In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from math import sqrt
from tqdm import tqdm
from timeit import default_timer as timer
```

Intro to ML

Ex0

1. Algebra, Random variables

$\Omega =$ The sample space (the set of outcomes)
$\omega =$ The outcome
$P(\omega) > 0 =$ The probability of an outcome
$\sum_{\omega \in \Omega} P(\omega) = 1$ So sum of probabilities is 1

$X =$ a real-valued random variable (satunnaismuuttuja)

$X(\omega)$ creates a mapping between real numbers and the possible outcomes

a)

An operator L is linear if it obeys:

$$L(x + y) = Lx + Ly$$

where x, y are eg. functions

and

$$L(\lambda x) = \lambda Lx$$

where $\lambda \in \Re$

Show that the operator E is linear:

$E[X(\omega)] = \sum_\omega P(\omega)X(\omega)$

$$E[F(\omega) + G(\omega)] = \sum_{\omega} P(\omega)\,(F(\omega) + G(\omega))$$
$$E[F(\omega) + G(\omega)] = \sum_{\omega} P(\omega)F(\omega) + \sum_{\omega} P(\omega)G(\omega)$$
$$E[F(\omega) + G(\omega)] = E[F(\omega)] + E[G(\omega)]$$

$$E[F + G] = E[F] + E[G]$$

and:

$$E[cF(\omega)] = \sum_{\omega} P(\omega)cF(\omega)$$
$$E[cF(\omega)] = c\sum_{\omega} P(\omega)F(\omega)$$
$$E[cF(\omega)] = cE[F(\omega)]$$

$\mathrm{Var}[X(\omega)] = E[(X(\omega) - E[X(\omega)])^2]$

b)

$$\begin{aligned}
\mathrm{Var}[X] &= E[(X - E[X])^2] \\
\mathrm{Var}[X] &= E[X^2 - 2XE[X] + E[X]^2] \\
\mathrm{Var}[X] &= E[X^2] - E[2XE[X]] + E[E[X]^2] \\
\mathrm{Var}[X] &= E[X^2] - 2E[X]E[X] + E[X]^2 \\
\mathrm{Var}[X] &= E[X^2] - 2E[X]^2 + E[X]^2 \\
\mathrm{Var}[X] &= E[X^2] - E[X]^2
\end{aligned}$$

Here $E[X]$ is the expectation value ie. a number so eg. $E[2XE[X]] = 2E[X]E[X]$ works

2. Bayes Rule

a) Derivation

The probability of event A given B is:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

The probability of event B given A is:

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

Noting that:

$$P(B \cap A) = P(A \cap B)$$

We get:

$$P(A|B)P(B) = P(B|A)P(A)$$

Rearranging gives Bayes:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

b) Medical test

$A =$ allergy
$\neg A =$ no allergy
$T =$ positive test
$\neg T =$ negative test

false positive in 1 % of the healthy:

$$P(T|\neg A) = 0.01$$

true negative in 99 % of the healthy:

$$P(\neg T|\neg A) = 0.99$$

false negative in 10 % of the allergic:

$$P(\neg T|A) = 0.1$$

true positive in 90 % of the allergic:

$$P(T|A) = 0.9$$

15 % of the population in Finland have allergy:

$$P(A) = 0.15$$
$$P(\neg A) = 0.85$$

the probability that a person is allergic, if the test is positive:

$$P(A|T) = \frac{P(A \cap T)}{P(T)}$$
$$P(A|T) = \frac{P(T|A)P(A)}{P(T)}$$

Write:

$$P(T) = P(A \cap T) + P(\neg A \cap T)$$
$$P(T) = P(T|A)P(A) + P(T|\neg A)P(\neg A)$$

Bayes with the marginal:

$$P(A|T) = \frac{P(T|A)P(A)}{P(T|A)P(A) + P(T|\neg A)P(\neg A)}$$

	A	$\neg A$	**SUM**
T		0.01	
$\neg T$	0.1		
SUM	1	1	2

	A	$\neg A$	**SUM**
T	0.9	0.01	0.91
$\neg T$	0.1	0.99	1.09
SUM	1	1	2

Bayes with the marginal:

$$P(A|T) = \frac{0.9 \times 0.15}{0.9 \times 0.15 + 0.01 \times 0.85}$$
$$P(A|T) = 0.94076655$$
$$P(A|T) \approx 0.94$$

3. Matrix calculus

Matrix $\mathrm{A} \in \mathbb{R}^{n \times n}$
Eigenvalue $\lambda \in \mathbb{R}$ of A
Eigenvector $x \in \mathbb{R}^{n}$ if
Eigenfunction $\mathrm{A}x = \lambda x$

A has n orthonormal eigenvectors $x_i \in \mathbb{R}^n$
and corresponding eigenvalue $\lambda_i \in \mathbb{R}$
where $i \in [1, n]$

Orthonormality means:
$x_i^\top x_i = 1$ and
$x_i^\top x_j = 0$ if $i \neq j$

A new matrix:
$\mathrm{B} = \sum_{j=1}^{n} \lambda_j x_j x_j^\top$ "spectral decomposition"

A matrix A can be expressed in the basis of it's eigenvectors $X = [x_1 + x_2 + \ldots + x_n]$ and $X^\top$

$$\mathrm{A} = X \Lambda X^\top$$

where Λ is a diagonal matrix with $\mathrm{Tr}(\Lambda) = \sum_i^n \lambda_i$

$$\mathrm{A} = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ x_1 & x_2 & \ldots & x_n \\ \downarrow & \downarrow & & \downarrow \end{pmatrix} \begin{pmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{pmatrix} \begin{pmatrix} \leftarrow & x_1 & \rightarrow \\ \leftarrow & x_2 & \rightarrow \\ \leftarrow & \vdots & \rightarrow \\ \leftarrow & x_n & \rightarrow \end{pmatrix}$$

