Getting started with graphics

Erasmus Q-Intelligence B.V.

Data Science and Business Analytics
Programming

Content

Built-in pandas graphics vs library `seaborn`

The usual suspect

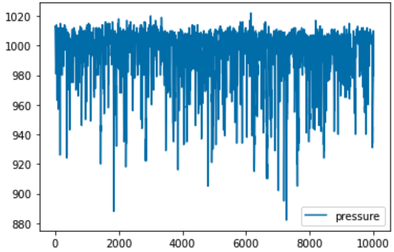
Function `pd.plot()`:

- Creates plot for data frame
- Lineplot if nothing is defined
- Possible to define the type to create other type of plots

Line plot

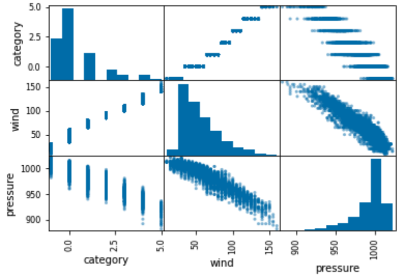
```
import pandas as pd

storms = pd.read_csv("storms.csv")
storms[["pressure"]].plot()
```



Scatterplot matrix

```
pd.plotting.scatter_matrix(storms[["status", "category",
                                   "wind", "pressure"]])
```



Built-in pandas graphics

+ Simple plotting interface
+ Allow the user to create quick plots for exploring the data
+ Simplified wrappers for the matplotlib AP

– Results can be visually lacking

The library matplotlib

- Launched in 2003 and still actively developed and maintained
- Most flexible and complete data visualisation library out there
- MATLAB style and interface
- Highly flexible and complete
- Low-level code can result in lengthy code
- Not always possible to use as quick analysis tool

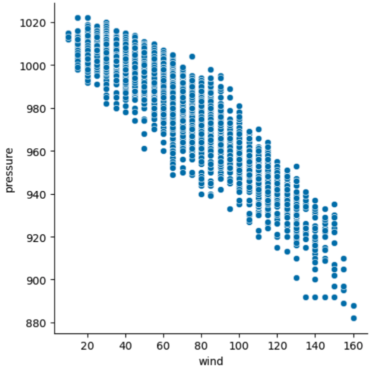
The library seaborn

- High-level interface for `matplotlib`
- Built-in functions for most statistical graphics
- Handles large amount of data with ease
- Integrates closely with pandas data structures
- Known for its nicely looking default style and color palettes

Basic plots

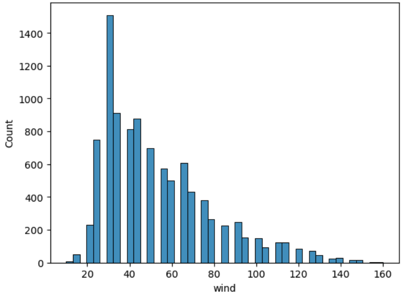
Scatterplot

```
import seaborn as sns
sns.relplot(data = storms, x = "wind", y = "pressure")
```



Histogram

```
sns.histplot(data = storms, x = "wind")
```



Histogram: number of bins

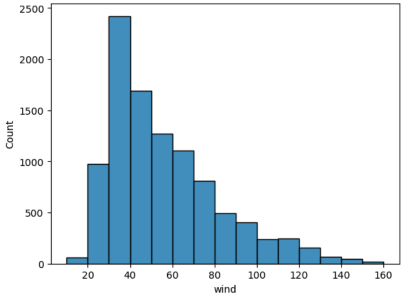
⟶ For histograms, it is always a good idea to play with the number of bins

- Number of bins can be specified with argument `bins`
  - If bins is an integer, it defines the number of equal-width bins in the range.
  - If bins is a sequence, it defines the bin edges.
  - If bins is a string, it is one of the binning strategies supported by numpy.histogram_bin_edges: 'auto', 'fd', 'doane', 'scott', 'stone', 'rice', 'sturges', or 'sqrt'.

.

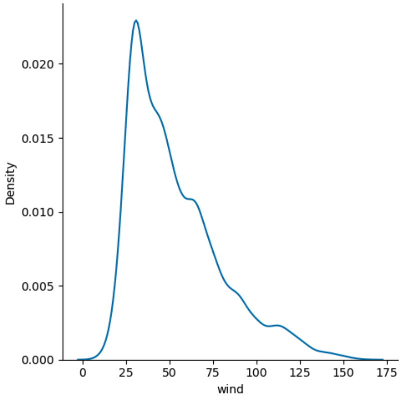
Histogram: number of bins

```
sns.histplot(data = storms, x = "wind", bins = 15)
```



Density plot

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



Density plot: kernel and bandwidth

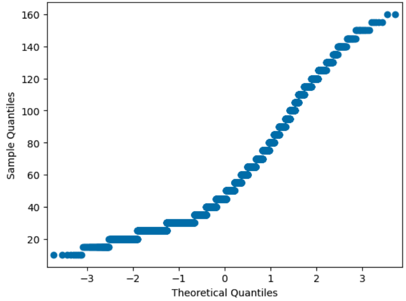
- Density estimate depends on the kernel and smoothing bandwidth
- Used (or default) Gaussian kernel is symmetric and therefore not optimal for asymmetric distributions

$\longrightarrow$ Still useful to get an insight on the shape of the distribution, but be aware of those issues

Quantile-quantile plot

A quantile-quantile plot is not straightforward with sesaborn nor pandas, but it is with the statsmodels library.

```
import statsmodels.api as sm
sm.qqplot(storms['wind'])
```



Quantile-quantile plot: straight line?

$\longrightarrow$ Plot sample quantiles against theoretical quantiles

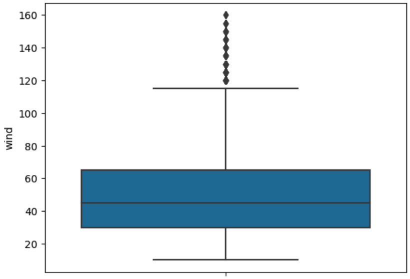
$\longrightarrow$ If the distributional assumption holds, the points form almost a straight line

$\longrightarrow$ By default the normal distribution is used

$\longrightarrow$ Distribution can be specified with argument `dist`

Boxplot

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



Boxplot statistics

Upper whisker	Largest point still within $1.5 \cdot IQR$ of the upper quartile
Top of box	Upper quartile (i.e., 75% quantile)
Middle line	Median (i.e., 50% quantile)
Bottom of box	Lower quartile (i.e., 25% quantile)
Lower whisker	Smallest point still within $1.5 \cdot IQR$ of the lower quartile
IQR	Interquartile range (i.e., difference between upper and lower quartile)

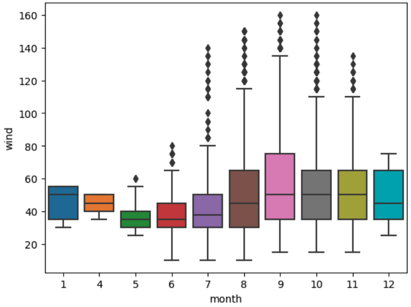
$\longrightarrow$ No assumption about statistical distribution

$\longrightarrow$ But: definition of whiskers assumes some degree of symmetry

Conditional boxplot

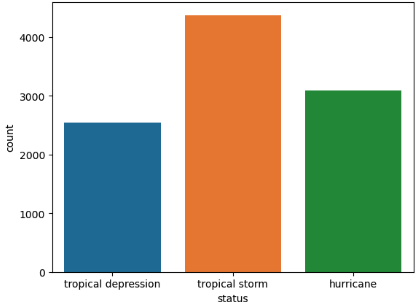
- Add additional x variable to create conditional boxplot

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



Barplot

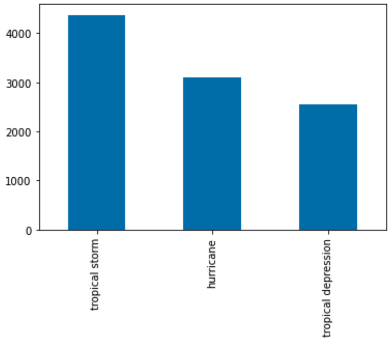
```
sns.countplot(data = storms, x = 'status')
```



Barplot using pandas

⟶ In pandas we have to calculate the height of the bars first with `value_counts`, more code needed and less attractive.

```
storms['status'].value_counts().plot.bar()
```



Time series plot

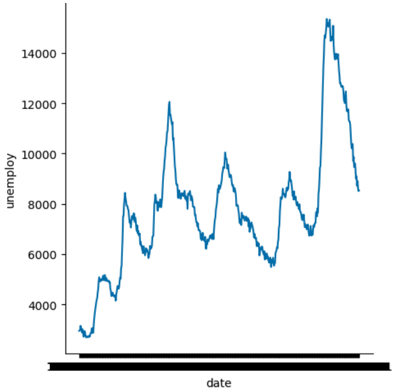
⟶ Simply change the argument `kind` to `"line"` instead of `"point"` in the function `relplot()` to draw connected line instead of scattered points

⟶ Example: US economic time series

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

Time series plot

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



plt.show()

- Simply calling the plotting function will give some additional, non-informational text to the plot call in Python
- Use `show()` from the library `matplotlib.pyplot` to avoid this
- `matplotlib.pyplot` is usually imported as `plt`

Some graphics

For a complete list of common graphics, click here. Important ones include:

`relplot()`	Points or lines
`axhline()`	Horizontal lines
`axvline()`	Vertical lines
`boxplot()`	Box and whiskers plot
`displot()`	Density estimate
`text()`	Text
`heatmap()`	Heat maps

$\longrightarrow$ Use appropriate graphics!

Conclusions

Conclusions

- Most figures are easily accessed via `seaborn`
- Results of `seaborn` are visually appealing
- Use `plt.show()` to show the plot
- Use scripts for reproducibility of the plots

Exercises

Load the `patents` data from the `patents.Rds` file, and do Exercise 1.1.