20220111_Informationsmanagement_MV1

January 12, 2022

[1]:
```
# Maschinelles Lernen
```

[2]:
```
# Mittelwert, Median und Mode
```

[3]:
```
speed = [99,86,87,88,111,86,103,87,94, 78,77,85,86]
```

[4]:
```
# Mittelwert

(99+86+87+88+111+86+103+87+94+78+77+85+86)/13
```

[4]:
```
89.76923076923077
```

[5]:
```
import numpy as np

x = np.mean(speed)

print(x)
```

```
89.76923076923077
```

[7]:
```
# 77, 78, 85, 86, 86, 86, 87, 87, 88, 94, 99. 103, 111

y = np.median(speed)

print(y)
```

```
87.0
```

[9]:
```
# mode

from scipy import stats

z = stats.mode(speed)

print(z)
```

```
ModeResult(mode=array([86]), count=array([3]))
```

[10]:
```
# Standard Abweichung

# std = sqrt(sum((xi-x)^2)/n-1)

x = np.std(speed)

print(x)
```

```
9.258292301032677
```

[11]:
```
# varianz

# varianz= std^2

x = np.var(speed)

print(x)
```

```
85.71597633136093
```

[12]:
```
# Perzentile. werden in Statistik verwendet um eine Zahl zu geben ,
# die den Wert beschreibt, unter dem ein bestimmter Prozentsatz der Werte␣
 ↪kleiner ist.

# Was ist das 75. Perzentil?

# 77, 78, 85, 86, 86, 86, 87, 87, 88, 94, 99, 103, 111

# 94 lässt 75% der Werte links von ihm und 25% der Werte rechts von ihm.

x = np.percentile(speed, 75)

print(x)
```

```
94.0
```

[13]:
```
x = np.percentile(speed, 30)

print(x)
```

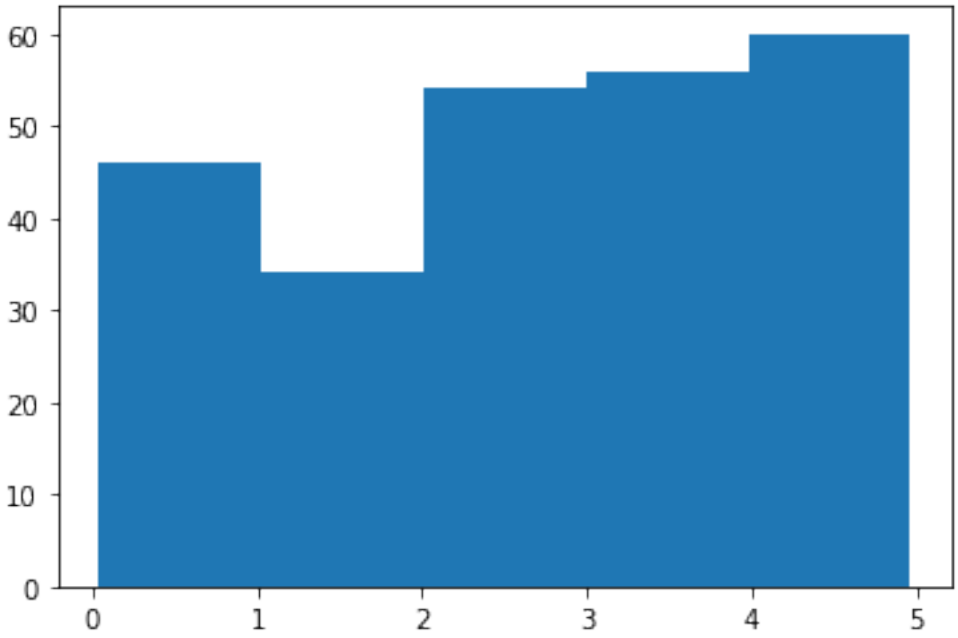
```
86.0
```

[16]:
```
# Wie können wir grosse Datenmengen erhalten?

# Um grosse Datensaetze zum Testen zu erstellen,
# verwenden wir das Python Modul Numpy,
# das eine Reihe von Mehoden enthält, um
# zufällige Datensätze beliebger Grösse zu erstellen.
```

