Grafieken

Contents

Frequenties kan je in een tabel weergeven, maar je kan ze ook grafisch weergeven. We bespreken hier kort een aantal voorstellingen.

We gebruiken hierbij de `matplotlib` library van Python. Deze wordt eerst kort toegelicht.

Tekenen met matplotlib

Matplotlib is een heel populaire library waarmee je eenvoudig grafieken kan maken. Je kan de library importeren met:

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

Als je een nieuwe grafiek wil starten, gebruik je best het volgende commando:

```
fig, ax = plt.subplots()
```

Als je bovenstaande commando's uitvoert, maak je een lege figuur `fig` met een assenstelsel `ax`. De meeste commando's om zaken te tekenen gebeuren vanaf het Axis-object (ax). Het figure-object heb je voor het maken van heel wat grafieken eigenlijk niet nodig. Je kan het eventueel negeren met:

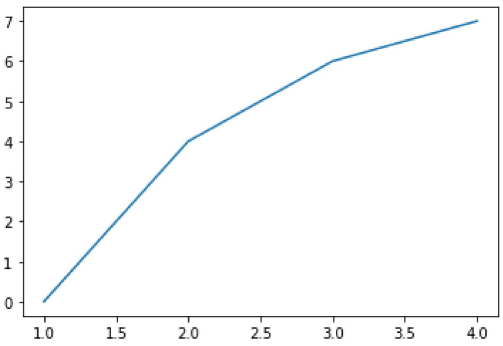
```
_, ax = plt.subplots()
```

Nu kan je figuren en grafieken tekenen. Je ziet hier een eenvoudig voorbeeld:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4], [0, 4, 6, 7])
```

```
[<matplotlib.lines.Line2D at 0x1fb36253dc0>]
```



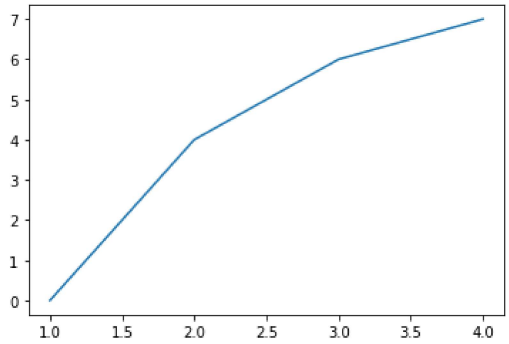
De grafiek zal automatisch zichtbaar zijn na het uitvoeren van bovenstaande commando. Lukt het toch niet dan kan je uitdrukkelijk opgeven van de figuur te tonen met:

```
fig.show()
```

Boven de grafiek zie je mogelijk nog iets vreemd staan [<matplotlib.lines.Line2D at 0x1fb36253dc0>]. Dit is in feite niets meer dan de return-waarde van het laatste commando `ax.plot([1, 2, 3, 4], [0, 4, 6, 7])`.

Je kan deze return-waarde 'inslikken' door het op te vangen in een anonieme variabele (_):

```
fig, ax = plt.subplots()
_ = ax.plot([1, 2, 3, 4], [0, 4, 6, 7])
```



Als je een grafiek wil exporteren, kan je het volgende commando gebruiken in plaats van `fig.show()`:

```
fig.savefig(bestandsnaam)
```

Afhankelijk van de bestandsnaam, zal Python het beeld in een bepaald formaat wegschrijven.

Grafieken hebben dikwijls een assenstelsel. Het is belangrijk om je assen ook steeds te benoemen. Dit kan je als volgt doen (doe dit je `show()` oproept):

```
ax.set_xlabel('tijd (s)')
ax.set_ylabel('tempertuur (K)')
```

Je ziet dat hier ook de eenheden vermeld zijn. Zo legt de grafiek zichzelf uit. Doe dit iedere keer voor elke grafiek! Bij een presentatie kunnen klanten heel erg moeilijk doen als deze informatie ontbreekt!

Indien je ook een titel wil geven aan de grafiek, dan kan dat met:

```
ax.set_title('Temperatuurverloop')
```

Rasterlijnen en legende kunnen de leesbaarheid van een grafiek drastisch verhogen. Het toevoegen is eenvoudig.

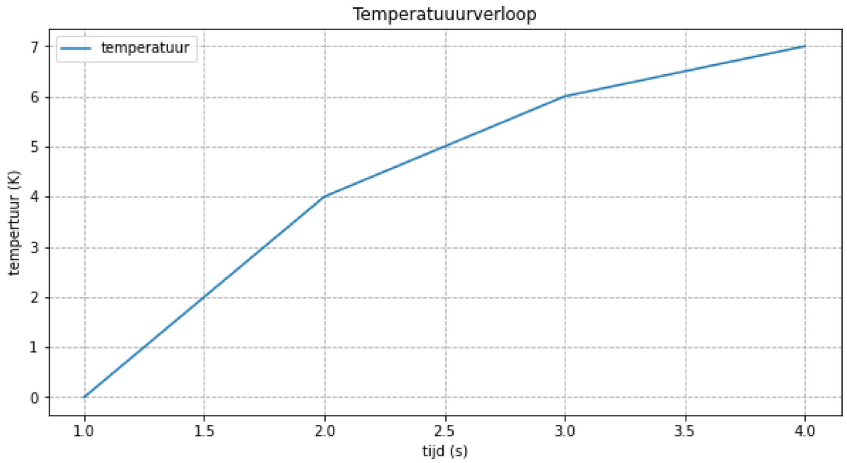
```
ax.legend()
ax.grid()
```

We kunnen de grootte van de figuur ook nog aanpassen, en label voorzien voor de gegevens. Alles samen geeft dit het volgende resultaat:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 5))
seconden = [1, 2, 3, 4]
kelvin = [0, 4, 6, 7]

ax.plot([1, 2, 3, 4], [0, 4, 6, 7], label='temperatuur')
ax.set_xlabel('tijd (s)')
ax.set_ylabel('tempertuur (K)')
ax.set_title('Temperatuuurverloop')
ax.grid(linestyle='--')
_ = ax.legend()
```



Taartdiagrammen

Een taartdiagram is een handig instrument als je de frequenties wil tonen van een beperkt aantal mogelijke waarden, waarbij de onderlinge verhouding niet belangrijk is of wanneer je die obscuur wil houden.

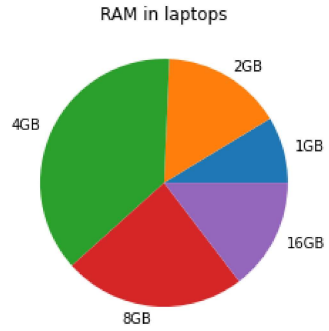
Je kan een taartdiagram gemakkelijk maken in Python. Als voorbeeld zullen we de kolom "RAM" gebruiken (we laten de NaN-waarden hier gewoon weg). Dit geeft volgend resultaat:

```
import pandas as pd

laptops = pd.read_csv('datasets/laptops.csv', sep=';', decimal=',')

x = laptops.RAM.value_counts().sort_index()
l = ['1GB', '2GB', '4GB', '8GB', '16GB']

fig, ax = plt.subplots()
ax.pie(x, labels=l)
_ = ax.set_title('RAM in laptops')
```



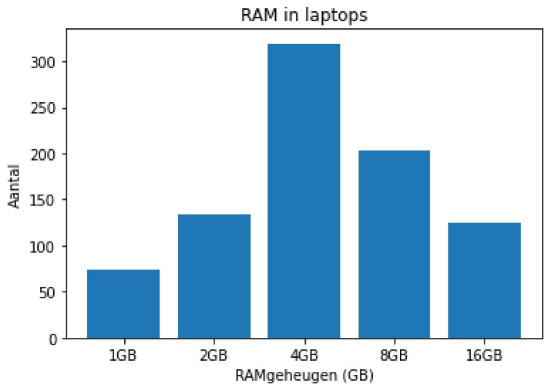
Het is moeilijk te zien of 2 en 16 GB even veel voorkomen of niet. Dat is typisch voor een taartdiagram: de onderlinge verhoudingen zijn niet zo duidelijk. Wanneer het aantal mogelijkheden te groot wordt, is een taartdiagram in ieder geval geen goede keuze meer want dan wordt deze grafiek te onduidelijk.

Staafdiagrammen en histogrammen

Staafdiagrammen laten wel de onderlinge verhoudingen zien. Je ziet hier een voorbeeld (terug voor het RAM-geheugen, zonder de `NaN`-waarden). Dit geeft een overzicht van de absolute frequenties. Het resultaat zie je hier:

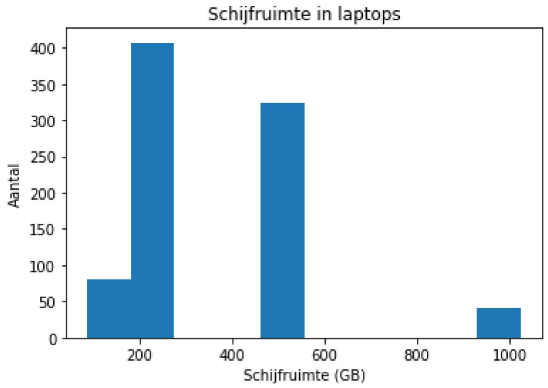
```
x = laptops.RAM.value_counts().sort_index()
l = ['1GB', '2GB', '4GB', '8GB', '16GB']

_, ax = plt.subplots()
ax.bar(l, x)
ax.set_title('RAM in laptops')
ax.set_xlabel('RAMgeheugen (GB)')
_ = ax.set_ylabel('Aantal')
```



Wanneer er klassen gemaakt zijn, kan dezelfde grafiek gebruikt worden. Maar als de variabele (quasi) continu is, dan gebruiken we een variant: het histogram. Bij een histogram worden de staven aan elkaar geplakt. Dit maakt duidelijk dat alle waarden mogelijk zijn en dat de klassen dus niet ver van elkaar staan. Je kan een histogram in Python als volgt maken:

```
ax = laptops.diskspace.plot.hist()
ax.set_title('Schijfruimte in laptops')
ax.set_xlabel('Schijfruimte (GB)')
_ = ax.set_ylabel('Aantal')
```

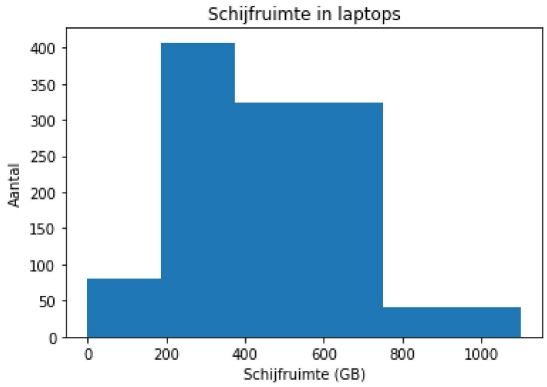


Het verschil met de vorige grafieken is dat deze grafiek gebouwd werd door Pandas zelf. De return-waarde is een Axis-object (`ax`) waarmee je de grafiek dan verder kan gaan configureren. Pandas heeft nog veel meer ingebouwde grafieken. Zoek zelf eens op wat er nog mogelijk is.

Merk verder ook op dat je geen klassen meegeeft aan deze functie (je hoeft `value_counts()` ook niet op te roepen). De functie zal namelijk zelf bepalen hoeveel klassen er nodig zijn, de waarden in klassen opdelen, de frequenties bepalen en dan de grafiek tonen.

Zoals je ziet, zijn er 10 klassen gemaakt. Je kan ook zelf klassen bepalen. Zo zien we in dit voorbeeld dat er 4 grootordes van harde schijven zijn: 120 GB, 250 GB, 500 GB en 1000 GB. We kunnen dus klassen maken voor elk van deze types. We nemen als cutpoints de waarden tussen de mogelijke groottes. Het resultaat is als volgt:

```
cutpoints = [0, 185, 375, 750, 1100]
ax = laptops.diskspace.plot.hist(bins=cutpoints)
ax.set_title('Schijfruimte in laptops')
ax.set_xlabel('Schijfruimte (GB)')
_ = ax.set_ylabel('Aantal')
```

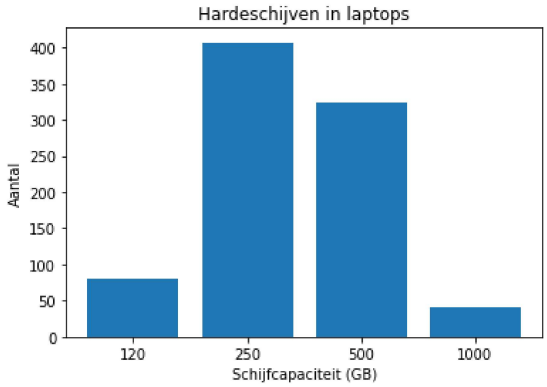


Merk op dat de klassenbreedte hier niet constant is. Dat is hier nuttig omdat de groottes van de harde schijven niet mooi lineair verdeeld zijn. In de meeste gevallen zullen de klassen echter even groot zijn.

Je kan argumenteren dat de laatste grafiek niet helemaal correct is: we hebben nl. juist opgemerkt dat er 4 types harde schijven zijn. We verwachten dus nooit een vrije ruimte van 350 GB. De variabele is dan eigenlijk niet echt continu. Als we dit duidelijk willen maken, moeten we toch een barplot maken. Dit kan als volgt:

```
cutpoints = [0, 185, 375, 750, 1100]
klassen = pd.cut(laptops.diskspace, bins=cutpoints)
l = ['120', '250', '500', '1000']
x = klassen.value_counts().sort_index()

_, ax = plt.subplots()
ax.bar(l, x)
ax.set_title('Hardeschijven in laptops')
ax.set_xlabel('Schijfcapaciteit (GB)')
_ = ax.set_ylabel('Aantal')
```



Andere voorstellingen

De vorige grafieken zijn redelijk klassiek in de statistiek. We kunnen frequenties echter ook met andere grafieken zichtbaar maken.

> **Goed om weten**
>
> Volgende grafieken zijn **louter informatief** en je hoeft ze zelf niet te kunnen maken.
> Je kan er natuurlijk wel veel uitleren.

Spider plots

In de psychologie zijn spider plots (ook wel spindiagram, radardiagram, spider chart of radar chart genoemd) heel populair. Ze zijn handig als je waarden hebt die steeds iets anders meten over hetzelfde onderwerp. Zo kan je bijvoorbeeld nagaan welke eigenschappen er veel voorkomen bij personen die data-analyst zijn.

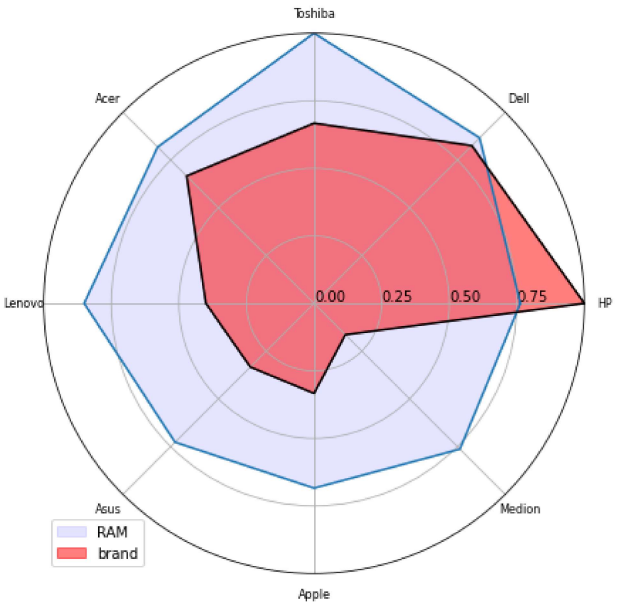
Er zijn verschillende manieren om een spiderplot te maken in Python. We illustreren hier slechts 1 manier. We laten we de frequenties zien van het merk van de laptops.

Het resultaat is als volgt:

```
import math
import matplotlib.pyplot as plt

brand_percentages = laptops.brand.value_counts() / laptops.brand.value_counts().max()
RAM_percentages = laptops.groupby(by='brand')['RAM'].mean() /
laptops.groupby(by='brand')['RAM'].mean().max()
categories = brand_percentages.index
brand_values = brand_percentages.to_list()
diskspace_values = RAM_percentages.to_list()
N = len(categories)
M = max(brand_values)
angles = [n / float(N) * 2 * math.pi for n in range(N)]

_, ax = plt.subplots(figsize=(7, 7), subplot_kw={'projection': 'polar'})
ax.set_rticks(angles)
ax.set_rlabel_position(0)
ax.set_xticks(angles)
ax.set_xticklabels(categories, size=8)
ax.set_yticks([n / 4 * M for n in range(4)])
plt.ylim(0, M)
ax.plot(angles + angles[:1], diskspace_values + diskspace_values[:1], linewidth=1.5,
linestyle='solid')
ax.fill(angles, diskspace_values, color='blue', alpha=0.1, label='RAM')
ax.plot(angles + angles[:1], brand_values + brand_values[:1], color='black',
linewidth=1.5, linestyle='solid')
ax.fill(angles, brand_values, color='red', alpha=0.5, label='brand')
_ = ax.legend()
```



Zoals je merkt, is het maken van een dergelijke plot al wel een pak **moeilijker**. Gelukkig hoef je dit niet zelf te kunnen.

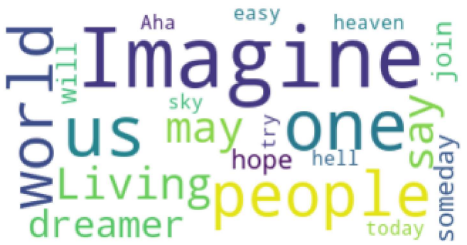
Wordclouds

Wordclouds zijn ook heel populair de afgelopen tijd. Ze laten zien welke woorden er veel voorkomen in een tekst. Hoe meer het woord voorkomt, hoe groter het woord getekend wordt. In Python kan je deze heel gemakkelijk maken. Hier zie je een mogelijk resultaat wanneer je de tekst van “Imagine” van John Lennon als input neemt.

```
import wordcloud as wc

lyrics = open('datasets/imagine.txt', 'r').read()

cloud = wc.WordCloud(max_font_size=70, max_words=20,
background_color='white').generate(lyrics)
_, ax = plt.subplots()
ax.imshow(cloud, interpolation='bilinear')
_ = ax.axis('off')
```



By Wouter Deketelaere, Kris Demuynck, Wim Dekeyser, Jan Van Overveldt
© Copyright 2023.

Toegepaste Informatica - Karel De Grote Hogeschool