Programmeeropdracht 6: Extra functionaliteit, data analyseren en grafieken maken

We zijn inmiddels al bij de laatste programmeeropdracht! In de afgelopen weken heb je al veel gestoeid, geprobeerd, fouten gemaakt, successen behaald, en bovenal, geleerd! In deze programmeeropdracht ga je kennismaken met een paar extra handige functies, het importeren van extra functionaliteit, het berekenen van beschrijvende statistieken en het maken van grafieken. Heel veel succes!

Extra functies

Door het gebruik van functies kunnen we meer bewerkingen op onze data toepassen. Veel functies zitten standaard in Python, zoals if en print. Standaard zijn er nog vele andere functies, zoals de upper-functie. De upper-functie converteert alle karakters naar hoofdletters. Probeer maar:

```
intro = 'Goedemorgen'
print(intro.upper())
```

Naast dat we alle karakters kunnen omzetten in hoofdletters, kunnen we ook van alle karakters kleine letters maken door middel van de lower-functie. Probeer maar:

```
reactie = 'Hetzelfde!'
print(reactie.lower())
```

We kunnen functies die gekoppeld zijn aan een variabele onderzoeken. Dit kan door de variabele naam in te typen. Dit brengt ons bij de help functie voor een functie, die inzichtelijk maakt welke functionaliteit gekoppeld is aan een variabele. Probeer maar eens.

De upper- en lower-functie zijn vooral erg handig wanneer je in een databestand waarden hebt staan die per ongeluk hoofdletters of kleine letters bevatten die je liever niet hebt. Soms kan het ook voorkomen dat je databestand waarden bevat met karakters die er niet thuishoren. Bijvoorbeeld: bij het invullen van een adaptieve vragenlijst heeft de respondent een typefout gemaakt. We kunnen een specifiek karakter uit een string waarde vervangen door middel van de replace functie. Probeer maar:

```
antwoord1 = "Eens"
antwoord2 = "ja["
```

```
# Vervang 'E' door 'e'

print (antwoord1.replace("E","e"))

# Vervang typfout '[' door niks ''

print (antwoord2.replace('[',''))
```

Naast de standaard functies zijn er zijn nog heel veel andere *functies* bijgeleverd, maar die zijn niet standaard beschikbaar. Voor deze functies moeten we Python vertellen dat we deze functies beschikbaar willen. Dit doen we door gebruik te maken van het import commando. Een voorbeeld van een functie die vaak voorkomt bij het programmeren is de capaciteit om random (Nederlands: willekeurig) een getal te genereren. Python heeft deze functionaliteit, maar die is niet standaard. Dus, dit moeten we importeren. Probeer maar:

```
import random
p = random.random()

print(p)
```

Zonder het importeren van de random functionaliteit krijg je een foutmelding. Probeer maar. Het gaat om de laatste regel van die foutmelding; "*NameError: name 'random' is not defined*". Deze is in zo normaal mogelijk Engels geformuleerd en redelijk te begrijpen: Python kent de naam random niet en kan er dus niks mee! Om gebruik te maken van de random-functie moet je dus de random bibliotheek importeren en alle functies daarin beschikbaar komen in jouw programma.

Functies kunnen bepaalde waarden aannemen, die we argumenten noemen, die beïnvloeden hoe een functie zich uit. Deze argumenten zijn gespecificeerd tussen de haakjes. In het vorige voorbeeld hebben we geen argumenten gebruikt en daarom alleen haakjes geopend en gesloten. Maar stel dat we een random nummer willen genereren tussen de 0 en 10 en niet uit een uniforme distributie tussen 0 en 1 (zoals random.random functie doet). Dit kan door middel van random.uniform functie, met de argumenten (1, 10). Probeer dit maar eens.

Je kunt deze argumenten ook definiëren door middel van variabelen:

```
import random
x = 0
y = 10

random_getal = random.uniform(0,10)

print(random_getal)
```

Documentatie

- De functies die beschikbaar zijn in de random library kan je vinden in de documentatie: https://docs.python.org/2/library/random.html
- Er nog meer libraries die horen bij standaard-Python, die vind je op: https://docs.python.org/2/library/

Numpy

Met het commando import kunnen we dus extra functionaliteit toevoegen aan Python. 'random' wat we hiervoor in de opdracht hebben geïmporteerd noemen we in Python ook wel een *library* (Nederlands: bibliotheek). Dat lijkt een gek woord, maar je kunt een library dus zien als een bibliotheek met nieuwe functies.