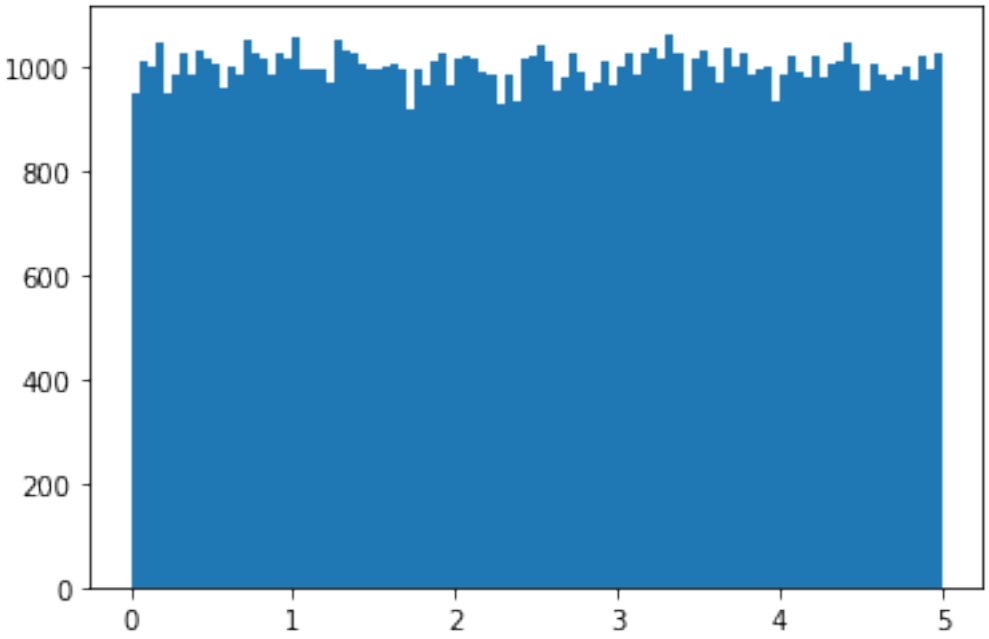
[17]:
```
# Erstellen Sie ein Array mit 250 zufälligen Gleitkommazahlen zwischen 0 und 5.

x = np.random.uniform(0,5,250)
print(x)
```

```
[3.01651274 0.45653488 4.83961723 2.62714375 2.98624017 3.69884981
 0.42710114 4.97370918 4.52038816 4.0724098  3.80226062 3.84920309
 1.64882777 1.12885721 2.48874843 4.43566429 3.99958481 2.06432969
 2.58871276 3.82464844 4.87264015 4.35457434 3.51991779 0.89832725
 2.55153732 0.99026218 4.31817024 2.3868238  1.12419229 3.77494045
 3.59466146 3.52713237 4.33274375 3.99627895 1.53268432 1.93989476
 1.02173973 2.80623228 4.13171082 4.57551617 3.54224333 1.35275923
 4.31641905 1.16042572 3.49815653 0.58476994 1.35258747 4.12000654
 2.26392272 1.56508711 0.01017165 4.59530752 3.97813093 4.40897233
 1.91265114 1.23423436 0.29375747 3.61629334 3.46810808 0.29777942
 1.03816489 1.46383946 4.23197201 3.12304142 2.73477601 3.6866955
 1.95454821 1.01630928 2.90957554 0.06590518 3.86544579 4.14076191
 0.42759243 4.86777864 0.68266213 0.64870113 0.86799045 1.02842583
 3.25269472 1.99220188 2.25222656 3.2008857  0.16509205 4.17475047
 4.10738194 3.83619595 1.61449473 4.62372988 0.01051716 2.4898749
 4.42842936 3.73927221 0.97889414 4.07279763 1.75869611 3.70145054
 1.03868619 0.26219984 2.12530326 3.23922148 4.19301551 3.47600621
 2.45402442 0.87729399 3.16800255 2.42376552 3.15767823 1.14615527
 4.86841648 4.55713246 0.17202507 4.1221104  4.5702988  2.42678412
 1.49795339 3.37650904 4.22082329 0.03349402 3.26623099 1.08120458
 4.94865599 3.43680891 0.84766032 0.36885705 1.89578196 2.3378471
 1.63211338 0.47837809 3.85749251 0.627571   2.33574753 1.56496087
 2.92480864 2.8530869  4.04761943 2.81054852 4.21782823 3.28157688
 1.90188356 4.17940004 4.85602086 2.80518059 3.74409683 0.31230533
 4.24299427 2.71925149 4.51652836 4.12875307 2.89179047 1.64760905
 1.46446737 0.20824682 1.20959272 0.06258261 2.24351923 0.87784525
 0.37626172 0.45767046 3.16993591 0.46941476 0.08486947 4.56807985
 0.76272919 3.1276517  3.70301775 0.54585944 0.09923132 4.59933284
 4.48389954 4.29060282 2.82108854 1.56638115 4.05313508 1.15803425
 1.14870665 0.3167192  1.58176552 0.57777996 3.12386843 2.53206317
 0.57202963 0.7000832  1.08383264 2.64594118 4.71479006 3.79259785
 1.13962779 0.31516722 2.03917618 3.03213221 0.09484256 0.03210493
 4.24324447 2.02535688 3.59782331 0.72262622 3.66255156 4.82544962
 0.52003903 2.86978308 3.67690644 2.41925221 3.73891061 4.85335486
 1.72623424 0.7127496  1.75507192 3.98914053 2.90732964 0.35750617
 0.90854079 4.9631514  3.5789227  0.10825915 3.35606694 3.97419883
 1.8047212  4.82645981 1.77743762 3.74208592 4.69974853 2.17831803
 3.26524894 3.80907893 2.62837345 2.68082069 1.40707013 3.78141461
 1.64122572 4.26235026 2.90414094 1.89204971 3.87311705 3.57467915
 4.70040488 3.15212731 1.75260376 3.51162406 4.36931325 4.32230142
 3.77201401 3.7660622  0.62262346 1.19515941 3.19667349 0.42857686
 4.53098644 3.92637981 1.69441189 0.83639861]
```

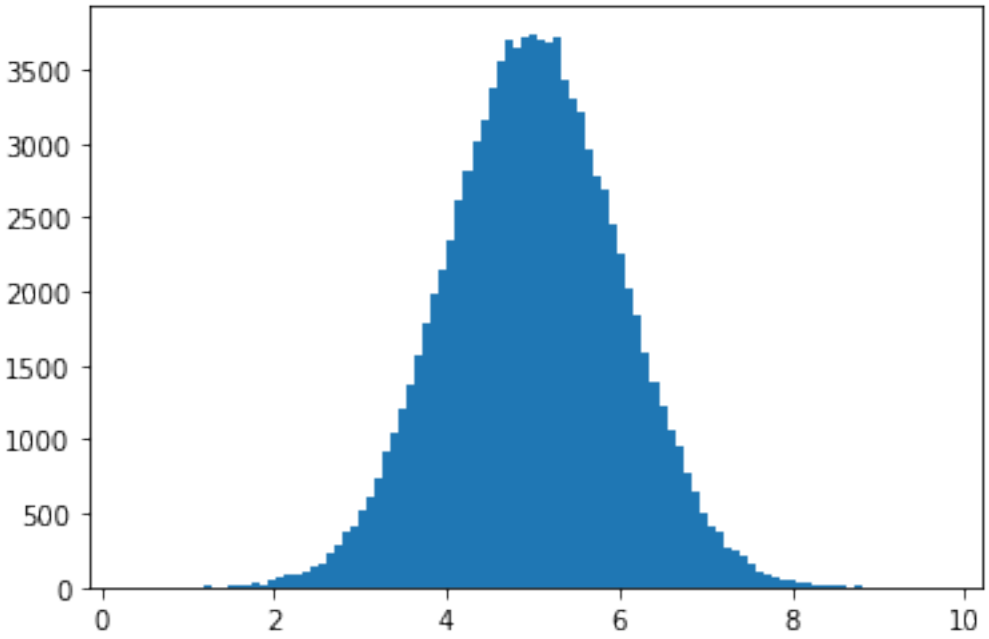
[18]:
```
import matplotlib.pyplot as plt

x = np.random.uniform(0,5,250)

plt.hist(x,5)
plt.show()
```



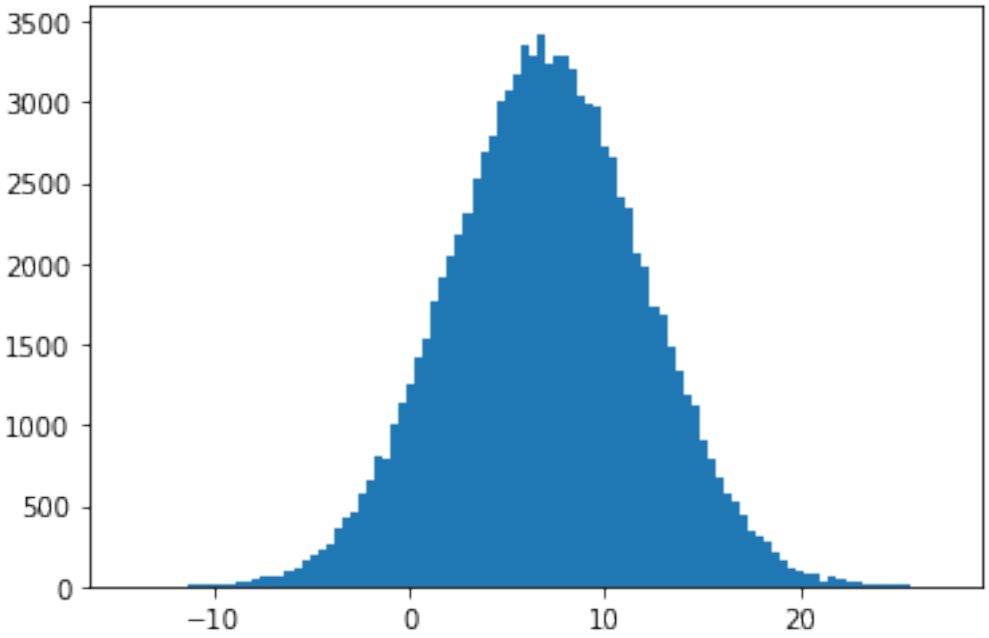
[21]:
```
# Big Data Vertilungen darstellen
# Erstellen Sie ein Array mit 100000 Zufallszahlen und zeigen Sie sie mit einem␣
 ↪Histogramm mit 100 Balken.

import numpy as np
import matplotlib.pyplot as plt

x = np.random.uniform(0,5,100000)

plt.hist(x,100)
plt.show()
```



[22]:
```
# Normale Datenverteilung

# Mittelwert 5, Std Abweichung 1, 100000 Datensätze

import numpy as np
import matplotlib.pyplot as plt

x = np.random.normal(5, 1, 100000)

plt.hist(x,100)
plt.show()
```

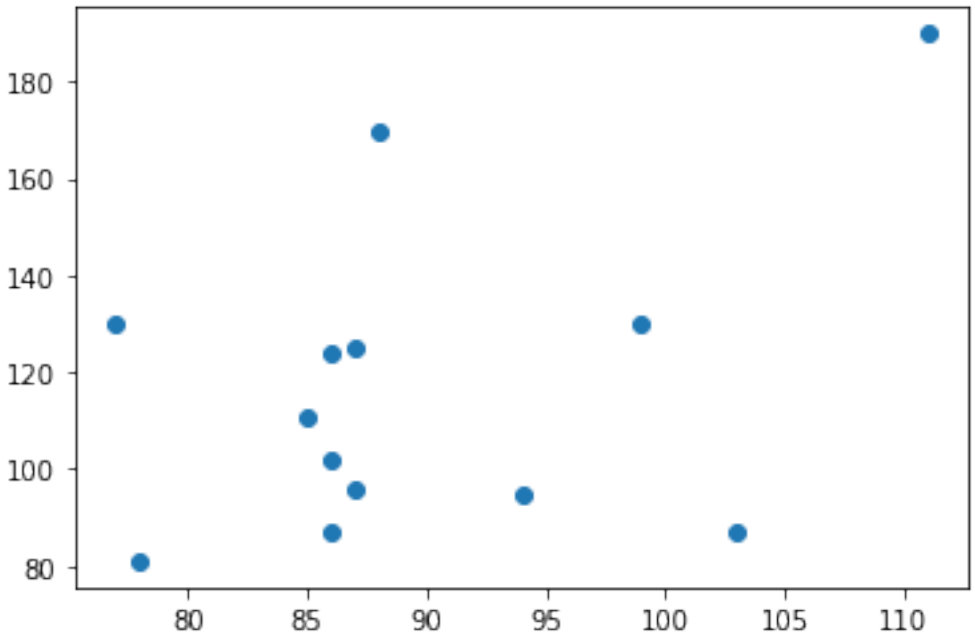


[24]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.normal(7, 5, 100000)

plt.hist(x,100)
plt.show()
```

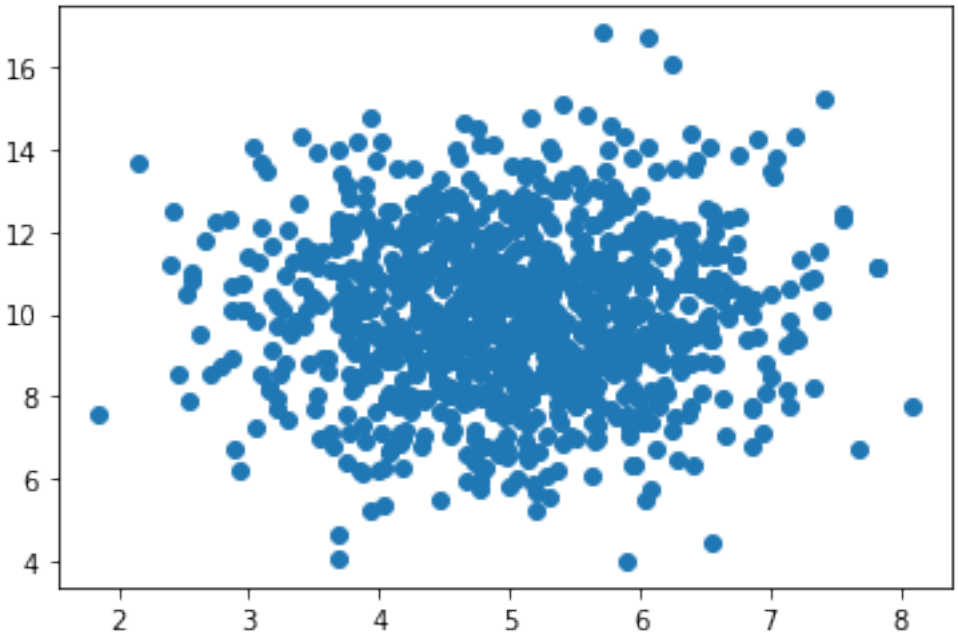


[25]:
```
# Scatter plot

