamente sobre os DataFrames*
  - *Com a sua interface de alto nível conseguimos produzir gráficos sofisticados e atrativos com menos esforço do que com o Matplotlib*
  - *E mais importante, permite facilmente visualizar a relação entre múltiplas variáveis (colunas), quer numéricas, quer categóricas*
    - Na programação estamos habituados a classificar esta segunda categoria de variáveis como variáveis de tipo enumerado
    - Na Data Science, quer as variáveis (colunas dos *datasets*) do tipo string, quer as numéricas que representem um código ou um id (que sirvam apenas para identificar ou qualificar algo, não para quantificar), são, por norma, interpretadas como categóricas
- De forma a ilustrarmos o tipo de gráficos que podem ser produzidos com a Seaborn, vamos aproveitar o DataFrame alunos usado anteriormente, para, a partir dele, visualizar diferentes combinações de variáveis numéricas e categóricas
  - *Com o objetivo de diversificar as demostrações, completou-se ainda mais a DataFrame alunos, acrescentando-lhe mais algumas linhas, a coluna 'género' e convertendo a sua última coluna para inteiro (com 0 a significar reprovado e 1 aprovado)*

O DataFrame alunos (para visualização)

numero	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
30000	Tó	M	ordin	20	28	False	19.2	1
31234	Ana	F	trab	20	20	False	12.2	0
33333	Rui	M	erasm	25	3	True	5.3	0
40000	Gil	M	ordin	27	28	False	15.7	1
44444	Zé	M	ordin	23	17	True	15.9	1
34567	Ivo	M	trab	21	27	False	14.0	1
35000	José	M	ordin	21	28	False	14.0	1
36000	Joel	M	ordin	22	26	True	12.0	1
37000	Bia	F	ordin	20	27	True	9.0	0
38000	Luís	M	erasm	20	25	False	8.0	0
39000	Rita	F	erasm	21	27	False	18.0	1
41000	Lara	F	trab	23	5	True	7.0	0
42000	Sara	F	trab	27	3	False	10.0	1

Visualizar 1 variável numérica

- Caso prendamos, por exemplo, analisar a distribuição da variável numérica notaIA, podemos usar a função seaborn.distplot()

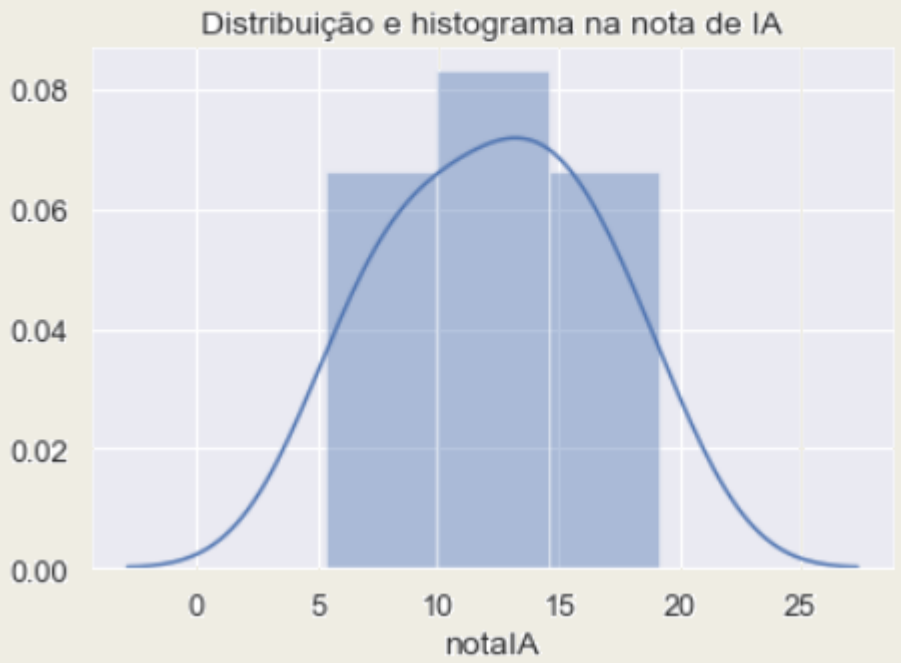
  (após importarmos o respetivo package, como é óbvio, e também o Pandas para podermos usar o DataFrame alunos)

```
import seaborn as sns
import pandas as pd
```

```
sns.set_style('darkgrid')
g=sns.distplot(alunos.notaIA)
g.set_title('Distribuição e histograma na nota de IA');
```

Seleciona um estilo de gráfico predefinido

Repare-se que podemos fornecer às funções da Seaborn as próprias colunas do nosso DataFrame

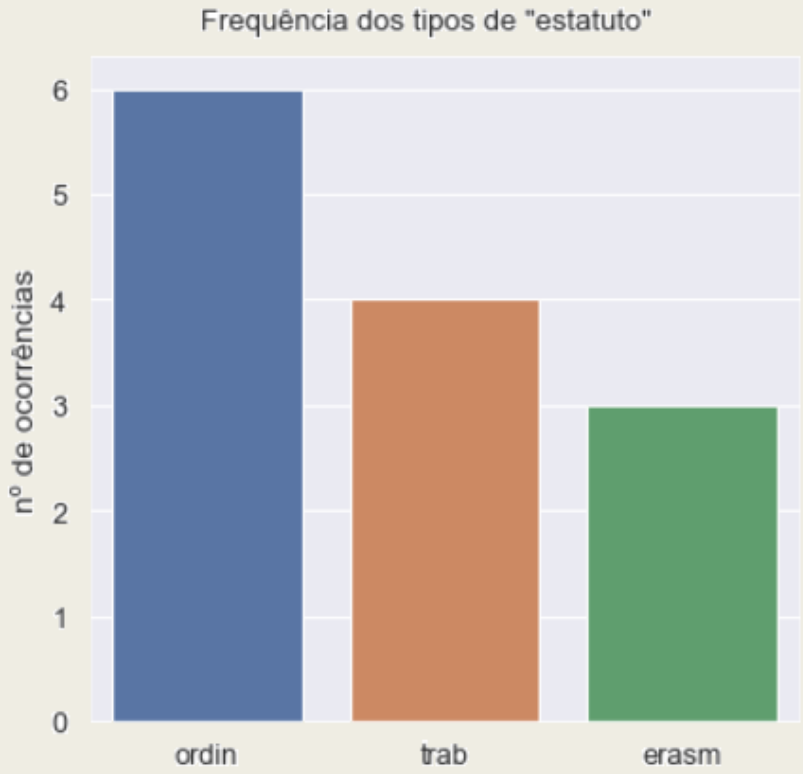


- Até com um minúsculo *dataset* de dados simulados, esta sua variável parece seguir uma distribuição normal...

Visualizar 1 variável categórica

- Se pretendermos analisar uma variável categórica, podemos, através da função seaborn.catplot(), mostrar a frequência com que ocorrem cada um dos diferentes valores dessa variável
  - *Mostremos, por exemplo, o nº de alunos por cada tipo de frequência (que, para não emaranhar a linguagem, também designamos "estatuto")*

```
g = sns.catplot(x="freq",kind='count', data=alunos)
g.fig.suptitle('Frequência dos tipos de "estatuto"', y=1.02, size=13)
g.set_axis_labels("", "nº de ocorrências");
```



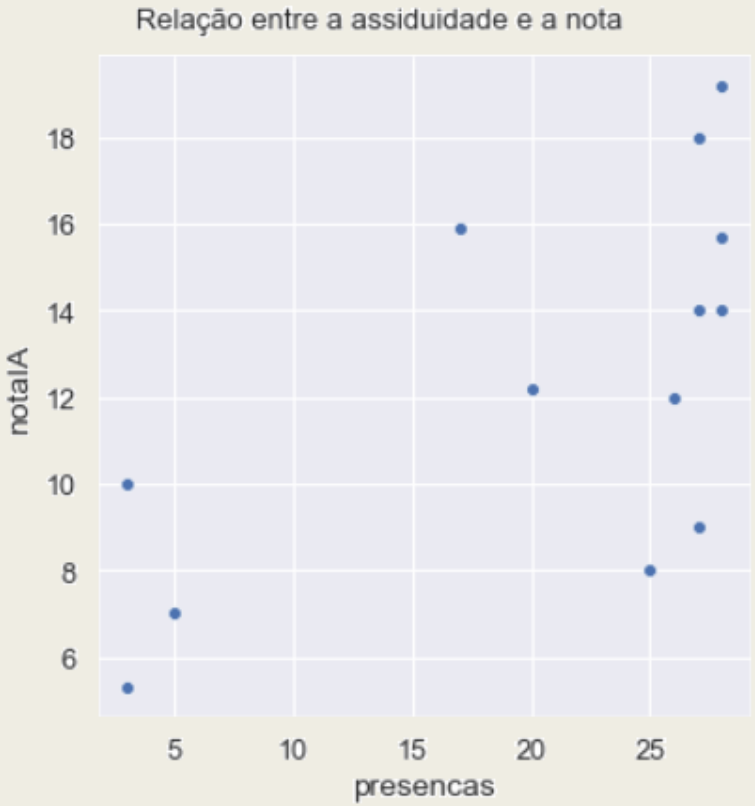
- Como esperado, os ordinários são em maior número…
- Mas também os restantes tipos de frequência começam a estar muito bem representados

(*dataset* com dados "inventados", não esquecer.)

Visualizar 2 variáveis numéricas

- Para visualizar a relação entre duas variáveis numéricas, pode ser usada a função seaborn.relplot()
  - *Mostremos, por exemplo, o gráfico de dispersão das variáveis 'presenças' e 'notaIA', para conseguirmos uma rápida perceção da influência que a assiduidade do aluno possa ter no seu desempenho*

```
g = sns.relplot(x="presencas",y="notaIA",data=alunos,kind='scatter');
g.fig.suptitle('Relação entre a assiduidade e a nota', y=1.02, size=13);
```

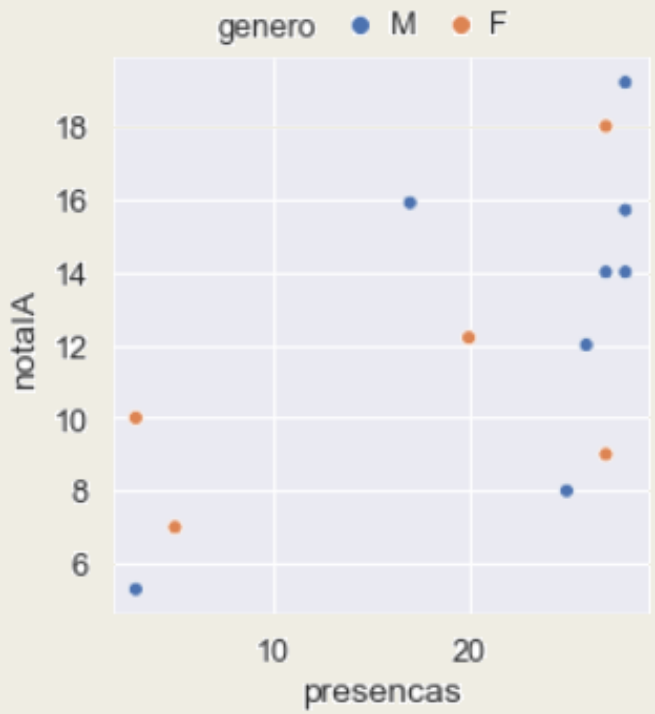


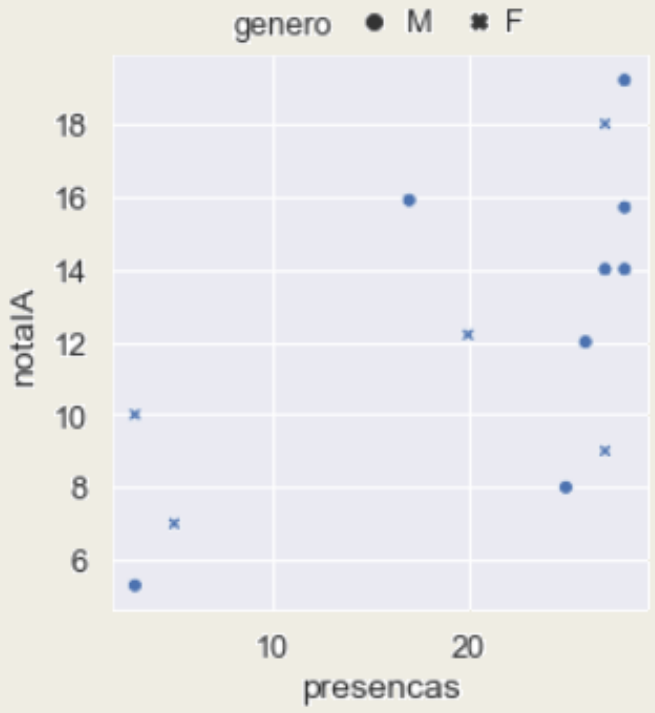
- Até mesmo com estes dados simulados, se percebe a grande importância da assiduidade na nota final... ☺

Visualizar 2 variáveis numéricas e 1 categórica

- Com duas variáveis numéricas e uma categórica, podemos usar esta última para diferenciar os pontos (marcadores) do gráfico 2D das duas variáveis numéricas
  - *A dimensão adicional que esta terceira variável introduz pode de facto ser refletida na cor ou na forma dos pontos do gráfico 2D*
- Para distinguirmos o género no gráfico anterior, acrescentemos-lhe essa variável categórica

```
g = sns.relplot(x="presencas",y="notaIA",data=alunos,kind='scatter',hue='genero')
g = sns.relplot(x="presencas",y="notaIA",data=alunos,kind='scatter',style='genero')
```

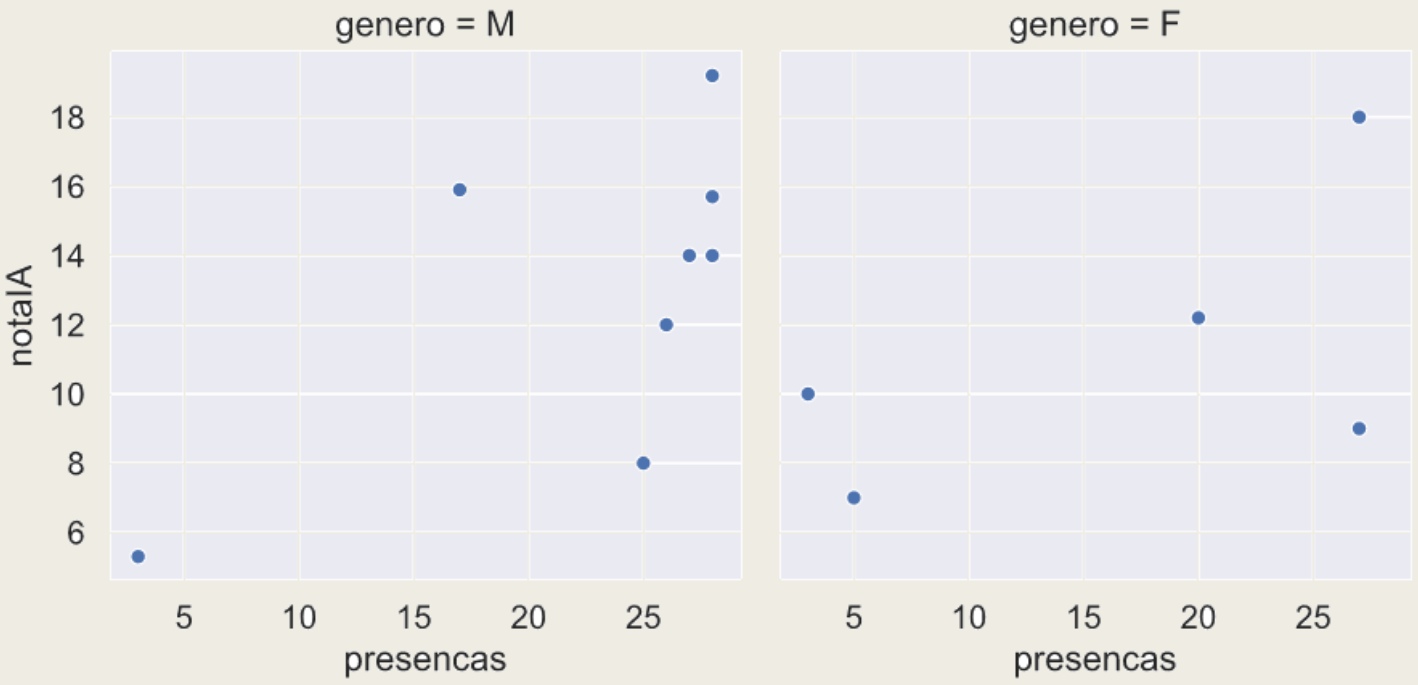




Visualizar 2 variáveis numéricas e 1 categórica

- Uma terceira opção, é mostrar em gráficos separados as observações das diferentes categorias

```
g = sns.relplot(x="presencas",y="notaIA",data=alunos,kind='scatter',col='genero');
```



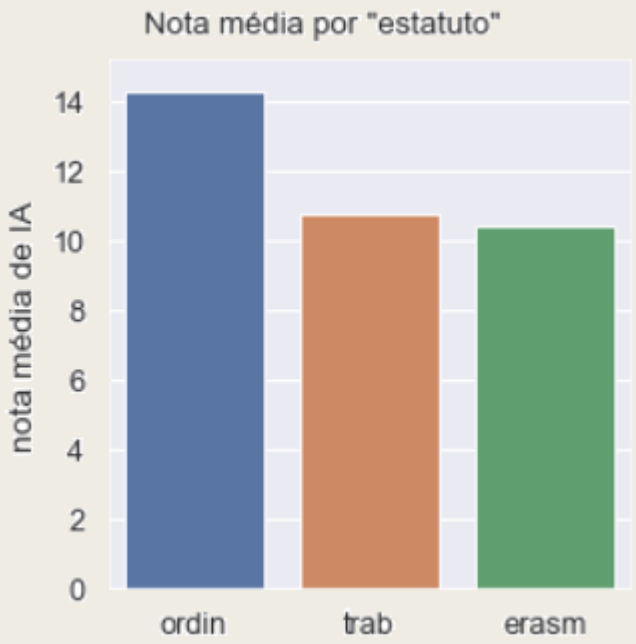
Visualizar 1 variável numérica e 1 categórica

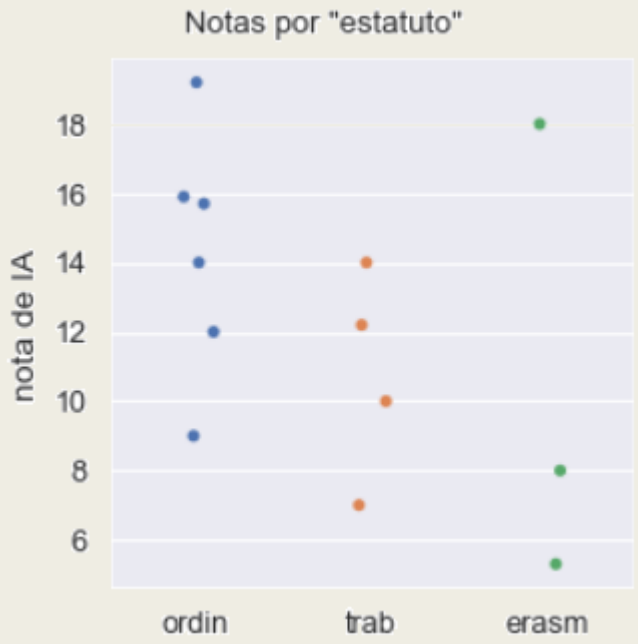
- Com uma variável numérica e outra categórica, é possível contruir vários tipos de gráficos, como por exemplo, gráficos de barras e *strip plots*

```
g = sns.catplot(x='freq',y='notaIA',kind='bar',ci=False,data=alunos)
g.fig.suptitle('Nota média por "estatuto"', y=1.02, size=13)
g.set_axis_labels("", "nota média de IA");
```

```
g = sns.catplot(x='freq',y='notaIA',kind='strip',ci=False,data=alunos)
g.fig.suptitle('Notas por "estatuto"', y=1.02, size=13)
g.set_axis_labels("", "nota de IA");
```

O gráfico de barras mostra os valores numéricos médios para cada categoria





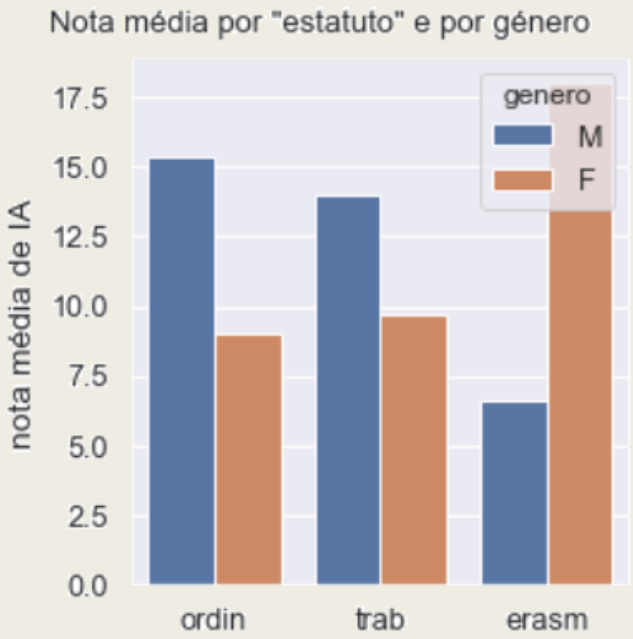
O *strip chart* mostra os valores numéricos da cada categoria em faixas separadas

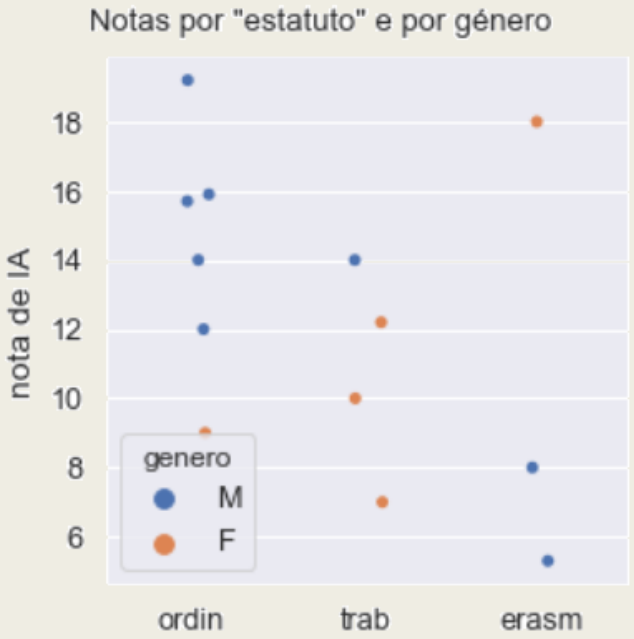
Visualizar 1 variável numérica e 2 categóricas

- Com uma variável numérica e duas categóricas, podemos acomodar a variável categórica adicional a qualquer um dos gráficos referidos no diapositivo anterior,
  - *quer através do parâmetro 'hue', refletindo-se essa categoria na cor,*
  - *quer através do parâmetro do parâmetro 'col', passando as observações das várias categorias a ser mostradas em gráficos separados*

```
g = sns.catplot(x='freq',y='notaIA',kind='bar',ci=False,data=alunos,hue='genero')
g.fig.suptitle('Nota média por "estatuto" e por género')
g.set_axis_labels("", "nota média de IA");
```

```
g = sns.catplot(x='freq',y='notaIA',kind='strip',ci=False,data=alunos,hue='genero')
g.fig.suptitle('Notas por "estatuto" e por género')
g.set_axis_labels("", "nota de IA");
```

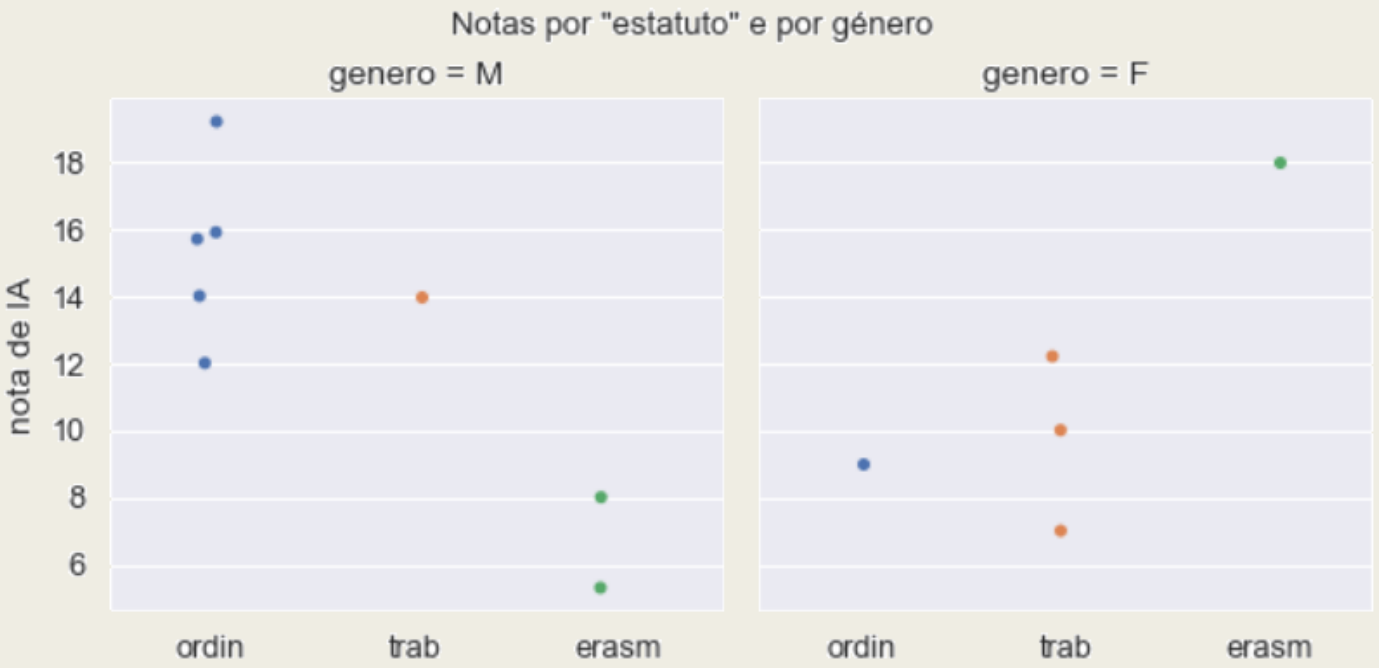




Visualizar 1 variável numérica e 2 categóricas

- As observações dos dois tipos de género podem ser apresentadas em dois gráficos *strip* separados, escolhendo a opção col='genero'

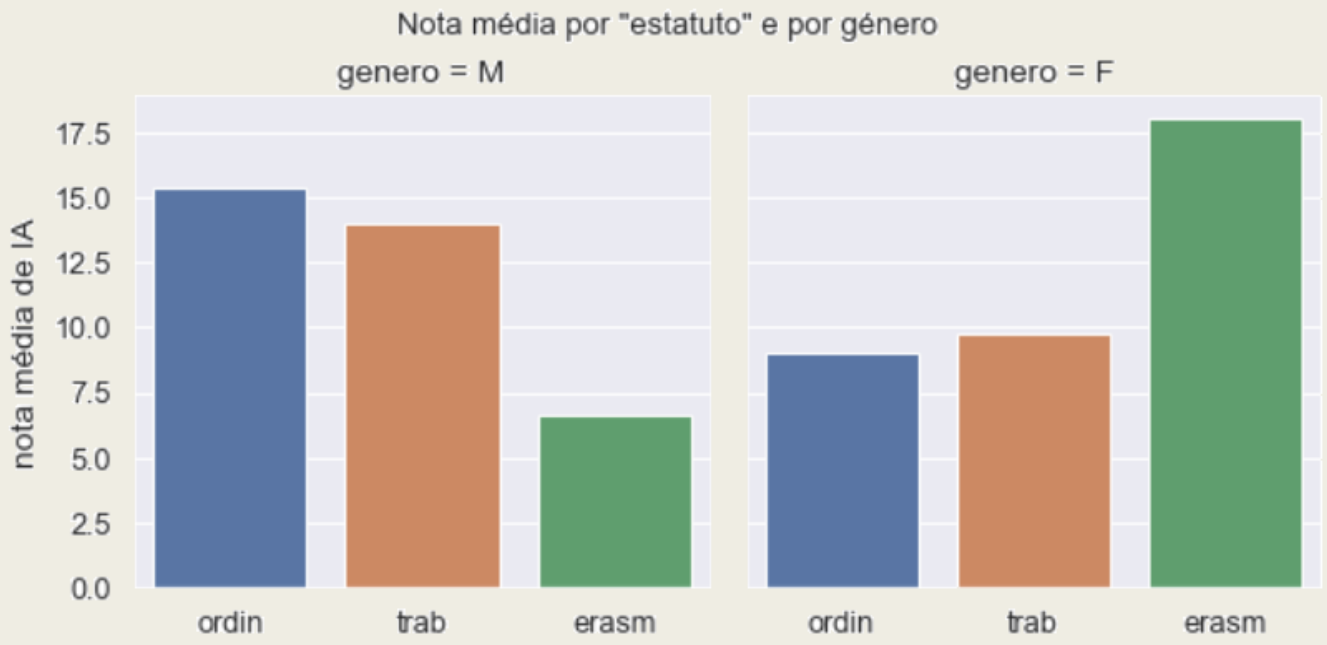
```
g = sns.catplot(x='freq',y='notaIA',kind='strip',ci=False,data=alunos, col='genero')
g.fig.suptitle('Notas por "estatuto" e por género')
g.set_axis_labels("", "nota de IA");
```



Visualizar 1 variável numérica e 2 categóricas

- As observações dos dois tipos de género podem também ser apresentadas em dois gráficos de barras separados

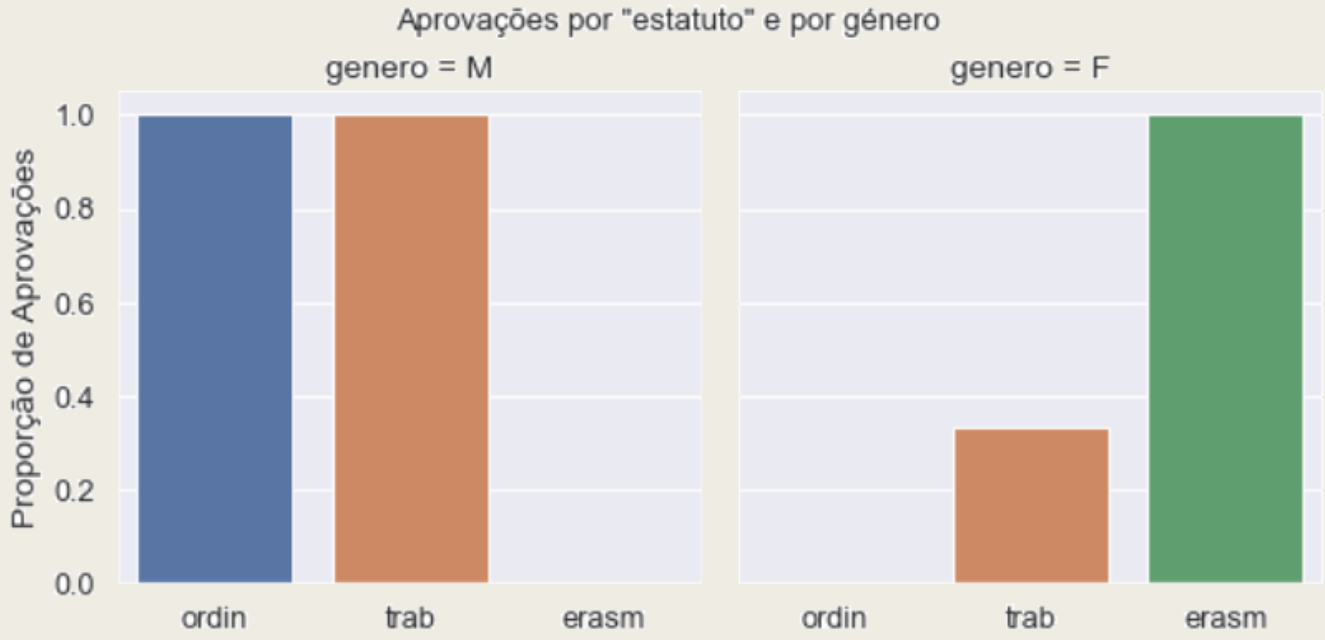
```
g = sns.catplot(x='freq',y='notaIA',kind='bar',ci=False,data=alunos,col='genero')
g.fig.suptitle('Nota média por "estatuto" e por género')
g.set_axis_labels("", "nota média de IA");
```



Visualizar 1 variável numérica e 2 categóricas

- Se, alternativamente, usarmos para a variável numérica do gráfico anterior a coluna 'aprovado' (de 0s e 1s), em vez da coluna 'notaIA', os gráficos obtidos passam a ter uma interpretação ainda mais interessante
  - *Repare-se que, neste caso, o valor médio de 0s e 1s acaba por nos dar outra informação mais relevante*
    - Dá-nos, precisamente, a proporção de 1s, que neste caso traduz o rácio de aprovações

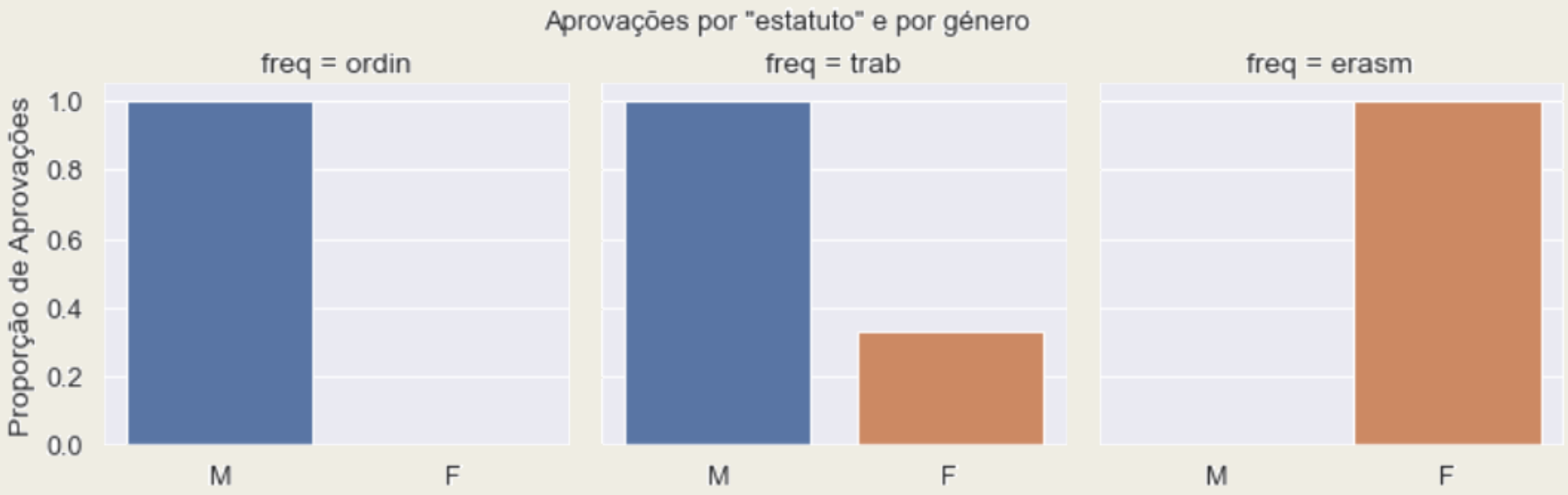
```
g = sns.catplot(x='freq',y='aprovado',kind='bar',ci=False,data=alunos,col='genero')
g.fig.suptitle('Aprovações por "estatuto" e por género')
g.set_axis_labels("", "Proporção de Aprovações");
```



Visualizar 1 variável numérica e 2 categóricas

- E, naturalmente, podemos também trocar o papel desempenhado pelas 2 variáveis categóricas

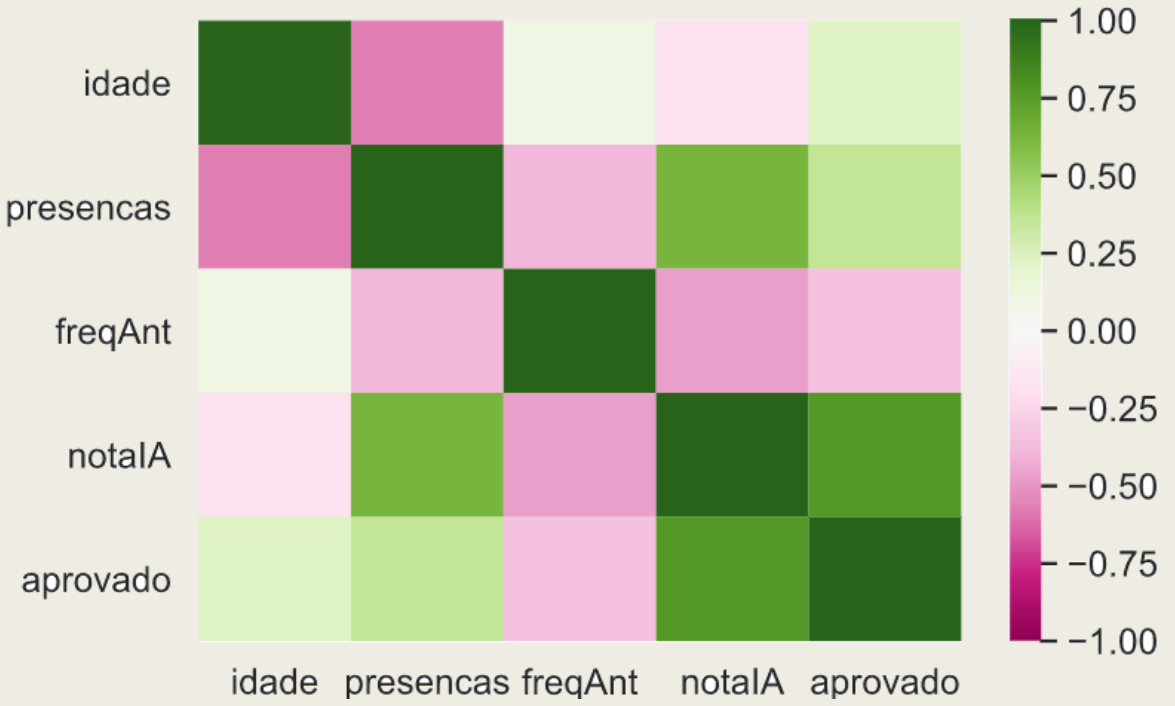
```
g = sns.catplot(x='genero',y='aprovado',kind='bar',ci=False,data=alunos,col='freq')
g.fig.suptitle('Aprovações por "estatuto" e por género')
g.set_axis_labels("", "Proporção de Aprovações");
```



Visualizar 3 ou mais variáveis numéricas

- Para visualizarmos 3 ou mais variáveis não categóricas, dispomos ainda da função seaborn.heatmap(), em que a variação duma dada métrica é apresentada com diferentes tonalidades cromáticas
  - *Uma das métricas mais usadas é a função de correlação*
  - *Num único gráfico deste tipo é então possível visualizar a relação de cada variável com cada uma das outras*
  - *Os pares de variáveis mais correlacionadas são representadas por tonalidades mais extremas, dependendo sempre da escala que for escolhida*

```
g = sns.heatmap(alunos.corr(), vmin=-1, vmax=1, cmap="PiYG");
```



Com a escala de tonalidades escolhida neste exemplo, a verde escuro estão representadas as variáveis mais correlacionadas positivamente, e a rosa escuro as mais correlacionadas negativamente

Mais sobre a Seaborn

- Embora consigamos com o Seaborn produzir rápida e facilmente gráficos complexos e atrativos, a Seaborn põe ao nosso dispor muitas opções de personalização, para que as apresentações gráficas resultem ainda mais perfeitas
  - *Esse será um tópico mais avançado que deverá ser explorado logo após se conseguir algum domino dos aspetos essenciais tratados aqui*
  - *Para informações mais detalhadas consultar o website oficial: https://seaborn.pydata.org/*

- A Seaborn traz também consigo um conjunto de *datasets* que podemos carregar através da função load_dataset(), e consultar o seus nomes com a função get_dataset_names()

```
sns.get_dataset_names()
```

```
['anagrams', 'anscombe', 'attention', 'brain_networks', 'car_crashes',
'diamonds', 'dots', 'exercise', 'flights', 'fmri', 'gammas', 'geyser',
'iris', 'mpg', 'penguins', 'planets', 'tips', 'titanic']
```

  - *Um deles, por exemplo, tem os dados dos passageiros que embarcaram no Titanic*

```
titanic = sns.load_dataset('titanic')
```

```
titanic.head(3)
```

	survived	sex	age	sibsp	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	male	22.0	1	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	female	38.0	1	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	female	26.0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True