Een voorbeeld van een uitgebreidere bibliotheek is de numpy-library. Een overzicht, documentatie en voorbeelden kan je vinden op http://www.numpy.org. We kunnen numpy ook importeren met het commando import. In de numpy-library vinden we functionaliteit om data te analyseren. Dat is handig!

In de numpy-library vinden we de functie array die we nu eerst gaan bespreken. Een array is een datastructuur met daarin meerdere elementen. Er zijn meerdere manieren om een array te creëren, zie het voorbeeld hieronder. Van de for-loops ken je nog de opdracht range. Een vergelijkbare functie in de numpy bibliotheek is de arange functie. Zo zijn deze twee stukken code equivalent:

```
# versie 1
r_list = list(range (10))
print (r_list)

# versie 2
import numpy
r_array = (numpy.arange(10))
print (r_array)
```

Probeer maar. Zoals je ziet produceren beide versies de getallen van 0 tot 9. Belangrijk daarbij is wel dat het verschillende data types zijn; de range functie produceert een lijst en de arange functie produceert een array. Dit verschil is belangrijk, want deze verschillende data types worden verschillend gebruikt (andere functionaliteit).

We kunnen de inhoud van een array bekijken vergelijkbaar met hoe dat bij een lijst gaat. Probeer maar:

```
import numpy
data = (numpy.arange (10))
print (data [1:5])
```

```
print (data[1:5:2])
```

Kijk goed welke getallen worden geprint op het scherm en waarom. Als je niet helemaal zeker bent waarom deze getallen worden geprint, bekijk dan ook het gehele array met het commando print (data).

Het voordeel van een array is dat het meerdere dimensies kan hebben. Een veel voorkomende vorm is een tabel, met meerdere rijen en kolommen. Een tabel kunnen we representeren door het creëren van een tweedimensionale array. Wanneer je een array hebt met meerdere dimensies (een tabel met meerdere rijen) werkt het als volgt. We creëren nu een array met random getallen door middel van de functie random.random in numpy. Probeer maar.

```
import numpy
data = numpy.random.random (size = [5, 3])
print (data)

# eerste rij, eerste kolom
print (data[0,0])
# eerste rij, laatste kolom
print (data[0,-1])
# tweede rij, laatste kolom
print (data[1,-1])
```

Zoals je hier kunt zien scheiden we de kolommen en rijen die we willen zien door middel van een komma. Het eerste getal tussen de vierkante haken geeft het rij-nummer en het tweede getal geeft het kolom-nummer. Door middel van een negatief getal kunnen we vanaf het einde tellen (zoals we gedaan hebben met -1, wat de laatste rij/kolom oplevert). Daarnaast kunnen we ook bepaalde dimensies van een array bekijken. Probeer maar:

```
import numpy
data = numpy.random.random (size=[5, 3])
print (data)

# alle rijen, eerste kolom
print (data[:,0])
# eerste rij, alle kolommen
print (data[0,:])
```

We kunnen ook rekenen over de items in een array. Bijvoorbeeld, we kunnen items van een array optellen. Daarbij kunnen we ook een bepaalde rij of kolom optellen. Probeer maar:

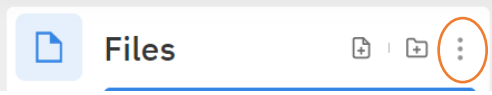
```
import numpy
data = numpy.random.random (size = [2, 2])
print (data)

# alle getallen in de array optellen
print (numpy.sum(data))

# alle getallen in elke rij optellen
print (numpy.sum (data, axis = 0))

# alle getallen in elke kolom optellen
print (numpy.sum (data, axis = 1))
```

Een array kan opgeslagen worden in een tekst file, die ook door andere programma's geopend kan worden. Dit kunnen we doen door gebruik te maken van de numpy.savetxt en numpy.loadtxt commando's te gebruiken. Probeer maar:

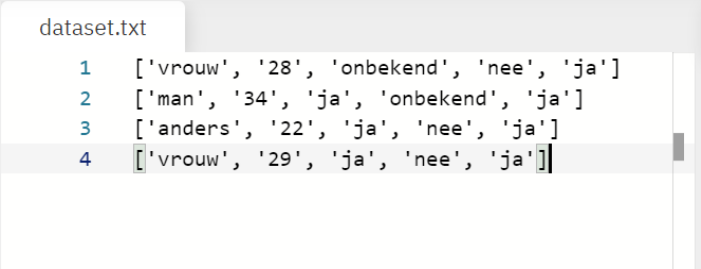
```
import numpy
data = numpy.random.random(size = [2, 2])
print (data)

# de array wordt opgeslagen in het bestand data.txt
numpy.savetxt("data.txt", data)

#het opgeslagen data.txt bestand wordt ingeladen onder de naam saved_data
saved_data = numpy.loadtxt("data.txt")
print (saved_data)
```

Data analyse met Numpy

Nu je bekend bent met arrays in numpy, kunnen we data gaan analyseren van een vragenlijst. Hiertoe moeten we eerst een tekstbestand met vragenlijstdata uploaden in repl.it. Ga naar Blackboard → week 4.7 en download het tekstbestand 'dataset.txt'. Ga vervolgens weer terug naar repl.it. Je kunt dataset.txt uploaden in repl.it door op de drie puntjes te klikken vlak boven de bestanden (zie onderstaande screenshot) en dan 'Upload file'. Het is gelukt als dataset.txt naast de overige bestanden staat.



Als het uploaden goed is gelukt ziet jouw dataset.txt er zo uit:



Om met de data te kunnen rekenen moeten we het eerst inladen. Dit lijkt een beetje op het openen van tekstbestanden zoals we dat in de vorige programmeeropdracht hebben gezien. Alleen nu maken we er niet een file object van maar een numpy array. Gebruik de onderstaande code om dataset.txt als numpy array op te slaan in de variabele ‘data’:

```
import numpy

data = numpy.loadtxt("dataset.txt", dtype = str, delimiter = ", ")

#Inspecteer de data
print (data)
```

In de functie *loadtxt* zie je dat er drie argumenten zijn gegeven. De eerste is het bestand zelf. Vervolgens ‘dtype = str’. ‘Dtype’ staat voor het data type dat de array gaat krijgen. We hebben dit argument de waarde ‘str’ meegegeven wat staat voor string. Kortom, het tweede argument zorgt ervoor dat we een array maken met het datatype ‘string’. Het derde en laatste argument ‘delimiter’ staat voor het scheidingsteken dat in het bestand wordt gebruikt. Kijk een goed naar dataset.txt, je ziet dan dat elk element in de lijst gescheiden is door een komma. Ook zit er na elke komma een beetje witruimte door een spatie. Het scheidingsteken bevat dus zowel de komma als de spatie: ‘, ‘. Als je een eigen dataset gaan inladen als array kan het zijn dat jij die spaties niet hebt of hele andere scheidingstekens gebruikt. Let daar goed op en pas dan de delimiter aan.

Als je naar de output van print (data) kijkt, dan valt je misschien op dat de array nog niet helemaal is zoals wij willen. Ten eerste bevatten de eerste waarden en laatste waarden in elke rij

een ‘[‘ of een ‘] ‘. Dit kun je gemakkelijk zien door aan Python een eerste waarde of laatste waarde van de rij op te vragen door print (data[0,0]) of print (data[0,-1]). Probeer maar eens.

Daarnaast valt op dat elke waarde in de array twee soorten aanhalingstekens bevat. Dit komt omdat we van waarden zoals ‘man’ en ‘28’ strings hebben gemaakt terwijl de waarden al ‘’ hadden. Je krijgt dan “ ’man’ ” en “ ’28’ ”. We willen zowel de ‘[‘ en ‘]’ bij de eerste en laatste waarden weghebben, maar ook de enkele aanhalingstekens. We kunnen hiervoor de replace functie gebruiken zoals we dat eerder in deze opdracht hebben gezien. Typ de onderstaande code over. Probeer goed te begrijpen wat er gebeurt.

```
# De shape functie haalt het aantal rijen en kolommen op van een numpy array
shape = data.shape

# Voor elk element i in de reeks van 0 tot het aantal rijen (shape[0] = 5))
for i in range(0,shape[0]):
  # Sla de waarde plek 0 en plek 4 in een rij (i) op in een variabele
  str1 = data[i,0]
  str2 = data[i,4]
  # Gebruik de replace functie op het karakter '[' of ']' ter vervangen door niks
('')
  data[i,0] = str1.replace('[','')
  data[i,4] = str2.replace(']','')

print (data)

# Voor elke kolom in de dataset
# En binnen een kolom, in elke rij
for kolom in range(0,shape[1]):
  for rij in range(0, shape[0]):
    # Pak de waarde in de huidige rij en kolom
    item = data[rij,kolom]
    # En vervang de hoge komma's met niks
    data[rij,kolom] = item.replace("'", '')

print(data)
```

Als je nu naar output van print (data) kijkt zul je zien dat alles netjes is opgeschoond. Top!

Nu we een array hebben die is opgeschoond van foutjes kunnen we beginnen met analyseren. Allereerst gaan we kijken hoe we kunnen berekenen hoe vaak een bepaald antwoord is gegeven op een bepaalde vraag. Bijvoorbeeld: hoe vaak is er met ‘man’ geantwoord op de vraag ‘met welk gender identificeert u zich?’ Er zijn in ieder geval twee manieren waarop we dit in Python

kunnen aanpakken. De eerste manier is met een for-loop. Eerst zorgen we ervoor dat de kolom uit de data waarin de antwoorden op de gender-vraag staan in een variabele wordt opgeslagen, in dit geval *genders*. De antwoorden op de gender-vraag staan op index 0 in de dataset. Vervolgens gebruiken we een for-loop om voor elk element in kolom die we hebben opgeslagen in *genders*, dat is dus ieder gegeven antwoord, te controleren of het element gelijk is aan 'man'. Zo ja? Dan tellen we het aantal mannen wat we tot nu toe hebben gezien op. Neem de volgende code over en probeer per regel te begrijpen wat er gebeurd:

```
# Eerst nemen we de gender kolom apart, in dit geval is dat de kolom op index 0
genders = data[:,0]

# Vervolgens maken we een variabele aan om de tel van het aantal mannen in bij te
houden
aantal_mannen = 0

# In een for-loop gaan we elk element in genders bij langs. Is het een man? Dan
tellen we 1 op bij de tel variabele aantal_mannen.
for gender in genders:
  if gender.lower() == "man": #lower() maakt eventuele hoofdletters klein
    aantal_mannen = aantal_mannen + 1

print("Aantal mannen in de data: ")
print(aantal_mannen)
```

Een andere manier om het bovenstaande te berekenen is met het attribuut *count*. Dit attribuut hebben we in programmeeropdracht 2 ook al gezien. Het attribuut 'count' werkt alleen bij een lijst. Inspecteer de onderstaande code en neem het over in Replit:

```
# Eerst nemen we de genders kolom apart
genders = data[:,0]

print("Aantal vrouwen in de data: ")
aantal_vrouwen = list(genders).count("vrouw")
print(aantal_vrouwen)
```

Nu heb je twee manieren gezien hoe je voor een enkele vraag uit de vragenlijst kunt berekenen hoe vaak een bepaald antwoord is gegeven. We gaan nu naar een andere beschrijvende statistiek kijken: het gemiddelde. Hiertoe maken we gebruik van de functie *numpy.mean.* Met deze functie gaan we berekenen wat de gemiddelde leeftijd is van de respondenten. Inspecteer de onderstaande code en neem het over:

```
# Eerst nemen we de leeftijd kolom apart, de leeftijden staan op index 1
leeftijden = data[:,1]
print(leeftijden)

# Nu zijn alle waarden nog strings. We kunnen alleen gemiddeldes berekenen met
floats (floats zijn getallen die decimalen kunnen hebben)
leeftijden = leeftijden.astype(float)
print(leeftijden)

# Nu kunnen we het gemiddelde berekenen
print(numpy.mean(leeftijden))
```

Het kunnen berekenen van de het aantal mannen/vrouwen anders en de gemiddelde leeftijd is leuk, maar vaak zijn we ook geïnteresseerd in hoe respondenten op de vragen hebben gescoord. In dit geval willen we graag weten op hoeveel vragen er 'ja' is geantwoord, maar met kleine aanpassingen is deze code toepasbaar op alle soorten antwoordmogelijkheden.

Een gemakkelijke manier om te kunnen bepalen hoe vaak er 'ja' is geantwoord is om de antwoorden te hercoderen: we veranderen de ja's in het getal '1' en de nee's in '0'. Dan hoeven we alleen de som te berekenen van de rij om erachter te komen hoeveel ja's we hebben. Immers, bij drie ja's is het totaal van de rij: 3. Er is alleen nog één probleem: we hebben ook waarden die op 'onbekend' staan. Dat zie zijn afkomstig van vragen die door de respondent niet zijn ingevuld, mogelijk omdat adaptiviteit in de vragenlijst ervoor heeft gezorgd dat de vraag niet gesteld is. Deze 'onbekend' zijn eigenlijk *missing values*. Python kan geen 'onbekend' optellen bij 1. De oplossing is door de waarden 'onbekend' te vervangen met een missing value. Hiertoe kun je de functie *numpy.nan* te gebruiken. 'nan' staat voor missing value.

We kunnen met de volgende code *alle* ja's, nee's en 'onbekend' in de array vervangen. Probeer maar:

```
# Verander alle ja's in 1
data[data == 'ja'] = 1
# Verander alle nee's in 0
data[data == 'nee'] = 0
# Verander alle 'onbekend' in een missing value (nan)
data[data == 'onbekend'] = numpy.nan

print(data)
```

Nu we alle waarden van de vragen hebben gehercodeerd kunnen we het totaal van de scores per respondent berekenen. Elke rij in de data is één respondent. We pakken daar voor alleen de scores op de inhoudelijke vragen en nemen demografische vragen dus niet mee. De inhoudelijke vragen vinden we van index 2 tot en met 5. Probeer maar:

```
## Totaal score op vragen per respondent
scores = data[:,2:5]
print(scores)

scores = scores.astype(float)
print (scores)

# We berekenen de som per rij met 'nansum'. Deze functie kan de missende vragen
negeren. Met axis = 1 zorgen we ervoor dat de som per rij wordt berekend.
totaalsom_per_respondent = numpy.nansum(scores, axis = 1)
print(totaalsom_per_respondent)
```

We weten nu hoe we de totaalscore per respondent op een selectie van inhoudelijke vragen kunnen bepalen. We gaan nu kijken hoe we voor dezelfde vragen de gemiddelde score per respondent kunnen bepalen. Hiervoor kunnen we gebruik maken van de functie *numpy.nanmean*. De functie ‘nanmean’ kan in tegenstelling tot ‘mean’ rekening houden met de missende waarden in de data.

```
## Gemiddelde score op vragen per respondent
scores = data[:,2:5]
print(scores)

scores = scores.astype(float)
print(scores)

# We berekenen het gemiddelde per rij met 'nanmean'. Deze functie kan de missende
vragen negeren.
gemiddelde_score_per_respondent = numpy.nanmean(scores, axis = 1)
print(gemiddelde_score_per_respondent)
```

Grafieken

Het is vaak handig om je resultaten te visualiseren in een grafiek of zelfs een filmpje. Om je boodschap en conclusies goed over te brengen is het belangrijk dat je aandacht besteedt aan de presentatie zodat het voor je publiek duidelijk is. Vaak duurt het verzamelen van de data zelf uren/dagen/weken. Neem dus altijd de tijd om de bijbehorende grafiek netjes en duidelijk te maken.

Er is een standaardpakket om resultaten te visualiseren in Python: `matplotlib`. Het is een zeer omvangrijk pakket waarvan we maar een fractie nodig zullen hebben. Een goede tutorial kan je hier vinden: http://matplotlib.org/users/pyplot_tutorial.html.

Er bestaat een enorme variatie in de manier waarop data en resultaten gevisualiseerd worden. Denk altijd na hoe *jij* denkt dat je het best de informatie weer kan geven zodat de *gebruiker* de juiste conclusie trekt. Zoek vervolgens in de `matplotlib`-documentatie hoe je dat voor elkaar kunt krijgen.

Hieronder een paar voorbeelden waarmee je de basiscommando's van `matplotlib` leert kennen. In de opgave eronder ga je die gebruiken.

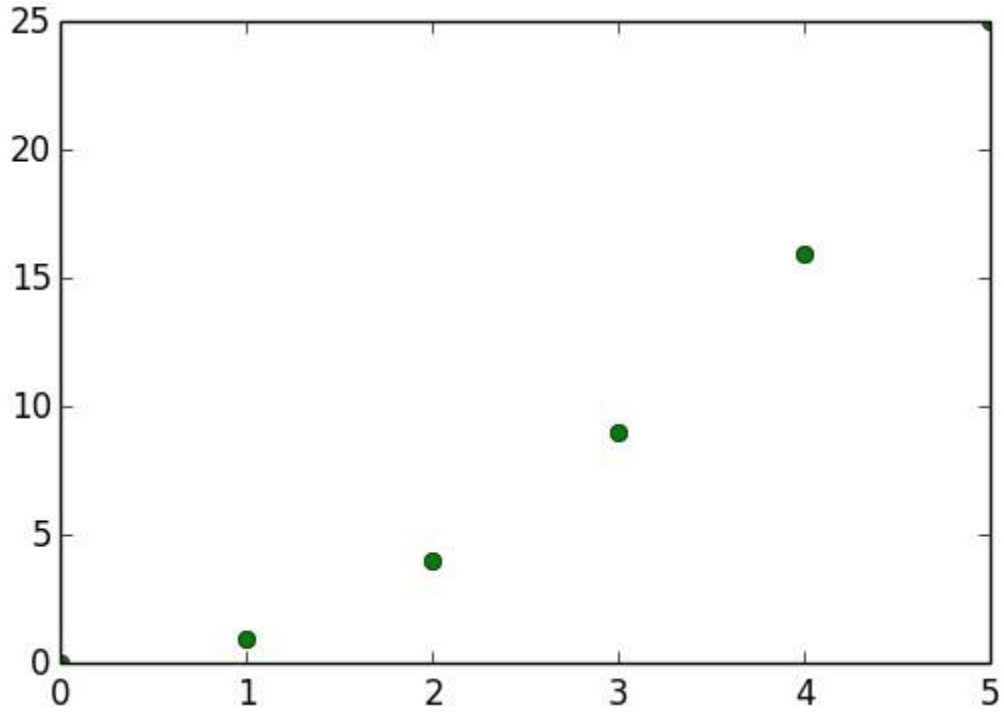
Een lijst met punten

We beginnen met het plotten van wat punten waarvan we de x-waardes (0,1,2,3,4,5) en de ywaardes (0,1,4,9,16,25) hebben. (In dit geval is het precies de functie x-kwadraat, maar dat hoeft natuurlijk niet). Om daar een grafiek van te maken doen we het volgende:

```
import matplotlib as mpl
import matplotlib.pyplot

# mpl.use('Agg') zorgt ervoor dat we de grafiek als PNG kunnen opslaan. Zie:
https://matplotlib.org/stable/tutorials/introductory/usage.html
mpl.use('Agg')

# de coordinaten per punt
x_coords = [0,1,2,3,4,5]
y_coords =[0,1,4,9,16,25]

# plot punten (y tegen x) met groene rondjes
```

```
matplotlib.pyplot.plot(x_coords,y_coords,'go')
matplotlib.pyplot.savefig('graph.png')
```

Hierbij is ervoor gekozen om de ‘marker’ (figuur waarmee elk punt weergegeven wordt) weer te geven als een groen rondje: het commando daarvoor is `'go'`. Als je een rode lijn had gewild had je voor `'r-'` kunnen kiezen. Probeer maar.

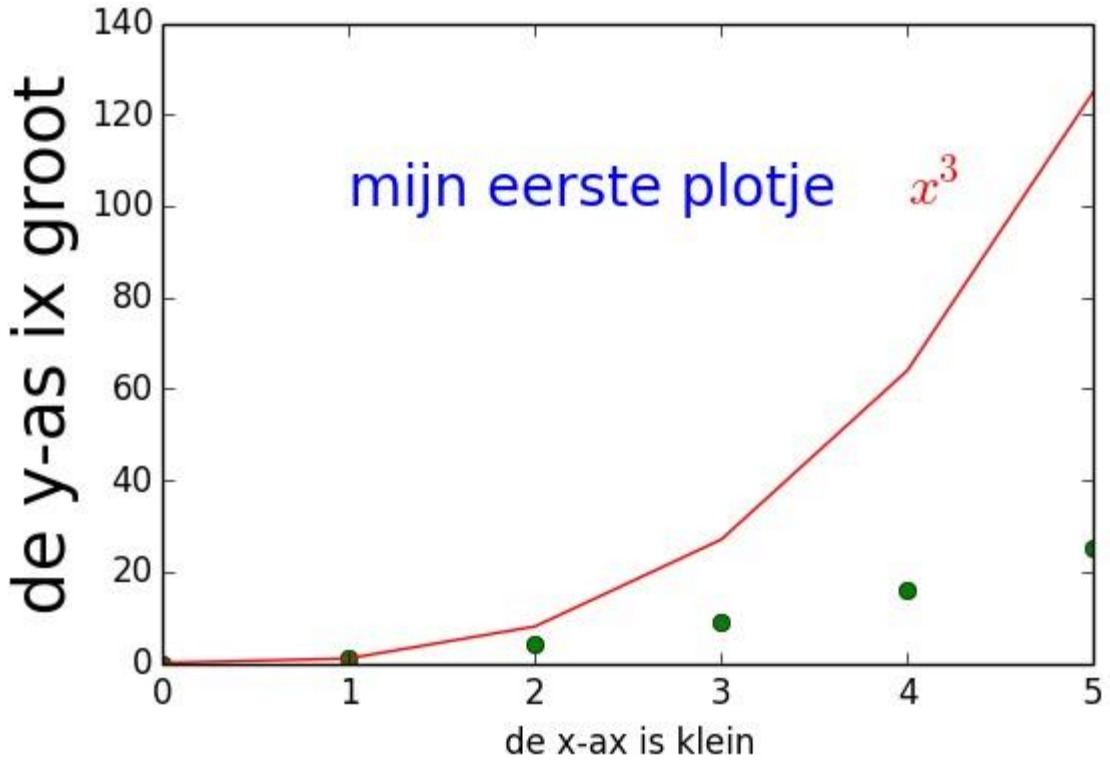


Meerdere grafieken en bijschriften

En natuurlijk behoren de assen labels te hebben, kan je 2 verschillende grafieken in 1 plot zetten en kan je zelf tekst weergeven. Om een extra grafiek (x^3) in de grafiek te zetten (met rode lijnen) doe je het volgende:

```
import matplotlib as mpl
import matplotlib.pyplot
mpl.use('Agg')

x_values  = [0,1,2,3,4,5]
x_squared = [0,1,4,9,16,25]
x_cubed   = [0,1,8,27,64,125]
```

```
# let op: grafiek met twee datasets!
matplotlib.pyplot.plot(x_values, x_squared, 'go', x_values, x_cubed, 'r-')

# Geef de x-as en y-as een label
matplotlib.pyplot.xlabel('de x-ax is klein')
matplotlib.pyplot.ylabel('de y-ax is groot', fontsize = 25)

matplotlib.pyplot.text(1.00,100., "mijn eerste plotje", color = 'blue', fontsize
= 20)
matplotlib.pyplot.text(4.00,100., "$x^3$", color = 'red', fontsize = 20)

matplotlib.pyplot.savefig('graph2.png')
```



Tot slot gaan we een histogram/staafdiagram maken. Een histogram/staafverdeling is een grafische weergave van een frequentieverdeling in klassen van gegroepeerde data. In een histogram/staafdiagram zie je kolommen die weergeven hoe vaak een bepaalde waarde (of een range van waarden) voorkomt. Probeer het volgende:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
numpy_hist=plt.figure()
plt.hist([1,2,1], bins=[0,1,2,3])
plt.savefig('graph3.png')
```

Wat er hier gebeurt is het volgende: de waarden zijn 1, 2 en 1. Van deze waarden wordt gekeken hoe vaak ze voor komen, en per waarde in een bin (bakje) gestopt. Er worden 3 bins gecreëerd; een van 0-1, een van 1-2, een van 2-3. In de bin 0-1 komen geen waarden (het is tot 1 en niet toten-met), want er zijn geen waarden onder de 1. In de tweede bin (1-2) komen de twee enen, en in de derde bin (2-3) komt 1 waarde (de 2). Dan krijg je dus het volgende figuur: