# Integratie Externe Functionaliteiten Les 8 : Matplotlib andere soorten grafieken

Matplotlib heeft buiten de standaard grafiek met punten verbonden met een lijn nog veel andere manieren om data weer te geven. We bekijken een aantal andere veel voorkomende grafiekstijlen.

Voorbeelden:

https://matplotlib.org/stable/gallery/index.html

## Scatter

**Scatter** is een voorstelling van losse punten. Data waar dus geen lijn tussen zit, maar wel een connectie tussen 2 waarden. We kunnen dan naar het volledige plaatje kijken of er groepjes van data zijn.

De x-as heeft ook meestal geen oplopende waardes. Dit zien we wel meestal terug bij een lijngrafiek.

vb. Leeftijden en woonplaatsen.

Bij een **scatter** plot is alle code hetzelfde. Er zijn 2 array's met evenveel waardes. Maar in plaats van de **plot()** functie gebruiken we de functie **scatter().**

Maak 2 array's met willekeurige getallen die door elkaar staan.

Maak hiervan een scatter plot met de functie scatter()

Als we meerdere datasets willen vergelijken voegen we meerdere array's toe. Telkens afgesloten met de scatter() functie. Op het einde roepen we dan de show() functie op.

# Styling

De **kleur** veranderen we op dezelfde manier met het color of c argument. Ook om de marker aan te passen gebruiken we hetzelfde argument. De styling blijft dus op eenzelfde manier werken.

We kunnen ook een array met kleuren meegeven. Zo krijgt elke markering een andere kleur. We roepen deze array dan op in het argument color van de scatter functie.

Let op! De lijst me kleuren moet wel even lang zijn als de array's met data.

Een andere manier om groepen duidelijk te tonen is het gebruik van een kleurenbalk. Dan bepaalt de positie van de markering de kleur.

De array voor de kleuren stellen we gelijk aan de x of y as.

We maken dan gebruik van het argument cmap.

Om ook de kleurenlegende te tonen voegen we voor show() de functie colorbar() toe.

Voeg een variabele kleur toe en stel deze gelijk aan de y-as.

Voeg de argumenten color met de waarde kleur en cmap met waarde 'Spectral' toe aan de scatter() functie.

Voeg ook nog de functie plt.colorbar() toe.

Om de kleurmap op te draaien voegen we achteraan _r, Spectral_r.

Lijst met mogelijke colormaps:

https://matplotlib.org/stable/tutorials/colors/colormaps.html?highlight=colormap

## Grootte

Om de grootte van elke marker individueel aan te passen gebruiken we dezelfde aanpak. We maken een extra, even lange, array aan met groottes voor de markers.

We roepen deze array op in het argument s in de scatter() functie.

Omdat de markers kunnen overlappen voegen we meestal ook het argument alpha toe. Dit zal de marker doorzichtig maken. Hier geven we een waarde tussen 0 en 1 in. 0 is dan helemaal doorzichtig en 1 volledig gevuld.

# Staafdiagram

We maken gebruik van een staafdiagram als we een frequentieverdeling willen weergeven. We geven dan een hoeveelheid per categorie weer.

De werkwijze blijft dezelfde. De gebruikte functie die gebruikt wordt is **bar().**

Om een horizontale staafdiagram te maken gebruiken we dan **barh()**.

> Maak een staafdiagram die de waarde van 4 categorieën weergeeft.
>
> Zet ze nadien ook horizontaal. Wat moeten we aanpassen?

De styling blijft ook hetzelfde werken. Een veelgebruikt argument bij staafdiagrammen is de breedte, width. De standaardwaarde hier is 0.8.

# Histogram

Een histogram geeft ook een frequentieverdeling weer. Met dat verschil dat ze visueel tegen elkaar staan.

Er wordt 1 array gebruikt en de waardes worden geteld. Hoeveel waardes liggen er tussen a en b, c en d,… .

Deze verdeling noemen we **bins**. Standaard zijn er 10 **bins**.

De array wordt van klein naar groot gesorteerd. Dan wordt deze in 10 gelijke delen verdeeld. Vervolgens wordt er geteld hoeveel waardes er per deel zijn. Dit zal de hoogte, waarde op de y-as, worden.

We gebruiken de functie hist(). Hier wordt dus maar 1 array meegegeven. De 2de waarde is het aantal **bins**. We kunnen hier ook het argument **bins** gebruiken, maar mogen ook alleen een int invullen.

Gebruik volgende array om een histogram te maken.

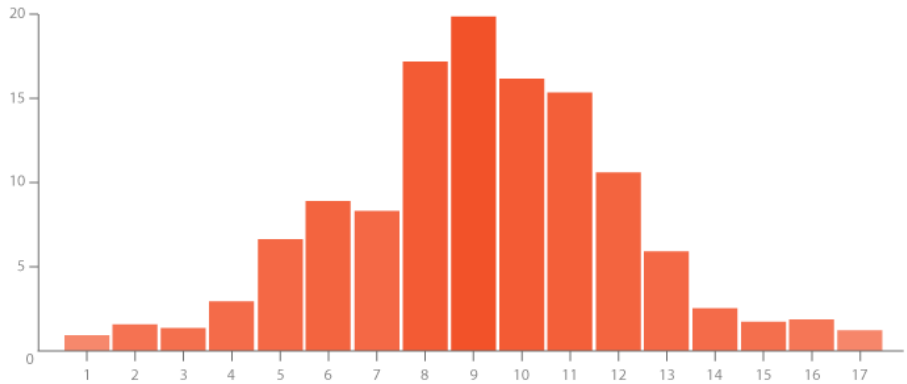
Verdeel deze onder in 5 bins.

lengte = [189, 185, 195, 149, 189, 147, 154, 174, 169, 195, 159, 192, 155, 191, 153, 157, 140, 144, 172, 157, 181, 182, 166, 167]

De stijl kunnen we ook aanpassen zoals we al kennen.

## Gauss-curve

In de praktijk wordt een histogram vaak gebruikt om een **gauss-curve** of **klokkromme** weer te geven.



Hierbij is de data meestal gecentreerd rond een bepaalde waarde. Denk aan een IQ curve. Het grootste deel van de bevolking heeft een IQ rond de 100 punten. Deze deint dan uit, zowel naar boven als naar beneden.

Dit wordt ook veel gebruikt bij machine learning. Dit omdat data meestal rond een bepaalde waarde centreren. Maar zo kunnen we uit data afleiden hoeveel er wordt afgeweken van de norm.

Numpy kan ook een willekeurige array genereren rond een bepaald punt. Deze is gebaseerd op de formule van een gauss-curve

X = np.random.normal(gecentreerde waarde, standaard afwijking, aantal waardes)

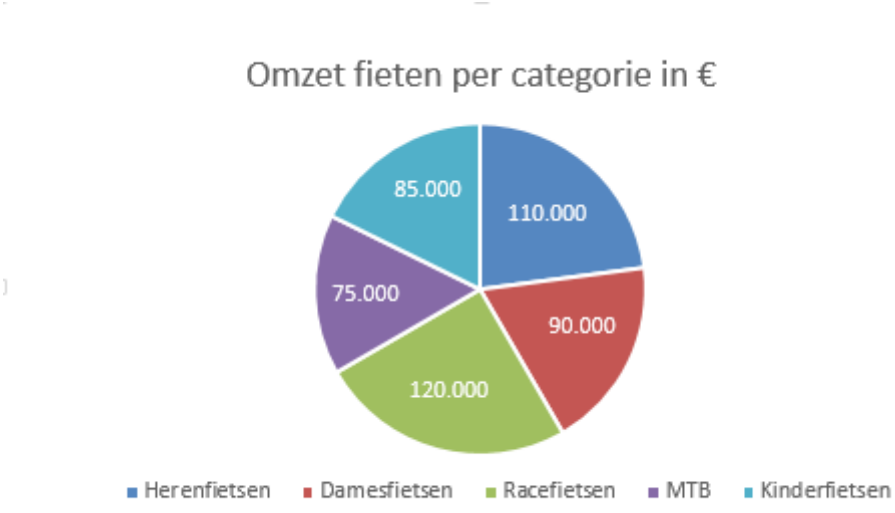
Maak een array van 500 waardes gecentreerd rond het nummer 200 en met een afwijking van 10.

Print deze af.

Geeft deze array weer in een histogram met 20 bins.

# Taartdiagram

Een taartdiagram dient om een verhouding tussen bepaalde categorieën weer te geven.



We gebruiken weer 1 array met waardes. De grootte van het deel wordt bepaald met een formule:  **waarde/som van de waardes.**

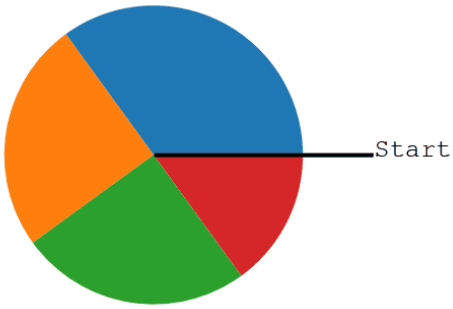
De functie **pie()** zal de taartdiagram weergeven.

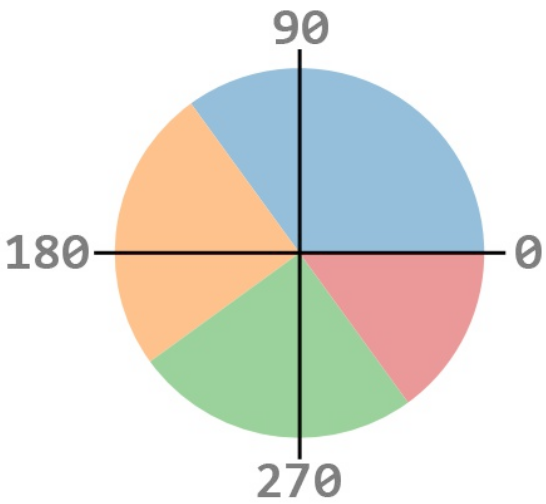
Maak een array met 4 waardes tussen 0 en 100.
Toon deze in een taartdiagram.

Een cirkel zonder duiding is niet zo... duidelijk. We gaan daarom labels toevoegen. Hiervoor maken we een 2de array aan. Deze voegen we dan toe met het argument **labels**.

Voeg labels toe aan de taartdiagram.

De startpositie van de cirkel ligt vast. We kunnen ook deze starthoek aanpassen. Dit met het argument **startangle**.



Om het nog duidelijker te maken voegen we een legende toe. Dit met de functie **plt.legend().** Hierin kunnen we met het argument **title** een titel toevoegen.

Om 1 deel uit te lichten snijden we dat deeltje uit de taart.
We maken een array die even lang is als de originele. Hierin plaatsen we alle waardes op **0.**
De waarde van het uit te snijden deel passen we aan. We vullen de afstand in hoe ver deze van het centrum moet staan.
Gebruik dit heel subtiel, gebruik een kleine waarde. Bij voorkeur een float.
Vervolgens vullen we deze array in bij het argument **explode**.

## Styling

De styling kunnen we aanpassen met een array met kleuren. Die dan aangevuld wordt in het argument **c** of **colors**.

Als extra stijlelement is er een schaduw mogelijk. Dit met het argument **shadow.** Deze is dan ingevuld met een boolean.

Pas de taartdiagram aan met:
een schaduw, ligt het grootste deel uit, voeg een legende toe, pas de kleuren aan en geeft de taart een slag.

# Subplots

Nu tonen we elk grafiek apart. We kunnen ook meerdere lijnen op 1 grafiek krijgen door de **show()** functie als laatste aan te roepen. Maar als we grafieken naast elkaar willen vergelijken gebruike we de functie **subplot()**.

We roepen deze functie telkens aan voor de grafiek functie. En we gebruiken maar 1 **show()** functie op het einde.

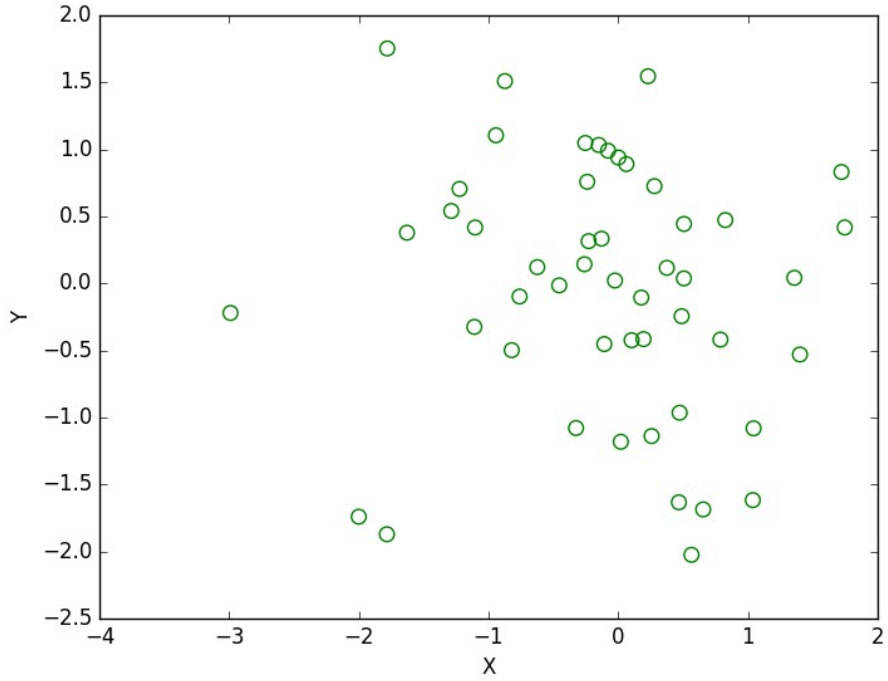
De subplot() functie heeft 3 argumenten nodig:

Het aantal rijen.
Het aantal kolommen.
De hoeveelste grafiek dit is.

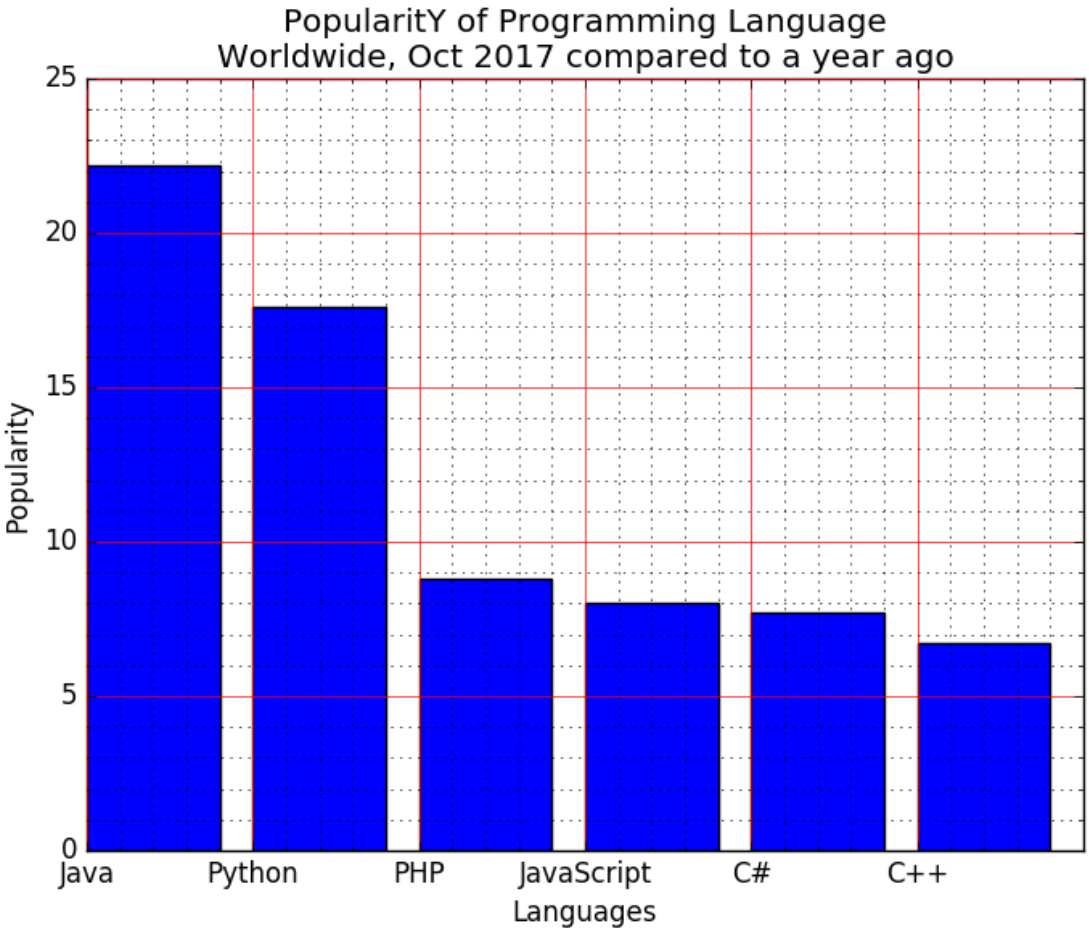
Plaats de laatste 3 grafieken die we aanmaakten naast elkaar.
Plaats de eerste 4 grafieken in 2 rijen en 2 kolommen.

OEFN:

Maak volgende grafieken na, gebruik willekeurig gegenereerde data array's.
1)



2)
programmeertalen: Java, Python, PHP, JavaScript, C#, C++
gebruik: 22.2, 17.6, 8.8, 8, 7.7, 6.7




3)