

Advanced graphics

Erasmus Q-Intelligence B.V.

Data Science and Business Analytics
Programming



Content

- 1 Software requirements
- 2 Reminder
- 3 Finetuning graphics
- 4 Adding to plots
- 5 Conclusions



Software requirements



Software requirements

- Data from packages dplyr and ggplot2 are used, made available for as csv for Python
- We use mainly functions from libraries pandas and seaborn, but also need matplotlib, statsmodel, numpy and mpl_toolkits for some specific cases

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import colors as mcolors
import numpy as np
import statsmodels.api as sm
from mpl_toolkits.axes_grid1 import make_axes_locatable
```



Data sets

Atlantic hurricane database track data, 1975-2015

```
storms = pd.read_csv("storms.csv")  
storms_2015 = storms[storms['year'] == 2015]
```

US economic time series

```
economics = pd.read_csv("economics.csv")
```

Eredivisie points of Ajax and Feyenoord

```
eredivisie = pd.read_csv("eredivisie.csv")
```

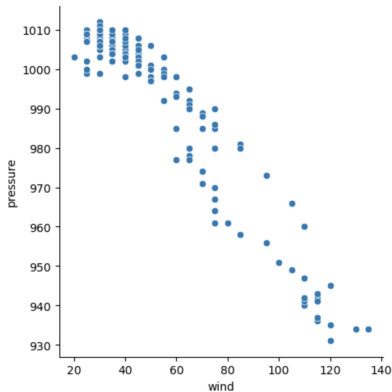


Reminder



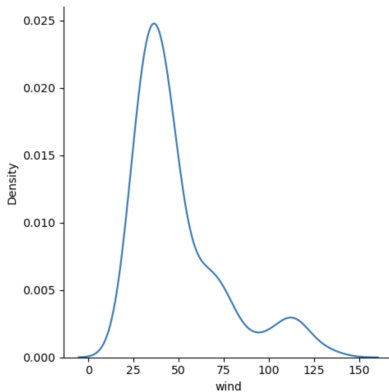
Scatterplot

```
sns.relplot(data = storms_2015, x = "wind",  
            y = "pressure")
```



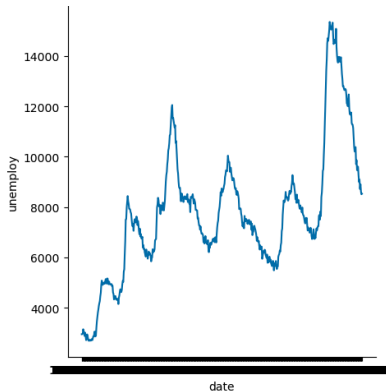
Density plot

```
sns.displot(storms_2015, x = "wind", kind = "kde")
```



Time series plot

```
sns.relplot(economics, kind = "line", x = "date",  
            y = "unemploy")
```



Finetuning graphics



Finetuning within in a plot

Finetuning through **arguments** and **functions**:

`color` or `c` Color for points/lines

`alpha` Transparency of colors

`marker` Symbol for points

`linestyle` Type of line

`s` Size of points/lines

→ If the **values should depend on some variable**, then use the `hue` and `palette` parameters
(information from those variables is mapped to the visual representation)



Specifying colors

There are many options, such as using single characters for simple, shorthand notation:

- 'b' as blue
- 'g' as green
- 'r' as red
- 'c' as cyan
- 'm' as magenta
- 'y' as yellow
- 'k' as black
- 'w' as white

→ For a list of other options: [click here](#)



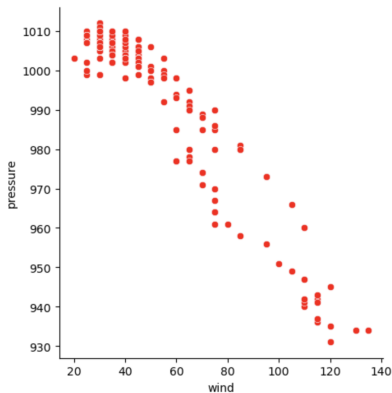
Specifying colors: full name

Another option is to use the full name of a color

black	k	dimgray	dimgray
gray	gray	darkgray	darkgray
silver	lightgray	lightgray	gainsboro
whitesmoke	w	white	snow
rosybrown	lightcoral	indianred	brown
firebrick	maroon	darkred	r
red	mistyrose	salmon	tomato
darksalmon	coral	orangered	lightsalmon
sienna	seashell	chocolate	saddlebrown
sandybrown	peachpuff	peru	linen
bisque	darkorange	burlwood	antiquewhite
tan	navajowhite	blanchedalmond	papayawhip
moccasin	orange	wheat	oldlace
floralwhite	darkgoldenrod	goldenrod	cornsilk
gold	lemonchiffon	khaki	palegoldenrod
darkkhaki	ivory	beige	lightyellow
lightgoldenrodyellow	olive	y	yellow
olivedrab	yellowgreen	darkolivegreen	greenyellow
chartreuse	lawngreen	honeydew	darkseagreen
palegreen	lightgreen	forestgreen	limegreen
darkgreen	g	green	lime
seagreen	mediumseagreen	springgreen	mintcream
mediumspringgreen	mediumaquamarine	aquamarine	turquoise
lightseagreen	mediumturquoise	azure	lightcyan
paleturquoise	darkslategray	darkslategrey	teal
darkcyan	c	aqua	cyan
darkturquoise	cadetblue	powderblue	lightblue
deepskyblue	skyblue	lightskyblue	steelblue
aliceblue	dodgerblue	lightslategray	lightslategray
slategray	slategrey	lightsteelblue	cornflowerblue
royalblue	ghostwhite	lavender	midnightblue
navy	darkblue	mediumblue	b
blue	slateblue	darkslateblue	mediumslateblue
mediumpurple	rebeccapurple	blueviolet	indigo
darkorchid	darkviolet	mediumorchid	thistle
plum	violet	purple	darkmagenta
m	fuchsia	magenta	orchid
mediumvioletred	deeppink	hotpink	lavenderblush
palevioletred	crimson	pink	lightpink

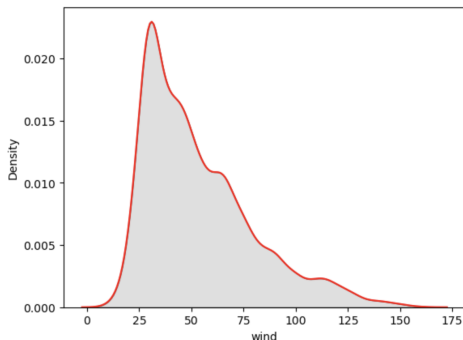
Named colors: scatterplot

```
sns.relplot(data = storms_2015, x = "wind",  
            y = "pressure", color = "red")
```



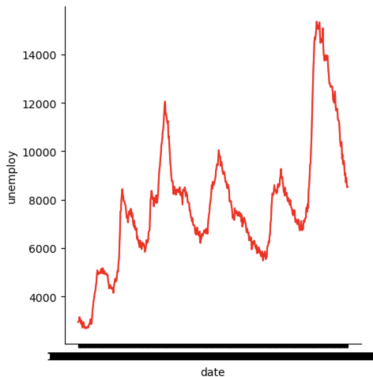
Named colors: density plot

```
sns.kdeplot(data = storms, x = "wind", color = 'grey',  
            fill=True)  
sns.kdeplot(data = storms, x = 'wind', color = 'red')
```

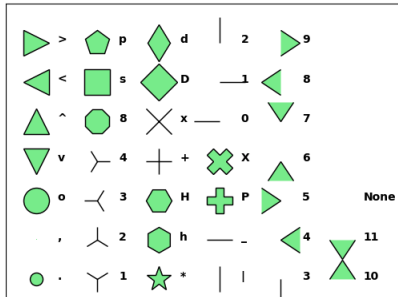


Named colors: time series plot

```
sns.relplot(economics, kind = "line", x = "date",  
            y = "unemploy", color = "red")
```



Plot symbols



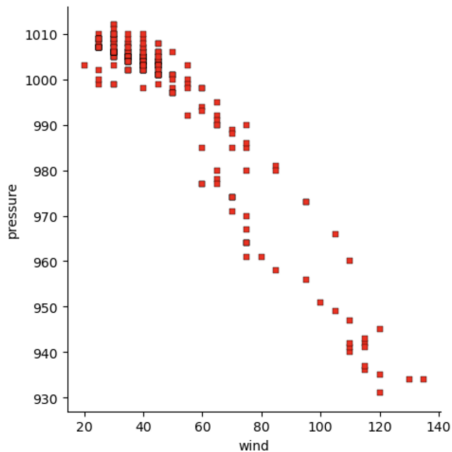
→ Specify fill color with parameter `color`, edge color with `edgecolor` and size with `s`

Plot symbols: scatterplot

```
sns.relplot(data = storms_2015, x = "wind",  
            y = "pressure", marker = "s",  
            color = "red", edgecolor = "black",  
            s = 15)
```







Plot symbols: scatterplot



Line types: density plot

Named linestyles

solid 'solid'	
dotted 'dotted'	
dashed 'dashed'	
dashdot 'dashdot'	

→ Other options, such as more white space between dots are possible, but not named. See the jupyter notebook of this lecture for those options.

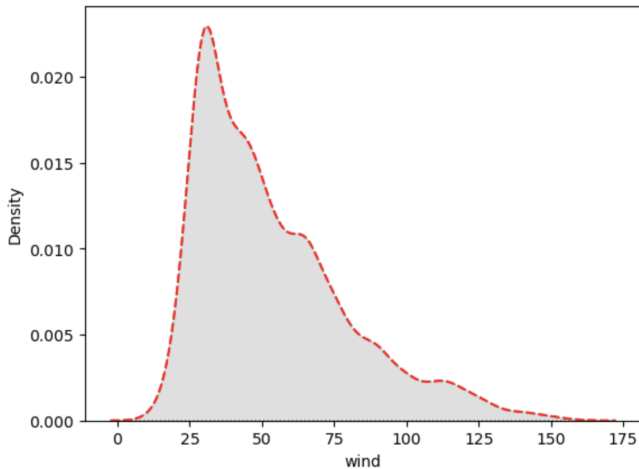


Line types: density plot

```
sns.kdeplot(data = storms, x = "wind",  
            color = 'grey', fill=True,  
            linestyle = "dotted")  
sns.kdeplot(data = storms, x = 'wind',  
            color = 'red', linestyle = "dashed")
```

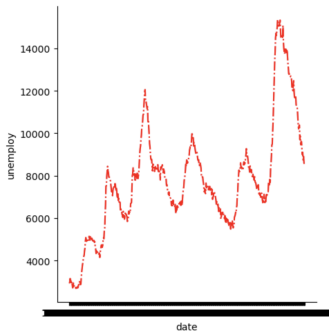


Line types: density plot



Line types: time series plot

```
sns.relplot(economics, kind = "line",  
            x = "date", y = "unemploy",  
            color = "red", linestyle = "dashdot")
```



Arguments for finetuning within a plotrevisited

Finetuning through **arguments** and **functions**:

`color` or `c` Color for points/lines

`alpha` Transparency of colors

`marker` Symbol for points

`linestyle` Type of line

`s` Size of points/lines

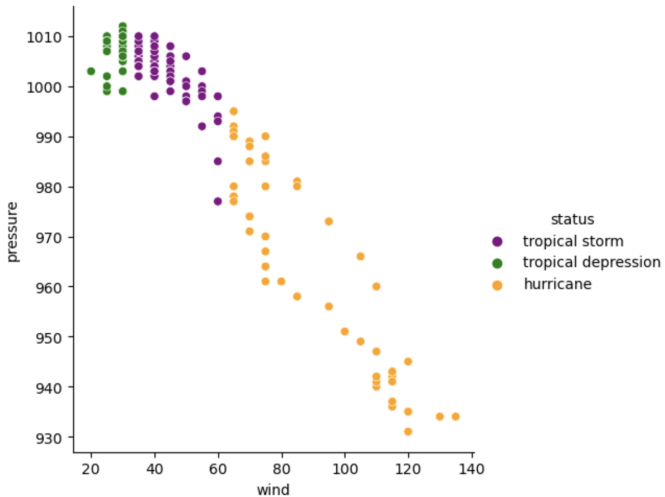


Colors: scatterplot

```
sns.relplot(data = storms_2015, x = "wind",  
            y = "pressure", hue = "status",  
            palette = ['purple', 'green', 'orange'])
```



Colors: scatterplot

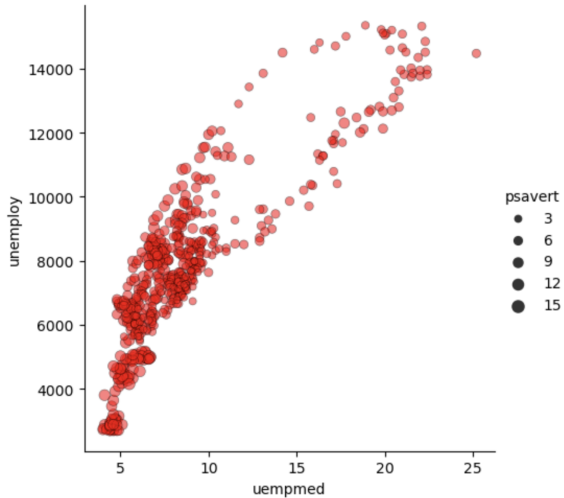


Size: scatterplot

```
sns.relplot(data = economics, x = "uempmed",  
            y = "unemploy", color = "red",  
            edgecolor = "black", size = "psavert",  
            alpha = 0.5)
```

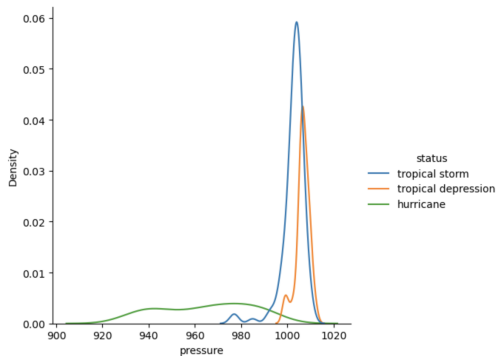


Size: scatterplot



Colors: density plot

```
sns.displot(storms_2015, x = "pressure",  
            hue = "status", kind = "kde")
```

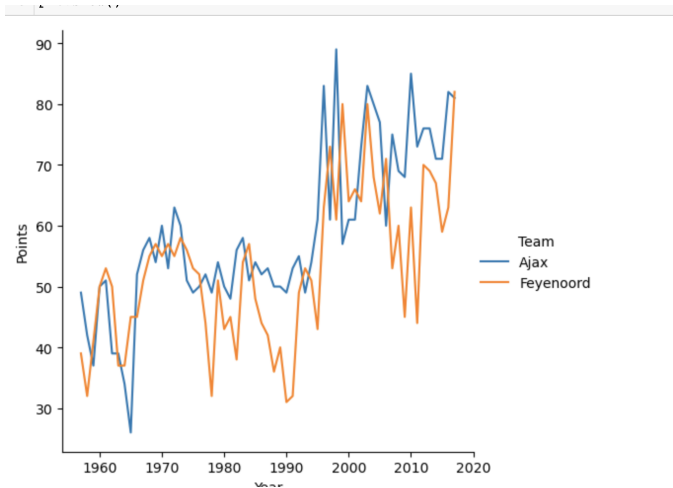


Colors: time series plot

```
sns.relplot(eredivisie, x = "Year",  
             y = "Points", hue = "Team",  
             kind = "line")
```



Colors: time series plot



Exercises

- Download file *AdvancedGraphics-Exercises.pdf* from Canvas and open it
- Do Exercise 1



Manual scales

- Built-in scales are used to obtain default values
- Manual scales can be applied, how depends on the change you would like to make

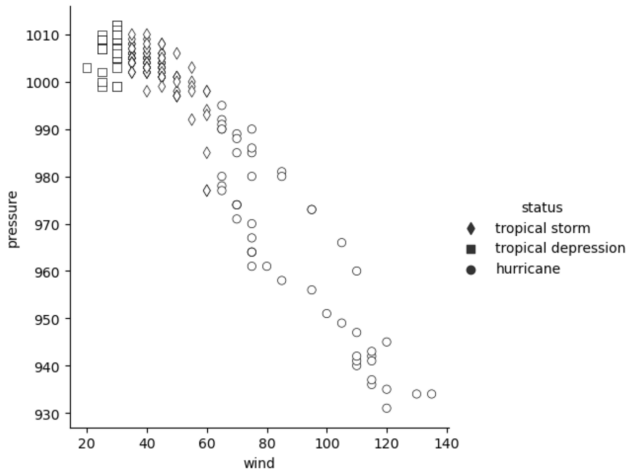


Plot symbols: scatterplot

```
sns.relplot(data = storms_2015, x = "wind",  
            y = "pressure", style = "status",  
            markers = ['d', 's', 'o'],  
            color = "white", edgecolor = "black")
```



Plot symbols: scatterplot



Plot symbols: scatterplot

```
sns.relplot(data = storms_2015, x = "wind",  
            y = "pressure", hue = "status",  
            palette = ['blue', 'orange', "red"],  
            style = "status",  
            markers = ['d', 's', 'o'])
```

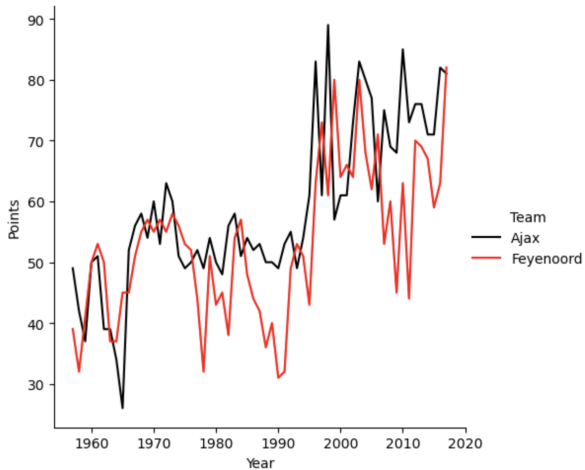


Colors: time series plot

```
sns.relplot(eredivisie, x = "Year", y = "Points",  
            hue = "Team", kind = "line",  
            palette = {"Ajax": "black",  
                      "Feyenoord": "red"})
```

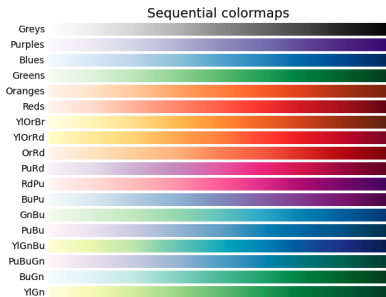


Colors: time series plot



Continuous scales

Continuous scales through argument `cmap`:



- Useful if points/lines/areas depend on a continuous variable
- [Click here for an overview of other color maps](#)



Colors: scatterplot

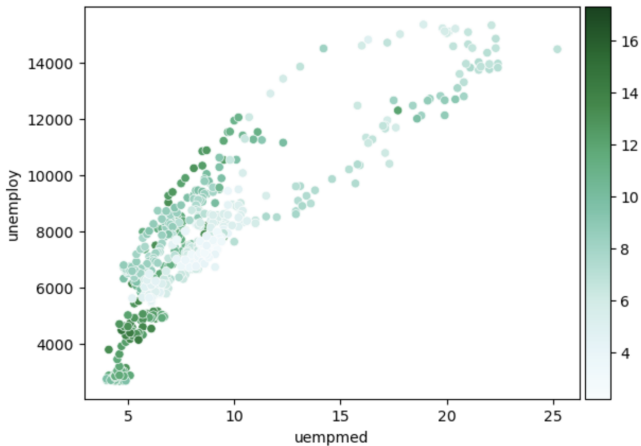
```
ax = sns.scatterplot(data = economics, x = "uempmed",
                    y = "unemploy", hue = "psavert",
                    palette = "BuGn")

# Create color bar
norm = plt.Normalize(economics['psavert'].min(),
                    economics['psavert'].max())
smmap = plt.cm.ScalarMappable(cmap="BuGn", norm=norm)
smmap.set_array([])

# Remove the legend and add a colorbar
ax.get_legend().remove()
ax.figure.colorbar(smmap)
```



Colors: scatterplot



Removing or adding legends

- For some plots, the legend might not provide new information, while for other plots the legend might be highly informative.
- In both pandas as seaborn the legend can be removed by setting the argument `legend` to `False`



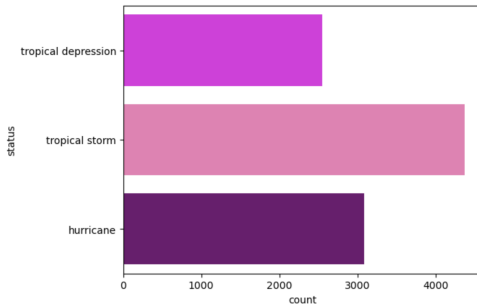
Flipping coordinates

- Some plots (e.g., barplots or conditional boxplots), are better displayed horizontally than vertically
 - Lengths are easier to judge for humans than heights
 - Easier to fit group labels
- Flipping coordinates can be done by swapping x and y in a barplot or boxplot.



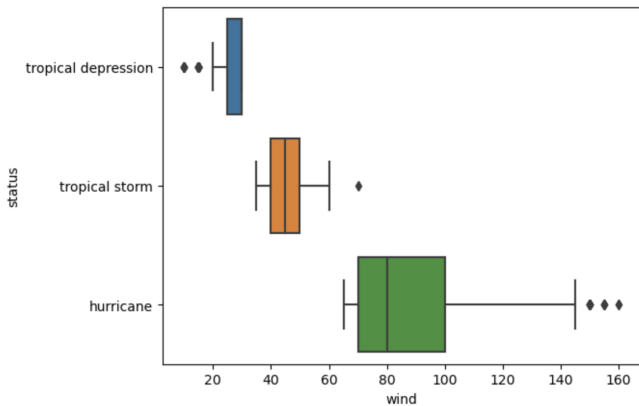
Flipping coordinates: barplot

```
sns.countplot(data = storms, y = 'status',  
               palette = ["magenta", "hotpink", "purple"])
```



Flipping coordinates: conditional boxplots

```
sns.boxplot(data = storms, x = "wind", y = "status")
```



Axis limits and annotation

Axis limits:

`set_xlim()` x-Axis limits

`set_ylim()` y-Axis limits

→ Two values for lower and upper limit, get current axes using function `plt.gca()`

Annotation: use functions on the current axes to add title and labels

`set_title()` Plot title

`set_xlabel()` x-Axis label

`set_ylabel()` y-Axis label

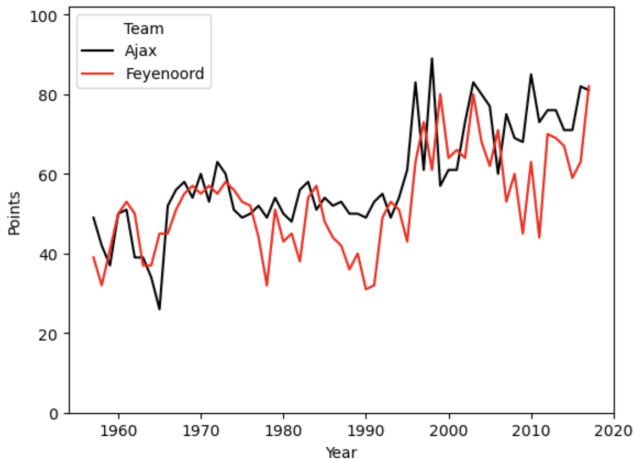


Axis limits: time series plot

```
sns.lineplot(eredivisie, x = "Year", y = "Points",  
             hue = "Team",  
             palette = {"Ajax": "black",  
                       "Feyenoord": "red"})  
  
ax = plt.gca()  
ax.set_ylim(0, 102)
```



Axis limits: time series plot



Exercises

→ Open file *AdvancedGraphics-Exercises.pdf*

→ Do Exercise 2



Adding to plots



Layers

- `matplotlib` works in layers
- Allows to add additional information to a plot
- Made easy by `seaborn`

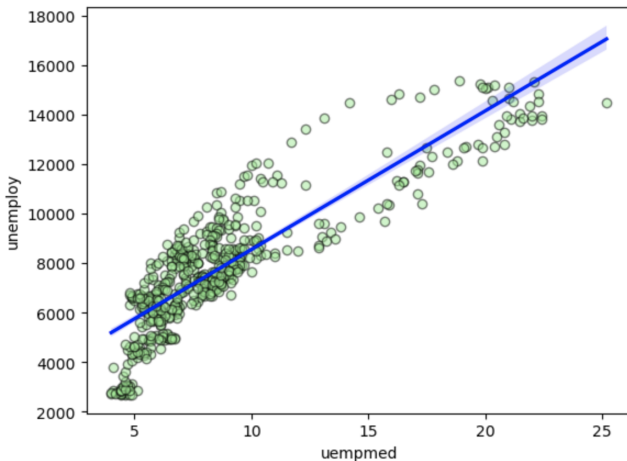


Adding a scatterplot smoother

```
sns.regplot(x = 'uempmed', y = 'unemploy',  
            data = economics, ci = 95,  
            scatter_kws = {"alpha": 0.5,  
                           "edgecolors": "black",  
                           "color": "lightgreen"},  
            line_kws = {"color": "blue"})
```



Adding a scatterplot smoother

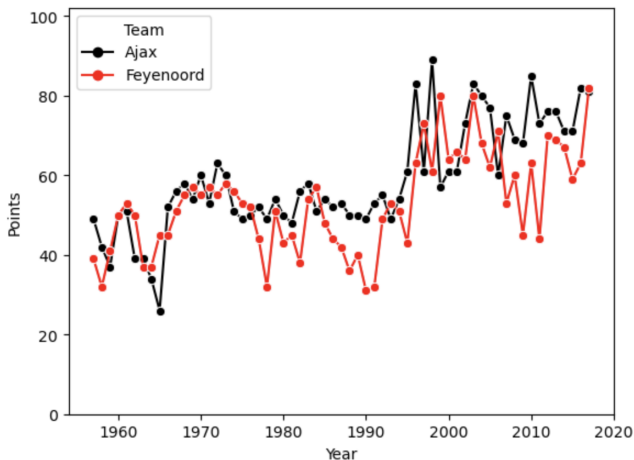


Plotting both points and lines

```
sns.lineplot(eredivisie, x = "Year",  
              y = "Points", hue = "Team",  
              palette = {"Ajax": "black",  
                          "Feyenoord": "red"},  
              style = "Team",  
              dashes = False,  
              markers= ["o", "o"])  
ax = plt.gca()  
ax.set_ylim(0, 102)
```



Plotting both points and lines



Adding vertical lines

```
# Turn date into a datetime type to get the axis correct
economics['date'] = pd.to_datetime(economics['date'])

# Calculate the unemployment relative to the population
economics['unemploy_pop'] =
    economics['unemploy']/economics['pop']

# Create figure
sns.relplot(economics, kind = "line",
            x = "date", y = "unemploy_pop",
            color = "#9aad88")
```



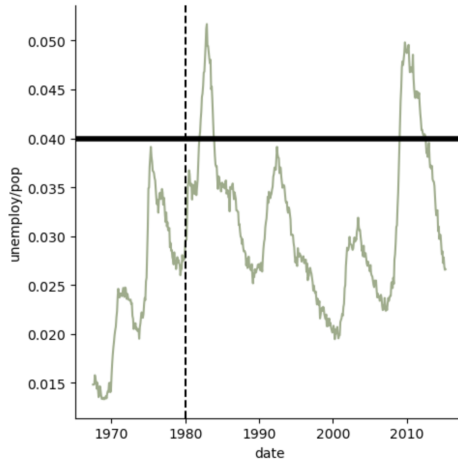
Adding vertical lines

```
# Add horizontal and vertical lines
plt.axhline(y = 0.04, c = "black", linewidth = 4)
plt.axvline(x = pd.to_datetime("1980-01-01"),
            c = "black", linestyle = "dashed")

# Axis labels and adjust ticks
plt.ylabel("unemploy/pop")
plt.xlabel("date")
```



Adding vertical lines



General settings for finetuning

Use style sheets to change the [general settings](#)

- `sns.set()` for the 'default' of seaborn
 - `sns.set_theme(style="white", palette= "pastel")` to change the style to white using pastel colors e.g.
 - `sns.reset_defaults()` to reset to the default of Python
- Many more options: [click here](#)
- If the [same setting](#) should be used for all points/lines/areas, you can change the general settings



Conclusions



Conclusions

- Library pandas offers a less flexible framework and less lengthy code.
- Library seaborn offers a wrapper around matplotlib which is still quite flexible and can produce **high-quality graphics** for publications.
- Sometimes seaborn does not suffice and one can use the more flexible library matplotlib, this can result in quite lengthy code.



Exercises

→ Open file *AdvancedGraphics-Exercises.pdf*

→ Do Exercise 3 and 4