gewicht = [130, 124, 125, 170, 190, 87, 87, 96, 95, 81, 130, 111, 102]
```

[26]:
```
plt.scatter(speed, gewicht)
plt.show()
```

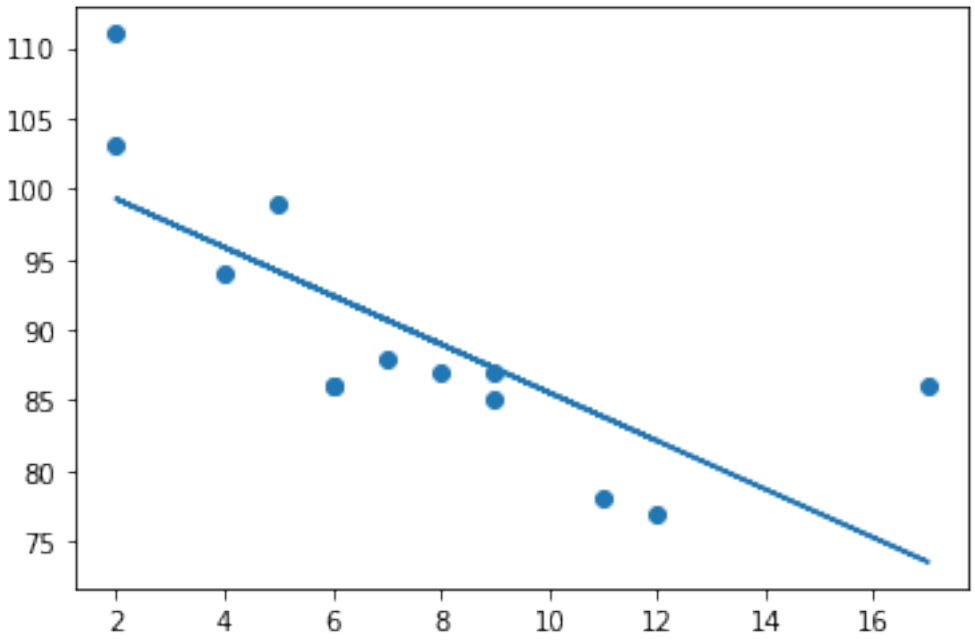


[27]:
```
# Ein Scatterplot mit 1000 Punkten und Normal verteilt

x = np.random.normal(5,1,1000)
y = np.random.normal(10,2,1000)

plt.scatter(x,y)
plt.show()
```



[30]:
```
# Lineare Regression

# Die Lineare Regression verwendet die Beziehung zw. den Datenpunkten,
# um eine gerade Linie durch alle zu siehen.
# Diese Linie kann dann verwendet werden um zukünftige Werte vorherzusagen.

import matplotlib.pyplot as plt
from scipy import stats

# erstellen wir die Arrays, die die Werte der x- und y-Achse darstellen

x = [5,6,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103, 87, 94, 78, 77, 85,86]

# Führen wir eine Methode aus, die einige wichtige Schlüsselwerte der linearen␣
 ↪Regression zurueckgibt:

slope, intercept, r, p, std_err = stats.linregress(x, y)

# Erstellen wir eine Funktion, die die slpe und intercept Werte verwendet um␣
 ↪einen neuen Wert zurueckzugeben.
# Dieser neue Wert stellt dar, wo auf der y-Achse der entsprechende x-Wert␣
 ↪platziert wird:

def myfunc(x):
```

```
    return slope * x + intercept

# Führen wir jeden Wert des x-Arrays durch die Funktion. Dies führt zu einem␣
 ↪neuem Array mit neuen Werten für die y-Achse

mymodel = list(map(myfunc, x))

plt.scatter(x,y)
plt.plot(x,mymodel)
plt.show()
```

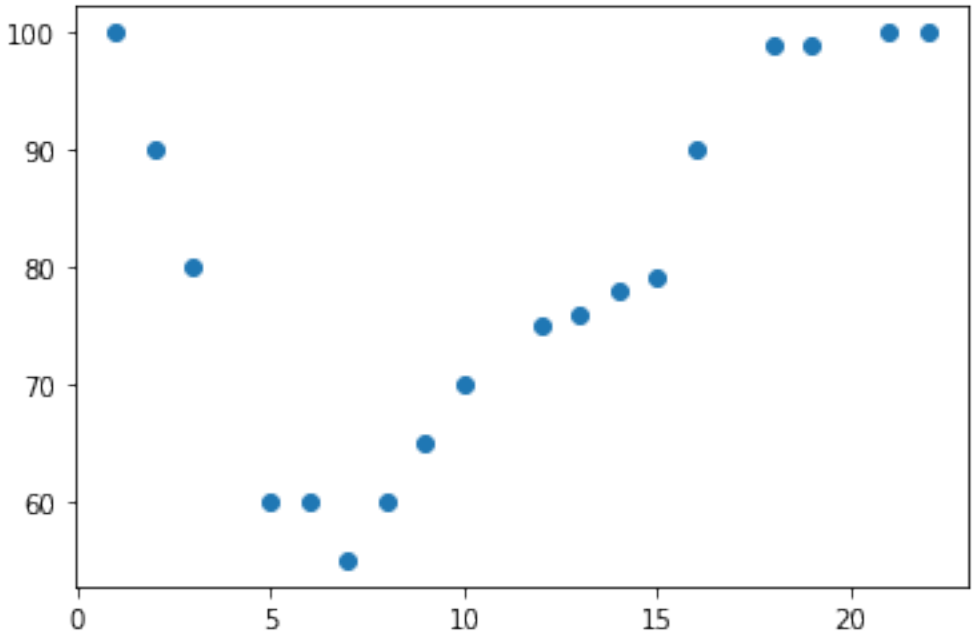


[32]:
```
speed = myfunc(10)

print(speed)
```

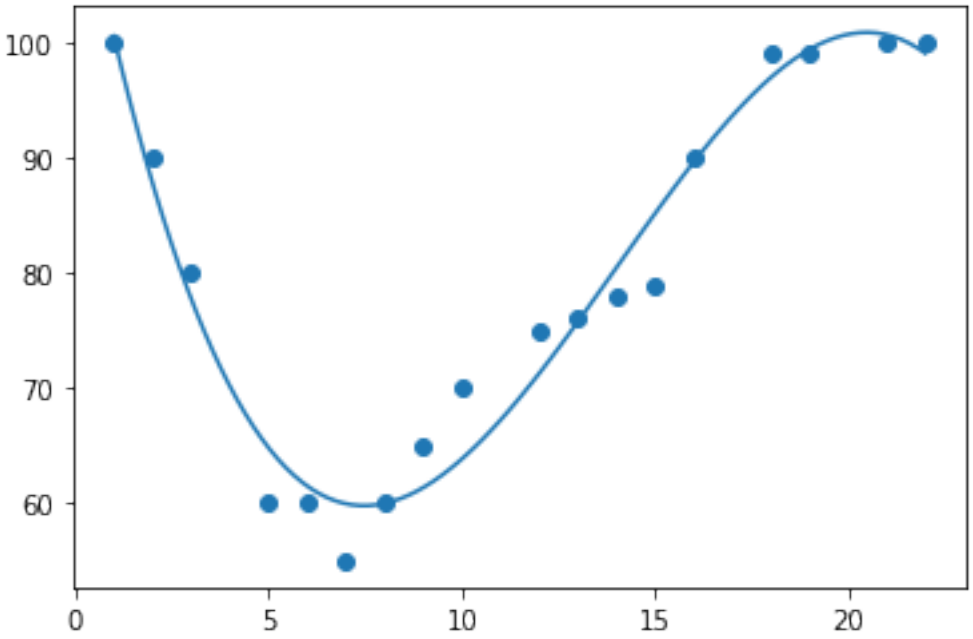
85.54624908958485

[36]:
```
# Polynomiale Regression

# Die polynomiale Regression verwendet
# wie die lineare Regression die Beziehung zw den Variablen x un y,
# um den besten Weg zu finden, um eine Linie durch die Datenpunkte zu ziehen.

x = [1,2,3,5, 6,7, 8, 9, 10, 12, 13, 14, 15, 16, 18, 19, 21, 22]
y = [100, 90, 80, 60, 60, 55, 60, 65, 70, 75, 76, 78, 79 ,90, 99, 99, 100, 100]
```

```
plt.scatter(x,y)
plt.show()
```



[43]:
```
mymodel = np.poly1d(np.polyfit(x,y,3))

myline = np.linspace(1,22,100)

plt.scatter(x,y)
plt.plot(myline, mymodel(myline))
plt.show()
```


100
90
80
70
60
0
5
10
15
20

[ ]: