PACKAGES FOR MACHINE LEARNING

Matplotlib & Seaborn Package

The Matplotlib package – a first vision[1]

- In the world of ML, graphical visualisation of information plays a vital role
  - *It is through graphical visualisation that we may often quickly identify essential characteristics and relationships between data*
- Matplotlib is currently Python's most scientific visualization package (follows the style of MATLAB graphic presentations)
  - *its interface is not the simplest, but it is a powerful library for creating a wide variety of good-quality graphics*
- To use a MATLAB-style interface in the production of graphics, one begins by importing the pyplot module from Matplotlib, as plt:

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

  - *Then, creating, for example, a one-dimensional NumPy array with the aboriginal data (50 values equally spaced between 0 and 2π)*

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

  - *and a second array with the order data (sine function)*

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

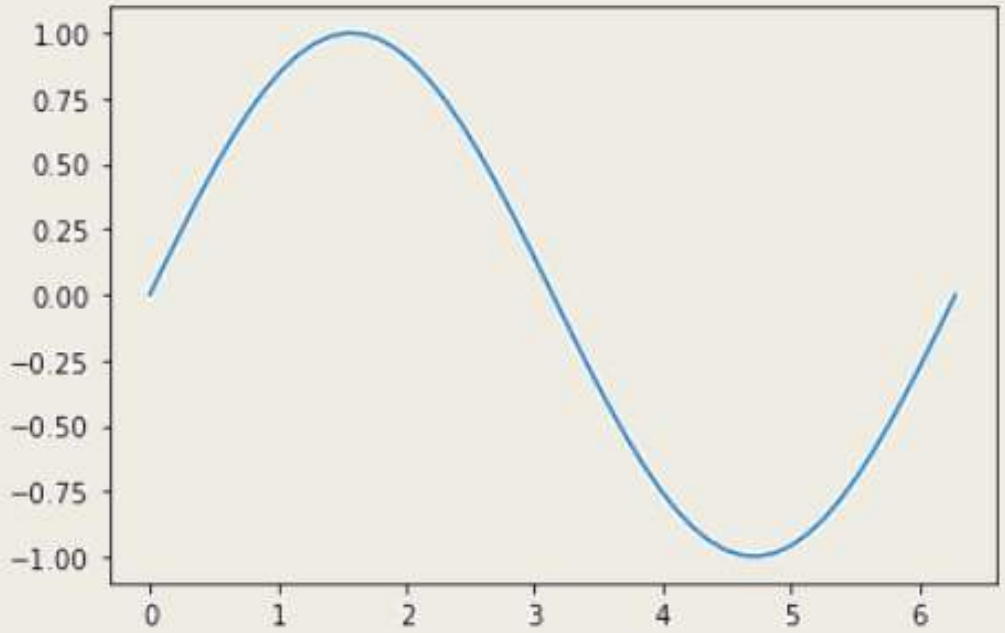
  - *we may easily draw the respective graph with plot() function of the plt module*

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

[1] *With content adapted from "A Whirlwind Tour of Python", of Jake Vanderplas, O'Reilly, 2016 (with CC0 licensing)*

The Matplotlib package – a first vision[1]

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



- By clicking on the chart, within the programming environment, we have access to some basic operations, such as zooming in and convert other formats.

In this brief presentation, a very simple example of creating a chart in Matplotlib proved to be a very simple example of creating a graph in Matplotlib

- *many other more elaborate chart types can be explored by browsing the Matplotlib online gallery*
  - https://matplotlib.org/gallery.html

Unraveling Matplotlib a little more

- With Matplotlib, complex, good quality 2D graphics can be easily generated,
  - *and when integrated into the Jupyter Notebook becomes an even more powerful tool in the world of ML*
- Your pyplot module, in turn, provides us with an interface to the Matplotlib library itself
  - *makes chart production a simpler task by providing us with resources to control line styles, font properties, formatting of axes, subtitles, size and positioning of graphics, etc.*
  - *together with NumPy, gives us an alternative environment to MatLab, with many commands and similar features*
- As it is with the pyplot that we will interact, it is this module that we should import
  - *pyplot is imported as plt, by convention, and can take this import two alternative forms:*

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

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

Line chart

- A line chart can be generated using simply two lists, the 1st with the values for the x-axis and the 2nd with the values for the y axis

```
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
```

As exemplified, the title, axle name and other specifications must be indicated separately, but next to the plot() function – the main responsible for the production of the graph

Predefined graphics styles

- So we don't waste a lot of time setting up the graphs, Matplolib provides us with a set of preset styles
  - *one of these styles is 'ggplot', which is based on the R language preview package*
  - *To choose a style, we use the use() function of the style module:*

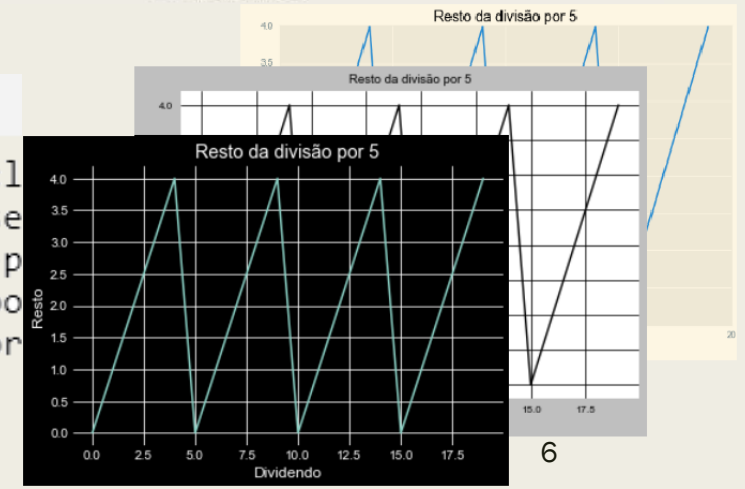
```
from matplotlib import style
style.use("ggplot")
```

  - *If we had chosen this style before the production of the chart, it would have the appearance that is.*

- Many other preset styles are available

```
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']
```



Overlap functions on the same chart

- Invoking the plot() function successively, multiple functions are drawn on the same chart
  - *In this type of graphics, with overlapping functions, there is almost always the need to add subtitles that allow us to distinguish the functions drawn*
- In order to illustrate the overlap of functions, let us create a second function that will result in not the rest, but the quotient of the entire division by 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]
```

- By invoking the plot() function for both the functions y(x) and z(x), the expected overlap is obtained

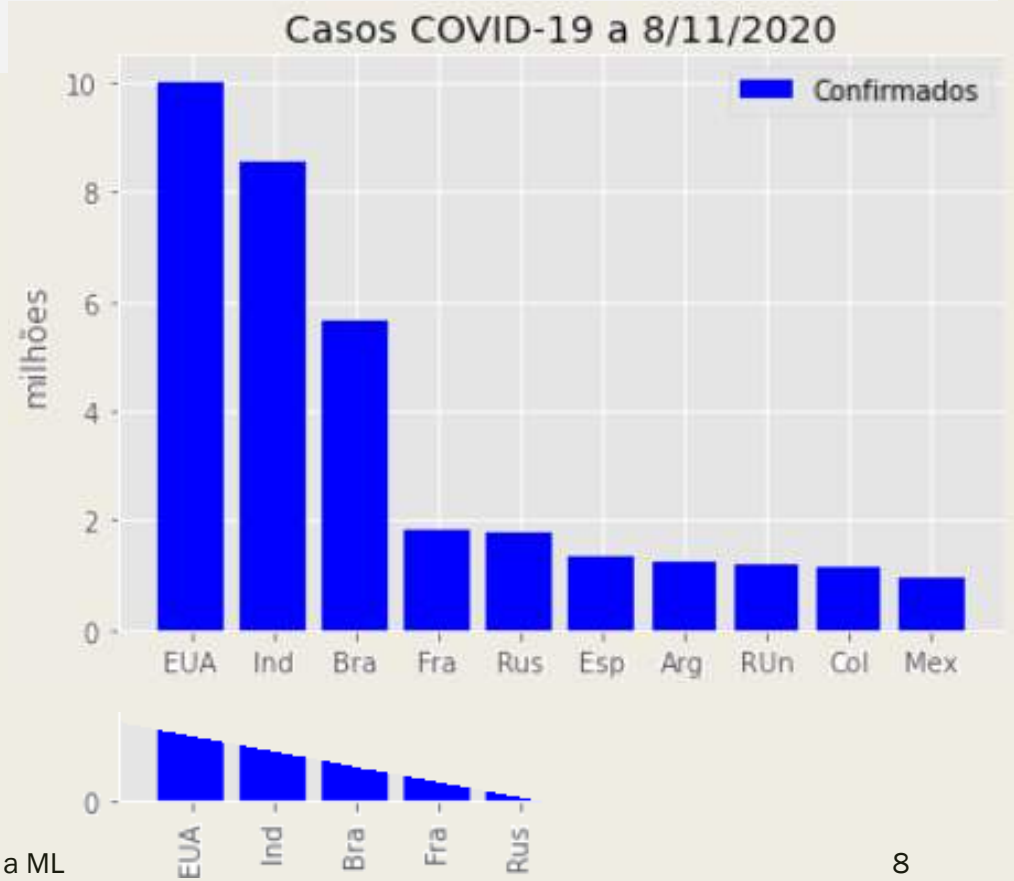
```
plt.plot(x,y, label='Resto')
plt.plot(x,z, label='Quociente')

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

plt.legend()
```

For the addition of subtitles, the legend() function and the label parameter of the plot() function are used

Bar chart

- Bar charts are useful for comparing data, not for representing functions
  - *For example, if we want to compare the number of COVID-19 to 8/Nov/2020 contagions, among the 10 most affected countries, a bar chart, generated by the bar() function, will be indicated*

```
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()
```



In the present example, care was taken to use for the x axis labels only the initial 3 of the country name

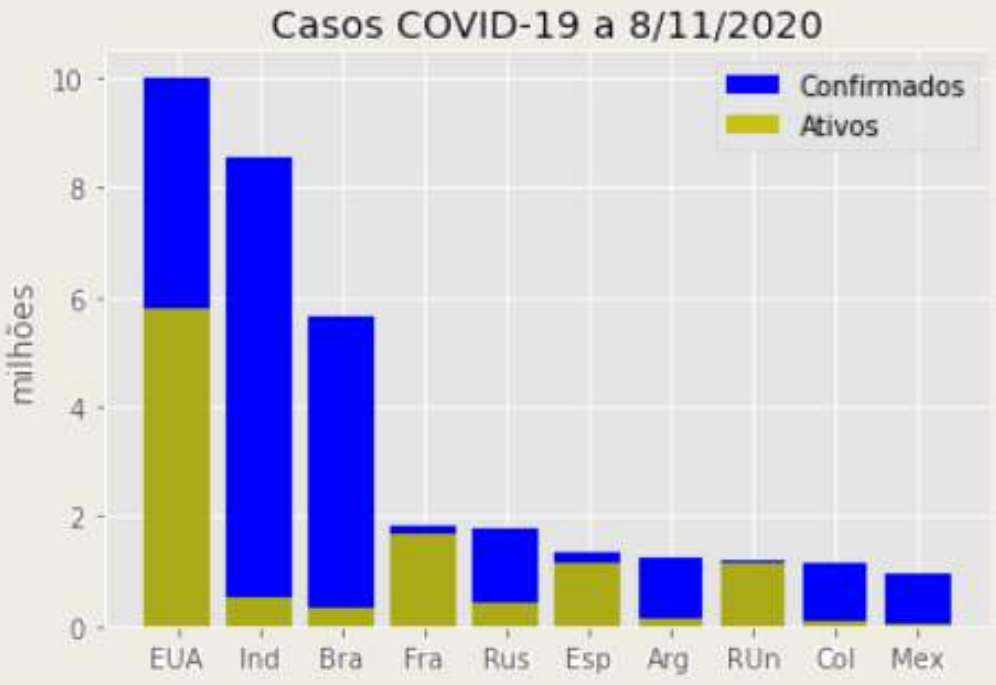
- But, not infrequently, it becomes difficult to include all labels on the x-axis
- In such cases, the solution would be to vertically dispose of these labels through the xticks function()

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

Overlap of bar charts

- As with line charts, we may also overpower 2 or more bar charts
  - *Continuing with the previous example, it may be interesting to show the graph of the numbers of cases still active in each country, overlaid on the chart of confirmed cases*
    - with this overlap, we have achieved an easy perception of the proportion of contagions that remain active in each country
  - *To obtain this overlap, simply add the following code to the previous code, responsible for generating the 2nd bar chart:*

```
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()
```



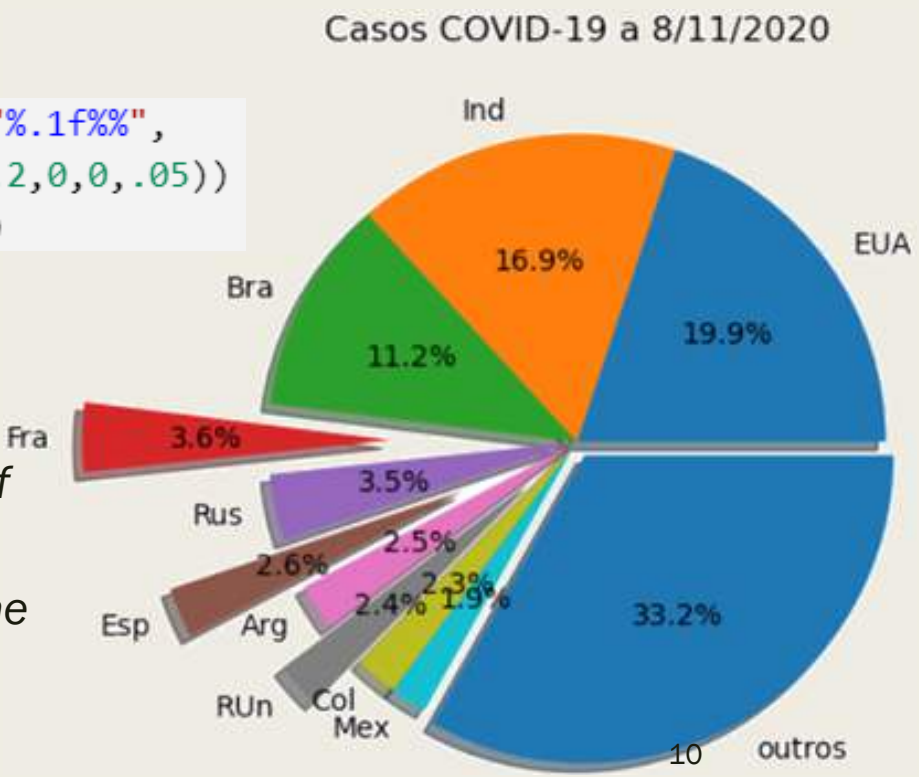
The alpha parameter regulates the opacity level between 0 and 1. It is particularly useful when some of the bars on the 2nd chart exceed those of the 1st (in this case, if we did not lower the alpha to an acceptable level of transparency, the 1st bars would be completely covered by the 2nd)

Circular graph

- Circular charts help represent the proportions of a whole
  - *therefore, the sum of the various values to be represented should make up 100%*
- To illustrate the construction of this type of chart, we can continue with the example of COVID-19 contagion
  - *it makes sense to use the pie chart to understand better what proportion of total contagion sits in each country*
  - *And since in this type of chart, we represent the entire universe of cases, we should start by adding another element to the series of countries, which represents all the cases of the rest of the world (16.74 million)*

```
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")
```



- The pie chart is generated by the pie() function
  - *With the parameter 'autopct' we indicate, through a formatting string, how the label of each slice is shown*
  - *With the 'explode' parameter, we indicate the percentage (from 0 to 1) of each share.*

Scatter plot

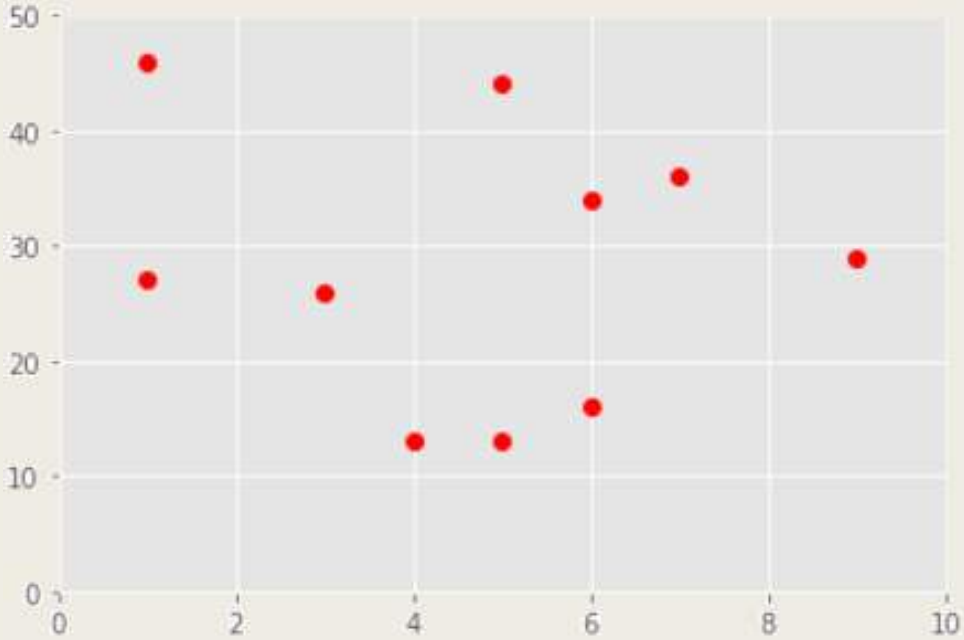
- A scatter plot uses points (or other type of markings) to represent in a two-dimensional plane the X and Y coordinates expressed by two variables, illustrating how the two are related
  - *This type of chart is also produced in Matplotlib by the plot() function; just pass it as a parameter a string with the formatting we want for the marker*
- To illustrate this type of chart, let's start by creating the two x and y coordinates

```
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]
```

And then invoke the plot() function

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



  - *If we did not include the string 'or', a line chart would be drawn, which in this case would not make much sense*

Scatter plot overlay

- Scatter plots can also be overlapped by repeatedly calling the plot() function or even through a single invocation
  - *To illustrate this second option, let's create the coordinates for two additional graphs*

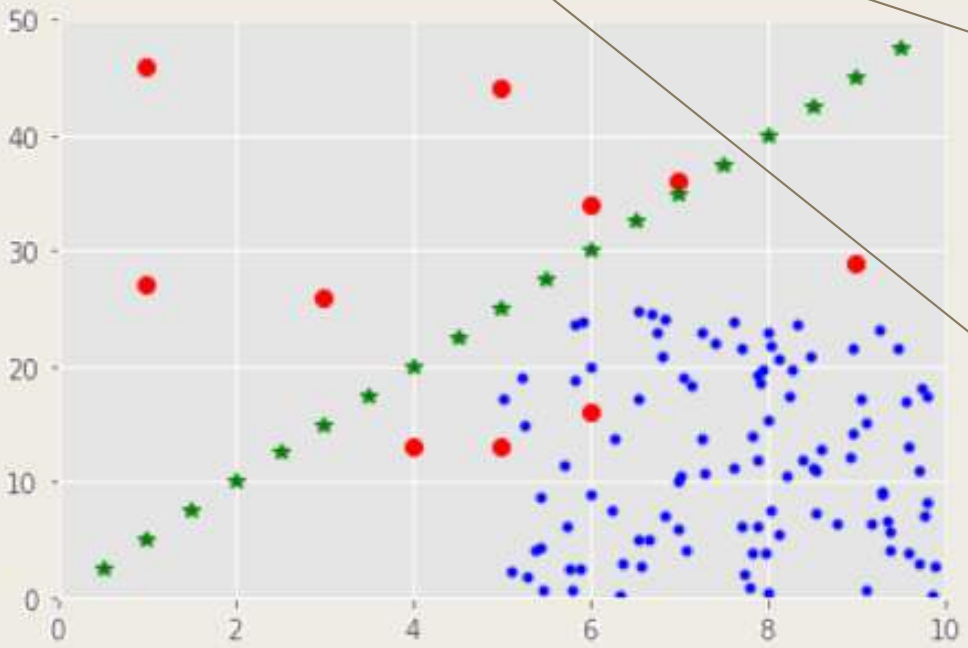
```
x1=np.arange(1,20)/2
y1=x1*5
x2=np.random.random(100)*5+5
y2=np.random.random(100)*25
```

100 random values between 5 and 10

100 random values between 0 and 25

  - *For the three graphs to appear overlapping, it is enough to move to the plot() function the coordinates and string of the marker of each of them in sequence*

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



'.b': blue dots

'*g': green stars

There are many other small strings to represent other shapes and colors of markers (see help(plt.plot))

To set the scale of the X and Y axes to specific values. Otherwise, the limits of the axes would adapt to the values presented.

Scatter plots with scatter() function

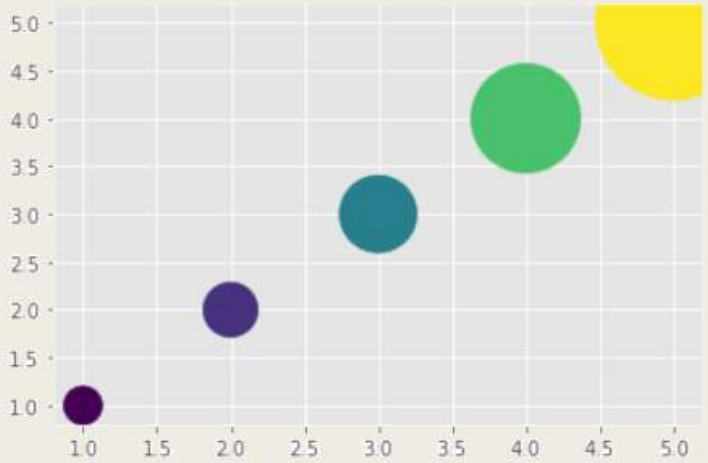
- More sophisticated scatter plots can be achieved with the scatter() function
  - *Used when we want variable color and size markers*
  - *While the scatter() function draws points without lines to interconnect them, the plot() function may or may not draw lines, depending on the arguments that are used*
- For example, we print the coordinate series (x2,y2) with the scatter() function
  - *For this purpose, we start by randomly defining the sequences of colors and sizes to use in the 100 markers*

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

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

These charts are sometimes also called bubble charts, and you can see why…

- Or, for easier understanding, look at this simpler example

```
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)
```



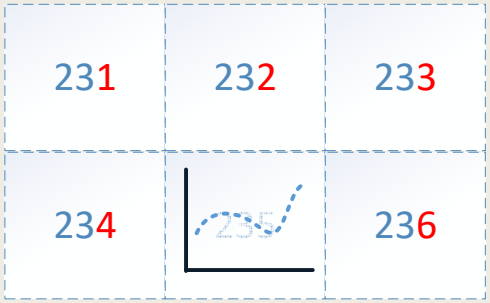
Subplots (combine without overlooking)

- We may also combine <u>without overlooking</u> multiple charts in the same figure (graphic window), drawing them side by side (horizontally or vertically) in a smaller size
  - *It is the subplot() function that scales and positions each of these small charts*
  - *More specifically, the subplot() function creates the rectangular subarea where the following chart to be generated will appear, with the dimension and position specified through a parameter formed by three digits:*
    - the first two establish the number of rows and columns of an invisible grid that defines the possible subareas where the following chart will be printed,
    - the third digit indicates the position of the subarea of this grid where the following graph will appear.
  - *For example, if we started with the*

```
subplot(235)
```

selects the 5th subarea defined by a 2-row, 3-column grid

the next chart will appear with the dimension and position indicated in the following figure



- The same indication could be given by passing each of the digits as a separate parameter: suplot(2,3,5)
- This option would have the advantage of not limiting the grid to the 3x3 dimension

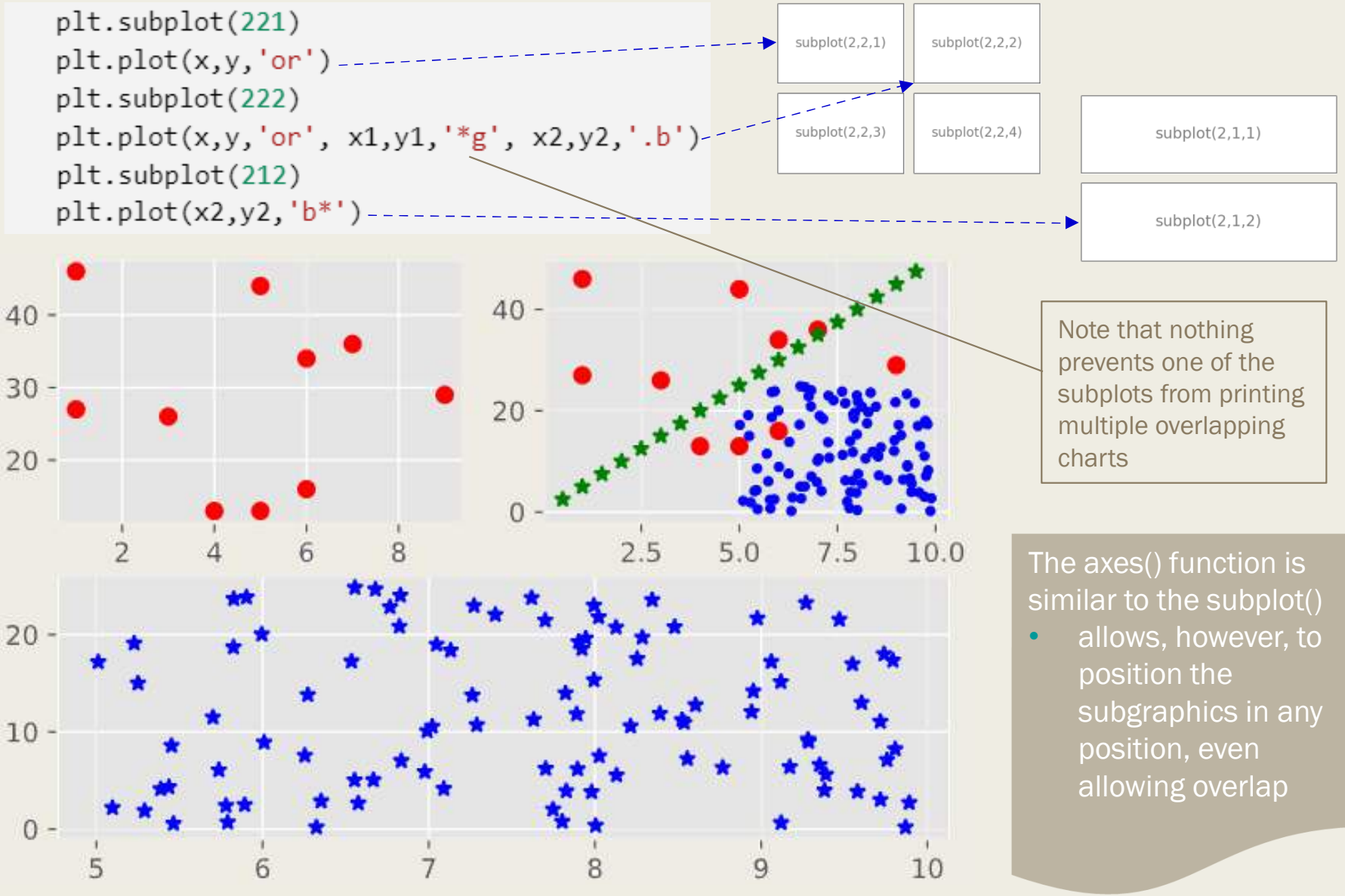
Subplots (possible subareas)

- The following are illustrations of the subareas selected in other possible subplots

Subplots (Exemplifying)

- With the sequences of values previously produced, we will exemplify the use of the subplot(function), drawing 3 combined graphs in the same figure

```
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*')
```



Note that nothing prevents one of the subplots from printing multiple overlapping charts

The axes() function is similar to the subplot()

- allows, however, to position the subgraphics in any position, even allowing overlap

The Seaborn Library

- Seaborn is a complementary data visualisation library based on matplotlib
  - *Provides us with a set of tools for building statistical graphs in Python*
  - *It is another excellent aid for the exploration and understanding of data*
  - *Perfectly integrated with Pandas data structures – its charting functions operate directly on dataframes*
  - *With its high-level interface, we can produce sophisticated and attractive graphics with less effort than with Matplotlib*
  - *And more importantly, it allows us to easily visualise the relationship between multiple variables (columns), whether numerical or categorical*
    - In programming, we are used to classifying this second category of variables as variables of enumerated type
    - In Data Science, either string variables (dataset columns) or numeric variables representing a code or id (which serve only to identify or qualify something, not to quantify) are, as a rule, interpreted as categorical
- To illustrate the type of charts that can be produced with Seaborn, let's take advantage of the student DataFrame used previously to, visualise different combinations of numerical and categorical variables
  - *In order to diversify the demonstrations, the student's dataframe was completed more, adding a few more rows, the column 'gender' and converting your last column to integer (with 0 meaning disapproved and 1 approved)*

The Students DataFrame (for viewing)

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

View 1 numeric variable

- If we hold, for example, an analysis of the distribution of the numerical variable notaIA, we can use the function seaborn.distplot()

  (after importing the respective package, of course, and also pandas so we can use the Student DataFrame)

```
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');
```

Selects a preset chart style

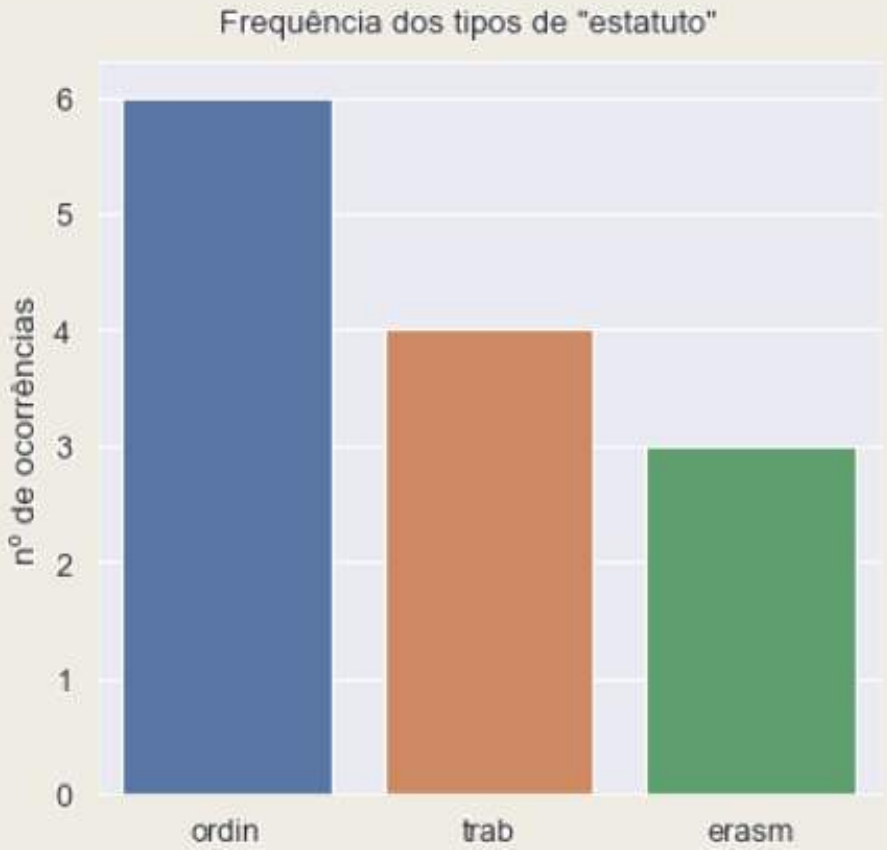
Note that we can provide Seaborn's own columns of our DataFrame

- Even with a tiny dataset of simulated data, this variable of yours seems to follow a normal distribution...

View 1 categorical variable

- If we want to analyse a categorical variable, we can do so through the seaborn.plot() function, show how often each of the different values of that variable occur
  - *Let us show, for example, the number of students for each type of frequency (which, in addition to not being entangled in the language, we also call "status")*

```
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");
```

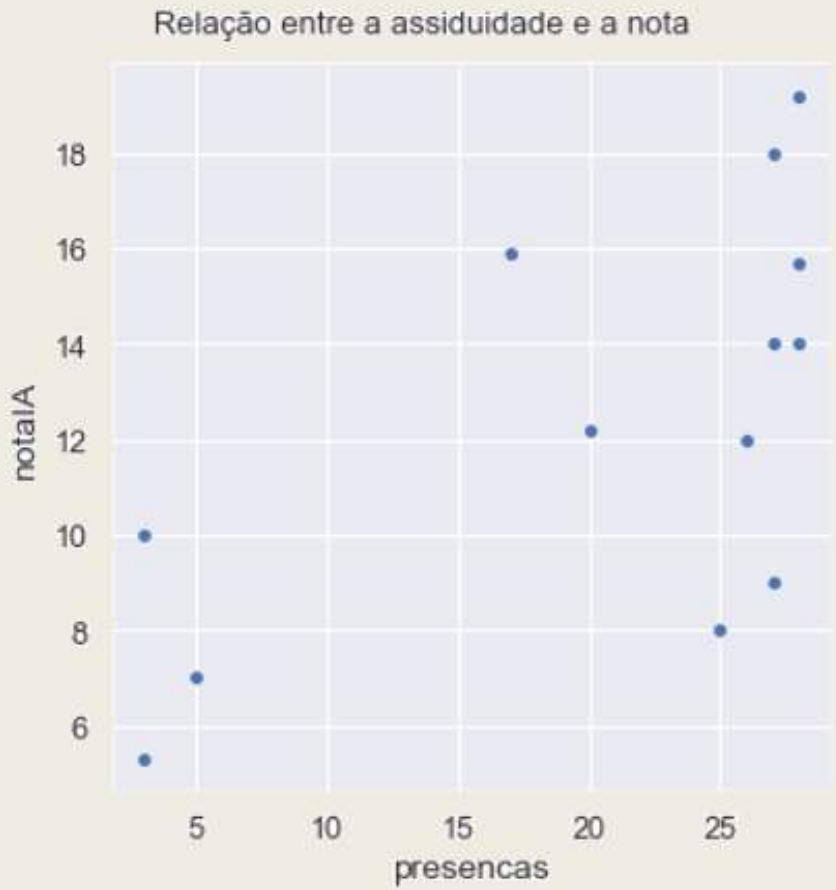


- As expected, the ordinary are outnumbered...
- But also the other frequency types begin to be very well represented (dataset with "invented" data, not forget.)

View 2 numeric variables

- To visualize the relationship between two numeric variables, the seaborn.relplot() function can be used
  - *Show, for example, the scatter plot of the variables 'presences' and 'notaIA', so that we can achieve a quick perception of the influence that the student's attendance may have on his performance*

```
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);
```

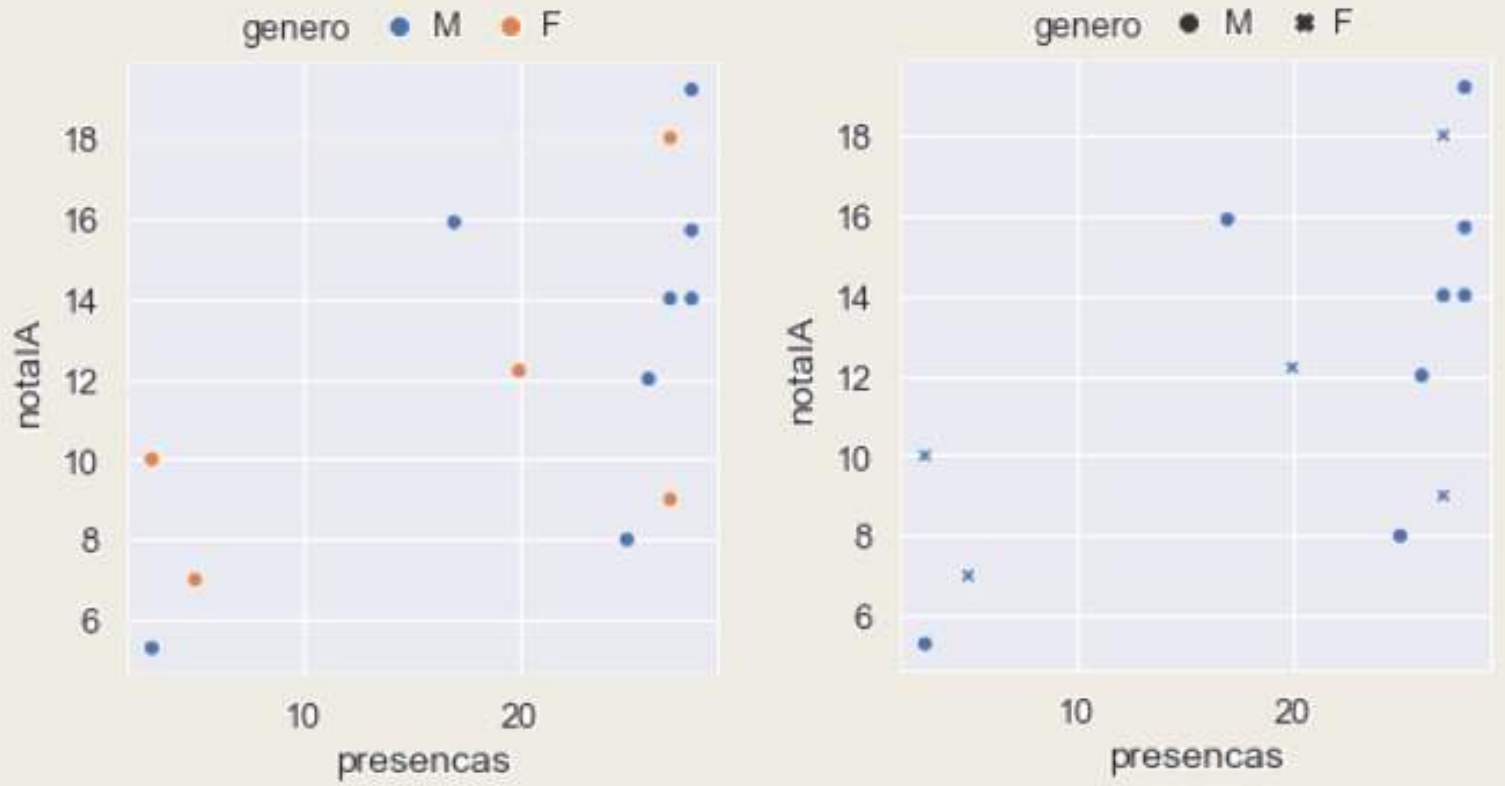


- Even with these simulated data, one realizes the great importance of attendance in the final note... ☺

View 2 numeric and 1 categorical variables

- With two numerical variables and one categorical variable, we can use the latter to differentiate the points (markers) of the 2D chart from the two numerical variables
  - *The dimension that this third variable introduces can in fact be reflected in the color or shape of the 2D chart points*
- To distinguish the gender in the previous graph, let us add this categorical variable to it

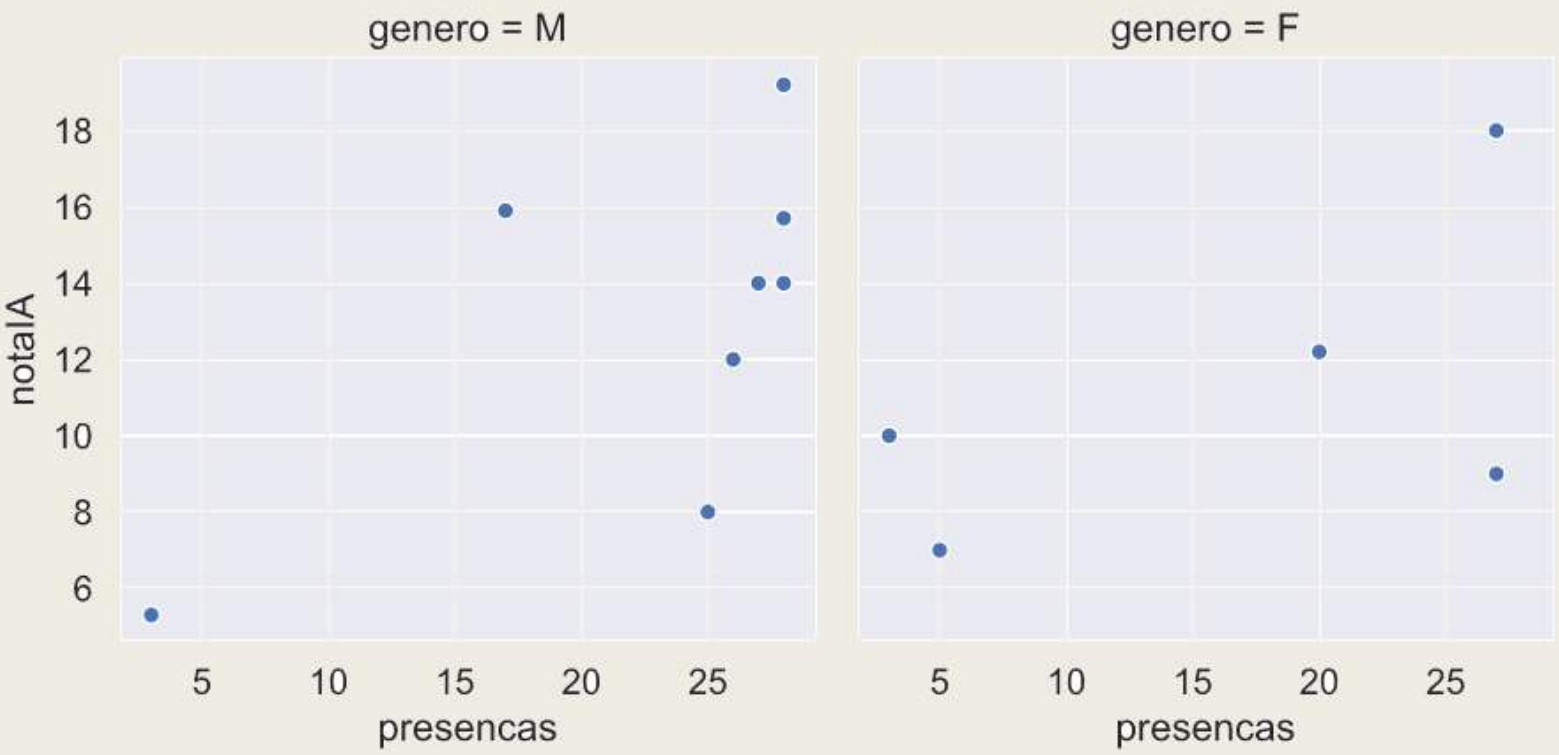
```
g = sns.relplot(x="presencas",y="notaIA",data=alunos,kind='scatter',hue='genero')
g = sns.relplot(x="presencas",y="notaIA",data=alunos,kind='scatter',style='genero')
```



View 2 numeric and 1 categorical variables

- A third option is to show in separate charts the observations of the different categories

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



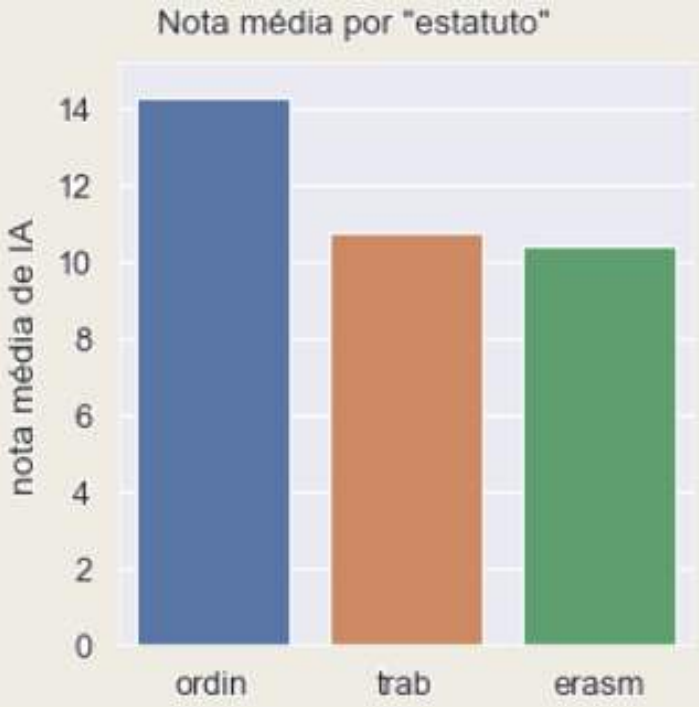
View 1 numeric variable and 1 categorical

- With a numeric and a categorical variable, you can build various types of charts, such as bar charts and 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");
```

The bar chart shows the average numerical values for each category



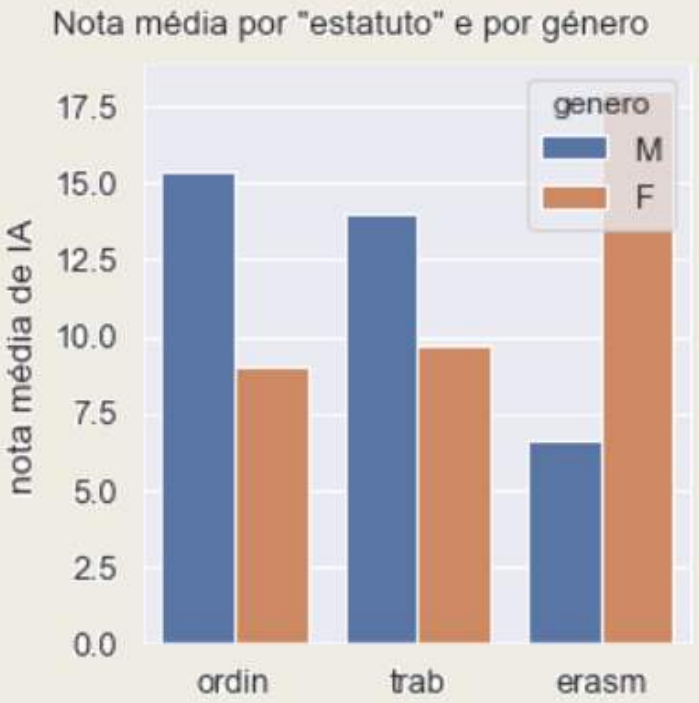
The strip chart shows the numerical values of each category in separate tracks

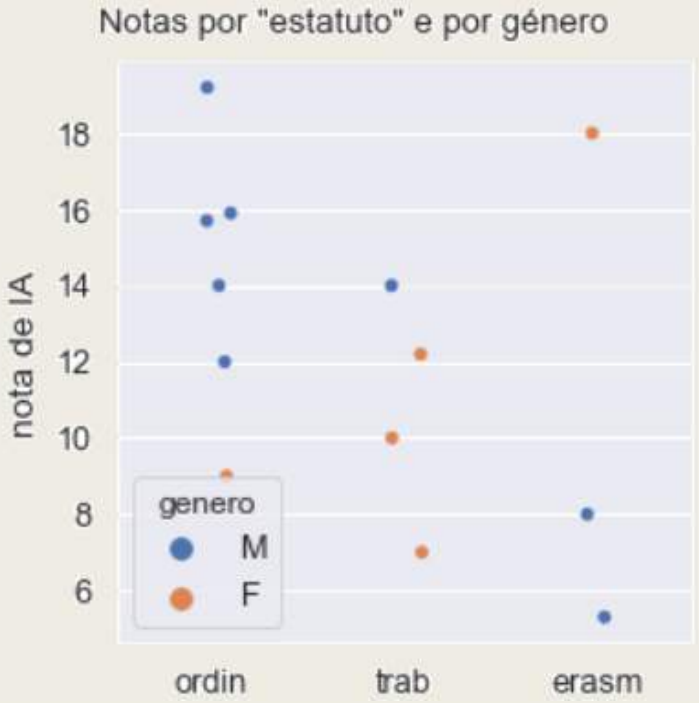
View 1 numeric variable and 2 categorical

- With a numerical variable and two categorical variables, we can accommodate the additional categorical variable to any of the graphs referenced in the previous slide,
  - *either through the 'hue' parameter, reflecting this category in the color,*
  - *either through the parameter parameter 'col', passing the observations of the various categories to be shown in separate graphs*

```
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");
```





View 1 numeric variable and 2 categorical

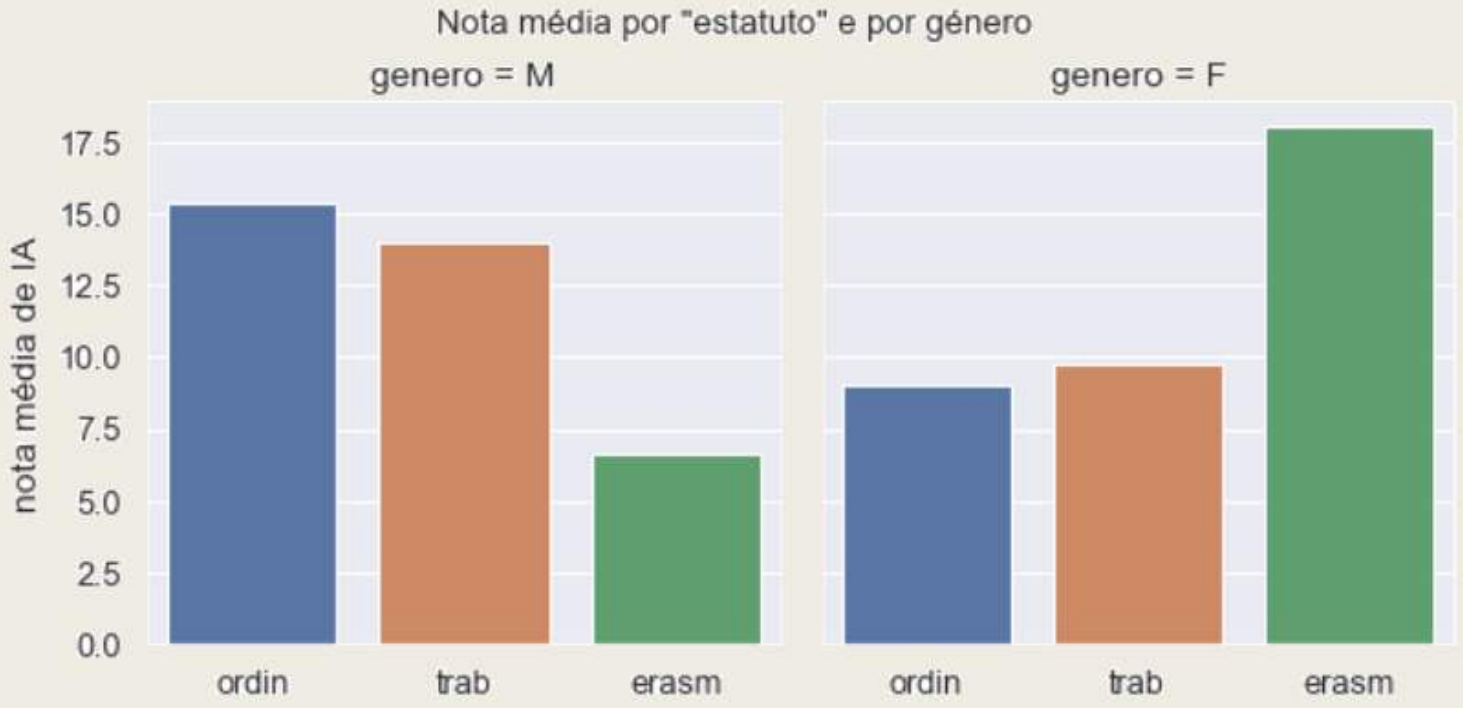
- Observations of both gender types can be presented in two separate strip charts, choosing the option col='genero'

```
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

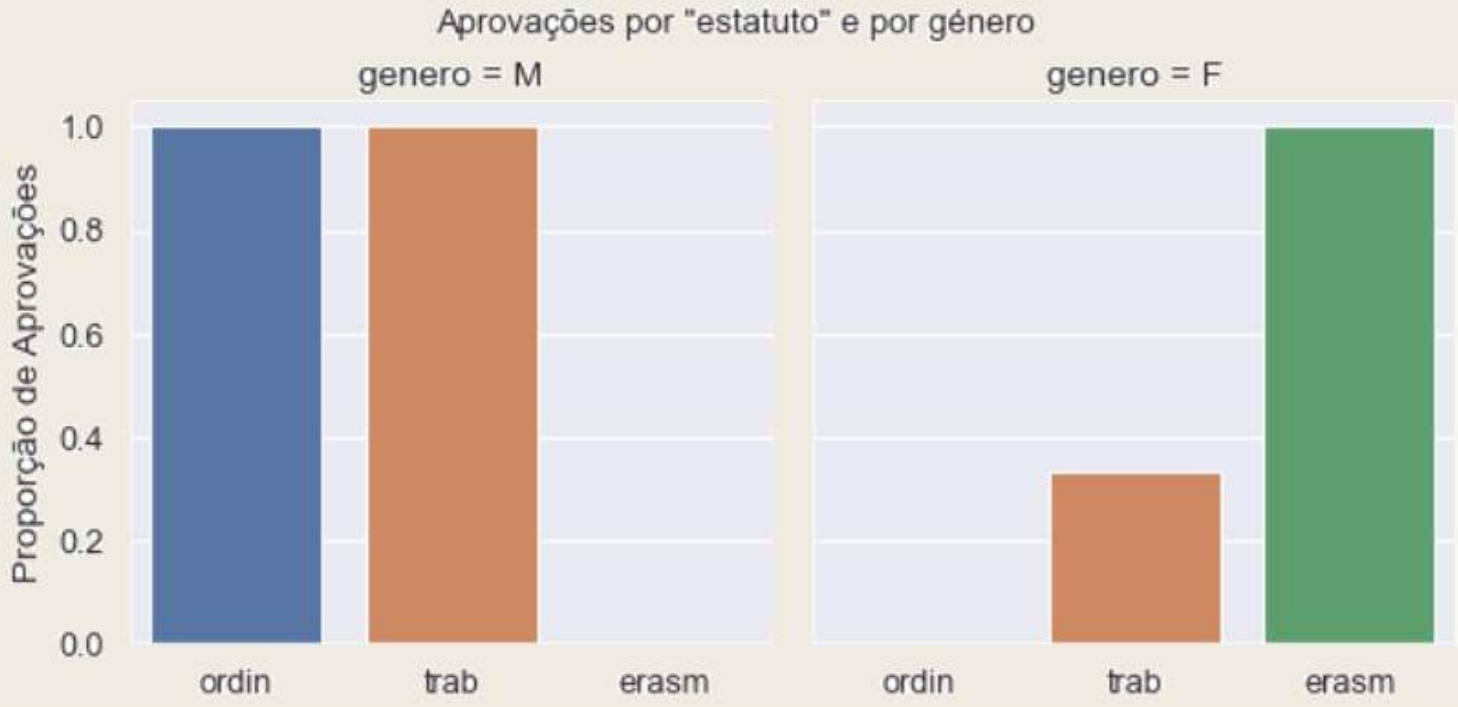
```
g = sns.catplot(x='freq',y='notaIA',kind='bar',ci=False,data=alunos,col='genero')
g.fig.suptitle('Nota média por "estatuto" e por género')
g.set_axis_labels("", "nota média de IA");
```



View 1 numeric variable and 2 categorical

- If, alternatively, we use for the numerical variable of the graph preceding the 'approved' column (of 0s and 1s), instead of the 'note' column, the obtained graphs will have a slightly different interpretation
  - *Note that in this case, the average value of 0s and 1s ends up giving us other more relevant information*
    - It gives us precisely the ratio of 1s, which in this case translates the approval ratio

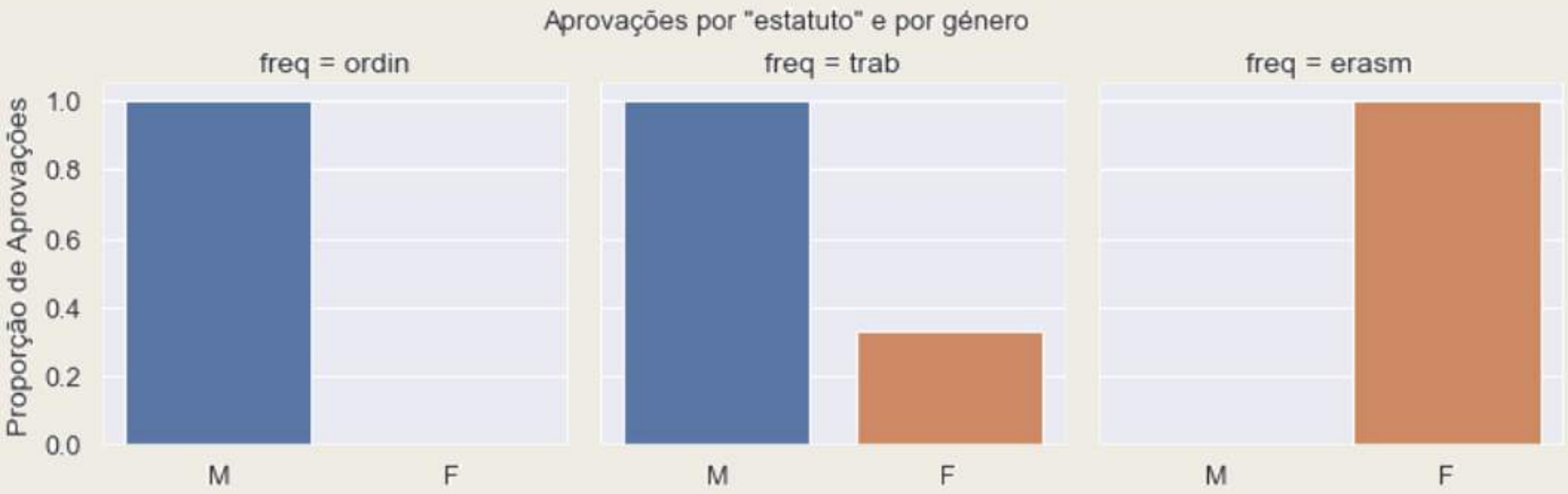
```
g = sns.catplot(x='freq',y='aprovado',kind='bar',ci=False,data=alunos,col='genero')
g.fig.suptitle('Aprovações por "estatuto" e por género')
g.set_axis_labels("", "Proporção de Aprovações");
```



View 1 numeric variable and 2 categorical

- And, of course, we can also exchange the role played by the 2 categorical variables

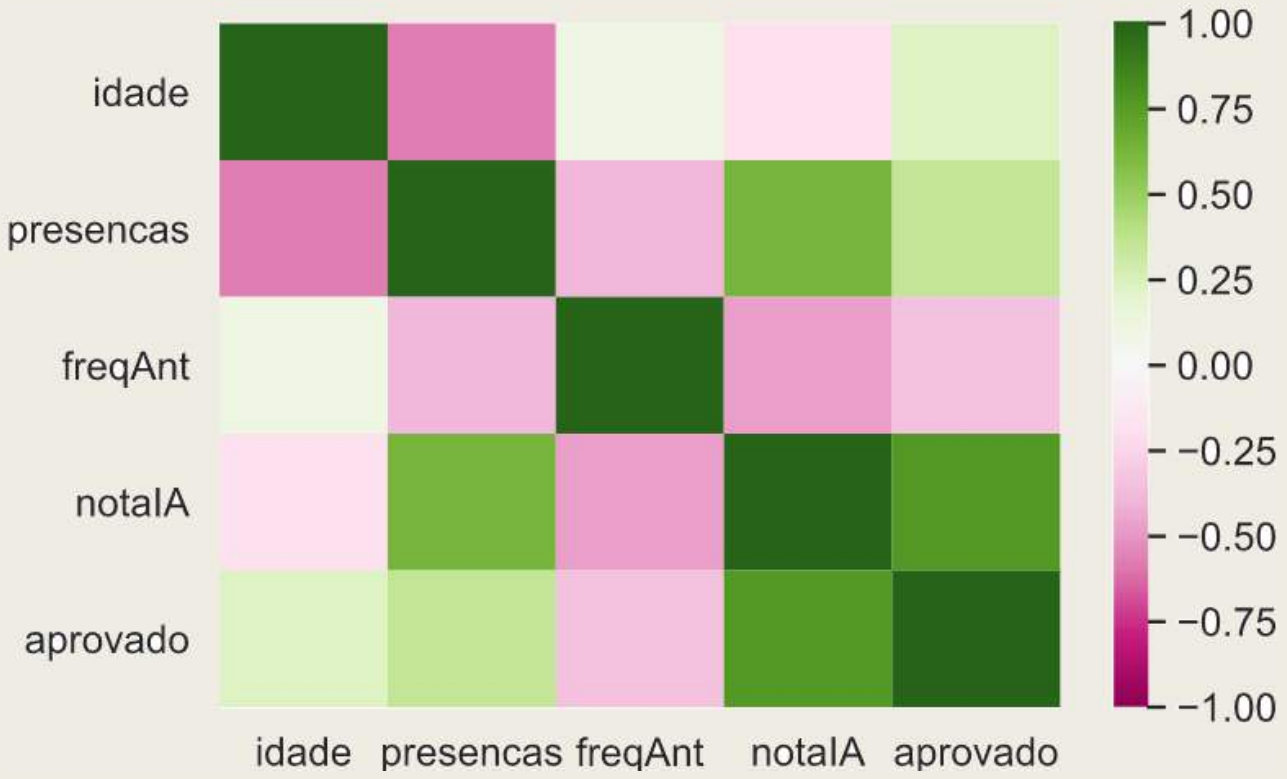
```
g = sns.catplot(x='genero',y='aprovado',kind='bar',ci=False,data=alunos,col='freq')
g.fig.suptitle('Aprovações por "estatuto" e por género')
g.set_axis_labels("", "Proporção de Aprovações");
```



View 3 or more numeric variables

- To visualize 3 or more non-categorical variables, we also have the seaborn.heatmap() function, in which the variation of a given metric is presented with different chromatic shades
  - *One of the most commonly used metrics is the correlation function*
  - *In a single chart of this type it is then possible to visualize the relationship of each variable with any other*
  - *The most correlated pairs of variables are represented by more extreme shades, always depending on the scale that is chosen*

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



With the tonal scale chosen in this example, dark green represented the most positively correlated variables, and dark pink the most negatively correlated variables are represented

More about Seaborn

- While we can with Seaborn to quickly and easily produce complex and attractive graphics, Seaborn offers many customization options, so that the graphic presentations are even more perfect
  - *This will be a more advanced topic that should be explored soon after getting some mastery of the essential aspects dealt with here*
  - *For more detailed information see the official website:* https://seaborn.pydata.org/

- Seaborn carries with it a set of datasets that we can upload through the load_dataset() function, and refer to their names with the 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']
```

  - *One of them has the data of the passengers who boarded the 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

solve exercise #13, #14 e #15

from the book of exercises