Multiplying the matrices yields $\mathrm{A} = \sum_{i=1}^{n} \lambda_i x_i x_i^\top$ Hence $\mathrm{A} = \mathrm{B}$ and the eigenvectors and -values are the same for B as well.

4. Optimization

Constants:
$a \in \mathbb{R}$
$b \in \mathbb{R}$
$c \in \mathbb{R}$

A function:

$$f(x) = ax^4 + bx + c$$
$$f'(x) = 4ax^3 + b$$

a)

The extreme values are found when $f'(x) = 0$

$$4ax^3 + b = 0$$
$$4ax^3 = -b$$
$$x^3 = \frac{-b}{4a}$$
$$x = \sqrt[3]{\frac{-b}{4a}}$$

b)

Conditions: $a > 0$ and none of them can go to the infinity (and beyond)

5. Algorithms

The fibonacci numbers $F(i)$ are defined for $i \in \mathbb{N}$ as:
$F(i+2) = F(i+1) + F(i)$ when $F(1) = F(2) = 1$

> ***I know that no run time experiments are needed but I was curious so here they are anyway. Also those python formulations are neat so I'm going to keep them. Required O-notations are written under the plots and in the function descriptions.***

In [1]:
```
def fibo1(n):
    """Function to calculate values from fibonacci sequence using r
ecursion. Scales O(2^n)"""
    if n<= 1: return n
    else: return fibo1(n-1)+fibo1(n-2)
```

In [2]:
```
def fibo2(n):
    """Function to calculate valuesfrom fibonacci sequence. Scales
O(1)"""
    return int(((1+sqrt(5))**n-(1-sqrt(5))**n)/(2**n*sqrt(5)))
```

In [14]:
```
length = 50
```

In [15]:
```
idx = np.arange(length)
```

In [32]:
```
fibo1_df = pd.DataFrame(index=idx,columns=["Time","F"])
fibo1_list = []
fibo1_times = []
```

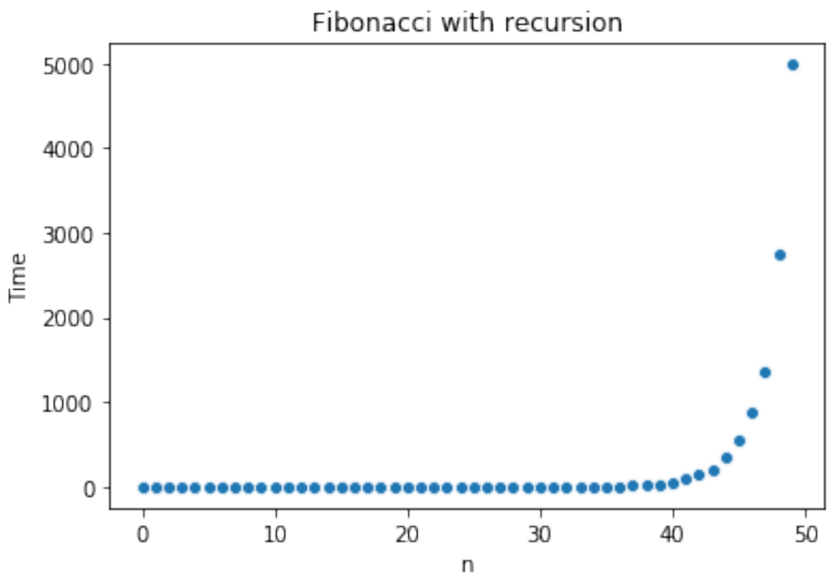
In [33]:
```
for n in tqdm(range(length)):
    start = timer()
    fibo1_list.append(fibo1(n))
    end = timer()
    time = end - start
    fibo1_times.append(time)
```

```
100%|██████████| 50/50 [3:11:21<00:00, 2436.57s/it]
```

In [34]:
```
fibo1_df["F"]=fibo1_list
fibo1_df["Time"]=fibo1_times
```

In [44]:
```
sns.scatterplot(x=idx,y=fibo1_df.Time)
plt.title("Fibonacci with recursion")
plt.xlabel("n")
plt.ylabel("Time")
```

Out[44]: `Text(0, 0.5, 'Time')`



So it almost doubles the time every time: Scales $O(2^n)$
Well the big-O is the worst case scenario here (not like one usually thinks)

In [23]:
```
fibo2_df = pd.DataFrame(index=idx,columns=["Time","F"])
fibo2_list = []
fibo2_times = []
```

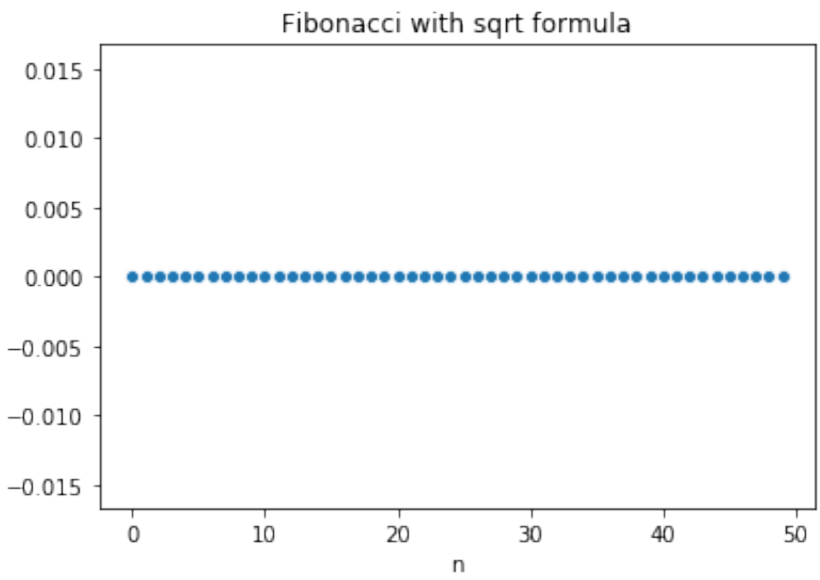
In [24]:
```
for n in tqdm(range(length)):
    start = timer()
    fibo2_list.append(fibo2(n))
    end = timer()
    time = end - start
    fibo2_times.append(time)
```

```
100%|██████████| 50/50 [00:00<00:00, 9364.79it/s]
```

In [25]:
```
fibo2_df["F"]=fibo2_list
fibo2_df["Time"]=fibo2_times
```

In [27]:
```
sns.scatterplot(x=idx, y=fibo2_df.Time)
plt.title("Fibonacci with sqrt formula")
plt.xlabel("n")
plt.ylabel("Time")
```

Out[27]:
```
Text(0, 0.5, '')
```



This follows basically $O(1)$ scaling

6. Data analysis

In [2]:
```
x_data = pd.read_csv("x.csv")
x_data.sample(4)
```

Out[2]:

	V1	V2	V3	V4	V5	V6	V7	V8	
53	-4.941701	-4.229189	7.417093	-2.490971	5.551929	0.107738	8.076430	-8.317607	1
809	-3.022010	-2.411519	7.275234	-1.656121	4.202106	1.116153	7.700970	-6.539154	1
981	-3.573994	-3.071186	6.382967	-1.323073	7.302999	1.176597	8.920917	-6.716963	1
892	-5.807970	-5.219509	6.553538	-3.051890	6.672151	-0.283036	9.336143	-9.051607	1

4 rows × 32 columns

In [3]:
```
x_data.shape
```

Out[3]: (1000, 32)

In [4]:
```
x_data.describe()
```

Out[4]:

	V1	V2	V3	V4	V5	V6	
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	100
mean	-5.493603	-4.848361	6.748369	-3.592207	5.432545	-1.009983	
std	1.860624	1.868029	1.002521	1.798006	1.561889	1.801655	
min	-12.051620	-11.449646	3.402454	-9.883042	0.650691	-7.255290	
25%	-6.720468	-6.091409	6.098733	-4.771207	4.390488	-2.164160	
50%	-5.483653	-4.871369	6.761059	-3.571100	5.406136	-0.987092	
75%	-4.240198	-3.555943	7.438199	-2.368798	6.432138	0.173440	
max	0.765007	1.443165	10.269498	1.833573	10.671700	4.451196	1

8 rows × 32 columns

Here I do what is asked: obtain the data indices without referring to them...

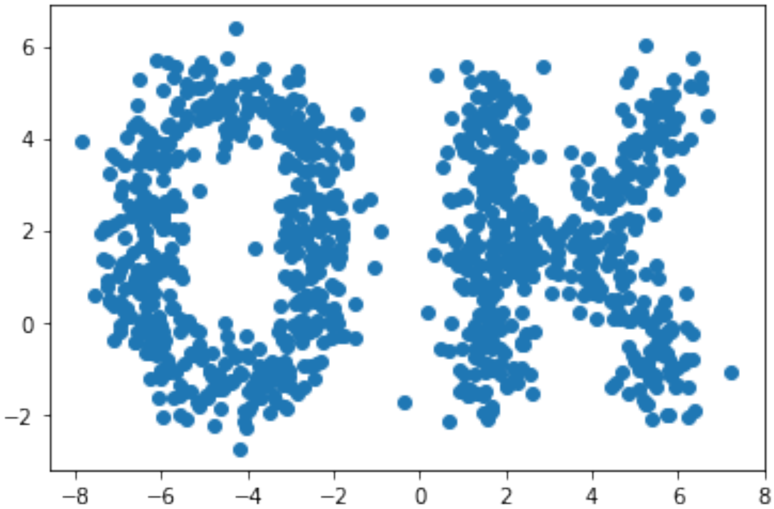
In [5]:
```
indices = np.where(x_data.describe().loc["std",:] > 2)[0]
```

In [6]:
```
use = x_data.iloc[:,indices]
```

And here they are just for plotting but okay I use some other syntax if that pleases..

In [7]:
```
plt.scatter(use.iloc[:,0],use.iloc[:,1])
```

Out[7]: <matplotlib.collections.PathCollection at 0x1a176f4668>



This should be ok 🧐

In [ ]: