np.zeros(shape, dtype)
np.ones()
np.full(shape, fill value)
np.arange(0, 20, 2)
np.linspace(0, 1, 5)
np.random.random((3,3))
- Between 0 and 1
np.random.randint(low, high, (3,4))
np.random.randint(10, size=(3,4))
.ndim . shape . size

a	0	1	2	3	← 2e
0	1	2	3	4	
1	5	6	7	8	
2	9	10	11	12	
3	13	14	15	16	

a[:, 3] = 4, 8, 12, 16
a[:-1, 1] = 2, 6, 10
a[0, [1,2]] = 2, 3
a[:3, ::2] = [1,3], [5,7], [9,11]
Re = a[::-1, ::-1]
a[0] = 1,2,3,4
a[:,0] = 1,5,9,13
a[::2] = [1,2,3,4], [9,10,11,12]

a[0,0] = ...
np.arange(1,10).reshape((3,3))
x[:, np.newaxis] : vieeve.
np.concatenate ([x,y])
- axis = 1,0, None
np.vstack / np.hstack
np.split(x, [...])
np.hsplit / np.hsplit
np.abs(absolute)
np.negative()
np.add.reduce(x)
np.add.accumulate
- tussenstapjes
np.outer
np.min(x, axis=...) of x.min()
X.mean(0).(1).()
np.count_nonzero(x < 6)
- count true
np.mean(x < 6, axis=1)
np.any(x > 8) np.all(x > 8)
np.sum(~((x <= 6) | (x >= 7)))
x[x < 5]

a[::2] = [1,2,3,4], [9,10,11,12] d= np.arange(x.shape[0]), x[d,d]

np.add.at(x, i(indexen), 1)
np.sort(x, axis=0)
np.argsort
np.partition(x, 3(split))
x[x%2 == 1] = -1 of
x.where(x%2 == 1, -1, x)
np.intersect1d
np.setdiff1d
np.where(a == b)
x = x[~np.isnan(x)]
A[np.isnan(A)] = 0
x.reshape((x.size,))

%paste - copy from site, handle multi-line input
%cpaste - more code
%timeit - test time, multiple runs
%a(ls) magic
pwd - print working directory
ls - list work. directory
cd - change directo.
mv - move file
use: ! ?
%pwm - in context limits

%time - time execution single state
%timeit - time repeated execution of
%prun - run code with profiler
%lprun - run ... line-by-line ...
%memit - measure memory use of single stat..
%mprun → run code with l-b-l mem profiler

Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
accuracy = (TP + TN) / (TP + TN + FP + FN)
p=0.1 → log 10 meaning
2 (P · 0.9) / ((P · 0.9) + ((1 − P) · 0.1))
2 (1 − P · 0.9) / ((P · 0.1) + ((1 − P) · 0.9))

np.nansum(), np.nanmin()

pd.read_csv("....csv", index_col="names")
df.sort_values("names") OR df.sort_index() ?? inplace=True
df.isnull() → True/False → serie data[data.notnull()]
df = df[df.Parti.notna()]
df.dropna() / df.fillna() → axis = , how / thresh
df.dropna(axis='rows', thresh=3)
df.fillna(method='(b)(f)ill', axis=...)
df["new"] = df[["...","..."]].mean() OR df["..."] + df["..."]
df[["...","..."]] > 9).sum()
df = df.sort_values(["...","..."], ascending=[False, True])
df = df.loc[df.Parti == "pvdd", "ministerie"]...
zelfde → df[df.Parti == "pvdd"].ministerie
df = df.drop([0], axis=1)
for c in [...]
df[c] = df[c].str... / df.columns
df["..."].value_counts()[:10].index ~~to list~~ tolist()
df[df....isin(^)]
df.groupby(["...","..."])
wat je zum zum doen
["..."].sum()
loc → value iloc → position
Serie ~~to~~ ander index → A.add(B, fill_value=0)
df = df.iloc[0] → 1e row = 0
df.subtract(df.R, axis=0)
.unstack 1e 16 26 → 1e 16 26
2e 16 26 → 2e

df.set_index("...") title=...
for i in range(L.shape[0]):
L[:,0][np.isnan(L[:,i])] = np.nanmedian(L[:,i])

pd.DataFrame(rng.randint(0,20,(2,2)), columns=list('AB'))
fill = A.stack().mean()
A.add(B, fill_value=fill)
pd.concat([df1, df2]) ander index
, axis=1/0 , ignore_index=True
, join='inner' → colum in beide voorkomen
, join_axes=[df1.columns]
df1.append(df2) columns / index hetzelfde
pd.merge(df1, df2) → colum zelfde naam en waard
pd.merge(..., on='...', left_on=..., right_on...
(zelfde waardes onder elkaar)
bij index → df1a.join(df2a)
pd.merge(df6, df7, how='outer' 'left' 'right'
...plot(kind='bar') 'hbar'

- kb → hoe lange in bytes

NP .aRRay(List of TuPles)

- ZOROS((3,4)) = vol 0
- ones " 1
- aRRange(start, stop, step) = openvolgende
- Linspace ")
- Full((shape), Int) volle aRRay
- Random. Random (SHape)
- shape
- ndim = dimensies
- info( ) = Help

axis 0 = Row's
" 1 = Colomn's

a. sort()
- ReshHape (SHape)
- concatenate((a,b) axis=0)
- vstack
- Hstack
- Random. Randint (start, stop (SHAP))

0 1 2 3 4 5 6 7 8 9

0 △ ↓
1 ↓
2 ↓
3 ↓
4 o o o o o o o o o o

s[0,1] = △
s[:,3] = ↓
s[1,:] = 0

Precision = TP / TP+FP

Recall = TP / TP+FN

ACC = TP+TN / P+N

F1 = 2TP / 2TP+FP+FN

Pandas pd.

DataFRame
- Melt (df) colums → Rows
- concat = append (axis 0,1)
- Pivot (colomn='kol', values='val')
- SORT-Values ('') ascending = False
- Renome
- SORt_index
- dROP
- FiCteR (REgex) →
- Loc[:, x2:x4 colums van x2 tot x4
- iLoc[3, [1,2,5]] 1,2,5 de colums
- values_count()
- nunique()
- desCRibe
- GRouPby(by='col')

- Pivot_table aggfunc = Bereide afg {'count', sum}
- CROSStab(col, col)
- index[0]

\. alles met .
LengtH$ eindigt met 'LengtH'
^Sepal begint met 'Sepal'
^x [1-5]$ " x, eindigt met 1,2,3,4,5
^(?!.species$).* alles behalve met species

IPython magic:
%ls: directory read
% magic: alle magic commands

List comprehension:
new_list = [i+1 for i in old_list if i < 5)
new_list = [i for i in range(10)]

Numpy:
X[pos]: get position in 1D array.
X[pos,pos]: get position in 2D array.
x.reshape((row, column)): zo reshape je een numpy array.
np.array([1,2,3]): maak een list en daarvan een numpy array.
X - 1: voer berekening uit op elk item in X.
x.reshape((getal,)) maak van een 2d array een 1d array.
X + 1 == 2: apply boolean mask on array.

Pandas:
pd.read_csv('csv'): een csv bestand inladen.
dF.column.value_counts(): laat de values van de verschillende elements zien.
dF.drop('column/row', axis = 0/1): verwijder een column of row van df.
~~dF.sort_values(['column1'])~~
dF.sort_values(['column1', 'column2'], ascending = [True, False])
dF.groupby('column')['Anothercolumn'].mean()
indexno = dF[(dF['column1'] <= 10) & (dF['column2'] >= 5)]
dF.drop(indexno, inplace = True): drop meerdere dingen tegelijk
x = Counter(z.split()): Tokenize text
xy = pd.Series(X).sort_values(): maak een pd series en sorteer de values.
XY[XY == 1]: geef alle unieke woorden in een series.
(XY == 1).mean(): geef gemiddelde van alle woorden met value 1.
dF.describe(): laat de values van dF zien.
dF.index.str.contains('x').head(10)
(dF_2 - dF.mean()) / dF.std(): z - normalize

Formules:

Precision: TP/(TP+FP)

Recall: TP/(TP+FN)

accuracy: (TP+TN)/(P+N)

Numpy P = array

Diagnal of array:

d = np.arrange(P.reshape[0])

P[d,d]

Slicing array:

P[Rij, kolom]

: alles, ijfer = grens (tot)

Array Platten

D.flatten()

~~np.sort(axis=0)~~

np.sort(P, axis=)

0 = kolom, 1 = Rij

np.random.randint

(1,10,(5,10))

(proberen)

np.T = transpose

From numpy import doc

?np.doc

np.lookfor

df.join

df.groupby(by='col')

df.merge

Axis=0: vertical/row/↓ Axis=0: horizontal/column/→

np.array(rij/kolom), shape(0,1)

Numpy

maken
np.zeros
np.arange
np.Linspace
np.full
np.random.random

inspecteren
a.shape
a.ndim
a.astype
a.isnan
a.nanmedian

math
np.median
np.mean
np.mode
np.sqrt

manipuleren
a.flatten
a.transpose
a.reshape
a.resize
a.column_stack
a.unstack
a.hsplit
a.vsplit

overig
a.sort
a.idxmax
a.diagonal
np.nonzero(L)

Boolean indexing
bad_acc = ~~acc[cond]~~ cond
newacc = ~~bla~~ acc[boolacc]

sorting
sort() → axis=1 per rij
axis=0 per kolom

broadcasting
a = [[1],[2],[3]]
b = [1,2,3]
a+b

Pandas

select
df.loc → kolom
df.iloc → index
df.idmin/idmax
df.ix
df.where
df.item
df.columns(name)

manipuleren
df.drop
df.dropna
df.fillna
df.apply
df.pivot_table
df.corr()
df.reset_index
df.unstack
df.set_index

plotten
df.plot()
df.bar
df.hist
df.scatter
df.T (transpose)

sort
df.sort_values([val1, val2], asc=[True, False])
df.sort_index

overig
df.unique
df.nunique
df.value_counts
df.group_by
random.randint
→ ie

logic operators
(^=xor)
| = or
& = and
~ = not

regex
re.match
re.sub
re.findall
\: escape/start
. : any except newline
^ : start
$: end
[] = set
| = or
( ) : group
insert:
- range
^ negate
\A : start
\Z : end
\B : empty
\d\D : (non) digit
\w\W : (non) alphanum
\s : whitespace

formules
Z-norm = x-mean/std

$P = \frac{TP}{TP+FP}$

$R = \frac{TP}{TP+FN}$

$A = \frac{TP+TN}{All}$

$F_1 = 2 \cdot \frac{P \cdot R}{P+R}$

		Actual P	Actual N
p	P	TP	FP
	N	FN	TN

ipython
?
* = wildcard
%time
%prun
%ls -lh
%cd
%pwd

Numpy

```
np.arange(start, stop, step)
np.linspace(start, stop, steps)
np.zeroes((3,3)) / np.ones((shape))
np.full((shape), value)
np.random.random((shape))
a.shape / ndim / size / dtype
a.astype(type) - convert to type
```

Indexing / Slicing

```
a[row, column]
b[0:2, 1] - rows 0&1 of column 1
a[a<2] - boolean Indexing
fancy indexing: b[[1,0,1,0],[0,1,2,0]]
pakt: (1,0)(0,1)(1,2)(0,0)
```

```
np.transpose(a,(1,0,2)) - kies dimensions
a.ravel() / a.flatten() - maak 1D
a.reshape(shape) / a.resize(shape)
np.append(a, value) / np.delete(a, [1])
np.insert(a, value, place)
np.hstack(a,b) / np.vstack(a,b)
np.repeat(a,3) → [1,1,1,2,2,2,3,3,3]
np.tile(a,3) → [1,2,3,1,2,3,1,2,3]
np.vsplit(a,3) / np.hsplit(a,3)
```

```
def middle_square(L):
    n = len(L)
    begin = int(n/2 - 2)
    eind = int(n/2 + 2)
    return L[begin:eind, begin:eind]
```

- Elements in 2 arrays match:
  `np.where(a==b)`
- Get all numbers between 5 and 10
  `np.where((a>=5) & (a<=10))`
  `a[(a>=5) & (a<=10)]`

- `np.sqrt(((X1 - X1.mean())**2).mean()) == np.std()`
- `L + np.arange(L.shape[1])` - +0 +1 +2 etc.
- `tvr.Party = tvr.Party.str.lower().str.replace(words,'')`
  `words = re.compile("woord|nog")`

```
greater than_9 = (df[['col1','col2','col3',...]].mean().sum()/sum)
ser1[~ser1.isin(ser2)]  np.percentile(1,q=
```

Pandas

```
S.value_counts(dropna=) values + count of Series
df[col] - return column of df as Series
df.columns ['a','b','c'] - rename colums
df[[col1],[col2]] - return columns as new df
pd.isnull() / pd.notnull() - boolean array
df.dropna() / df.fillna()
df.set_index('column_name')
df[(df[col]>0,5) & (df[col]<0,8)]
df.sort_values(col_name, ascending=)
df.groupby(col) / df.groupby([col1, col2])
df.groupby(col1)[col2].mean()   osc=[True, False]
df1.append(df2) - add rows of df1 to end of df2
pd.concat([df1, df2], axis=1) add col of 1 to 2
df.pivot_table(index=col1, values=[col2, col3])
pivot = df.pivot_table(values="Count", index='Name', columns='Sex', aggfunc=Sum)
iris.species.value_counts().plot(kind='bar')
X = iris.drop('species', axis=1)
iris.sort_values(['sepal_length','sepal_width'],
    ascending=[True, False]
iris.groupby('species')['sepal_length'].max()
iris[(iris['sepal_length']>5) & (iris['sepal_width']
    < sqrt(5))]
```

```
txt = text.split()
c = Counter(txt)
CS = pd.Series(c).sort_values(ascending=False)

CS[CS*CS.index.str.len()==24]
CS[CS==1].sum() / CS.count() unique
(CS==1).sum() / CS.sum() of all words
```

```
names = pd.read_csv('babynames.csv')
last_char = names.copy()
last_char['Name'] = last_char.Name.str[-1:]
mannen = last_char[last_char.Sex='M']
vrouwen = " " " " " ='F']
mannen_sorted = mannen[['Name','Count']]
    .groupby('Name').sum()
vrouwen_sorted  idem
mannen_sorted.columns = ['mannen']
vrouwen_ " "  = ['vrouwen']
combined = mannen_sorted
combined.join(vrouwen_sorted)
```

Precisie: TP / (TP + FP) - (0,9·0,01)/(0,9·0,01 + 0,1·0,99)

```
%ls          %time   %timeit
%matplotlib  %who_ls
%rm
```

from numpy import doc
?np.doc

?df.x
np.lookfor('')

finAce
financial integrators

Slicing

np.random.randint(start, stop, (rows, colums))
x[row, column]

Array manipulation:

x.flatten() / x.ravel()
x.reshape(row, column)
np.append(x, i)
np.insert(x, 1, 5)
np.delete(x, [1])
np.concatenate(x, i)
np.vstack((a,b))
np.hstack((a,b))
np.vsplit
np.hsplit

---

x.shape
x.astype(int)

---

np.zeros((3,4))
np.ones('')

Pandas

df.sort_values()
df.rename()
df.sort_index()
df.reset_index()
df.drop()

---

df.plot(kind = bar/hist)

df['w'].value_counts
df[df.Length > 7]

df['column1', 'colum2', 'colum3']

pd.series() ← make pandas object of something

TP = kans • aantal
FP = kansfout • aantal
Precision = TP / (TP - FP)
Recall = TP / (TP - FN)

		predicted: negative	predicted: positive
actual	negative	TN	FP
	positive	FN	TP

IIII

Bezoekadres: Laan van de Helende Meesters 13, 1186 AC Amstelveen
Telefoon +31 20 3113800, Fax +31 20 3113809
Postadres: Postbus 102, 1180 AC Amstelveen
E-mail service@finace.nl, Website www.finace.nl

import

```
import bz2
import codecs
import os
import random
import re
from collections import Counter, defaultdict
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sn
from bs4 import BeautifulSoup
from nose.tools import assert_almost_equal, assert_equal
```

```
from tqdm import tqdm_notebook
% matplotlib inline
pandas as pd
```

* irrelevante chars verwijderen

```
for i in range(len(page_list)):
    for j in range(len(page_list[i])):
        cleanregex = re.compile('<.*?>')
```

np.nan median
np a.flatten()

1

```
np.random.randint(n, size=n5mn), random.choice([j for j in range(n) if j != queens[i] and j not in diagonals[i]] for i in range(len(queens)))
plot, plt.xlabel('hoi'), plt.legend(), plt.plot(x, list, label='hoi'),
with bz2.open(leesfile(), encoding='utf8', mode='rt') as source_file:, source_file.readlines, if '</page>' in line, *
```

2

```
luktopdf: if 'file.txt' in os.listdir() return 'file', elif os.path.exists('../../data/week2/'):
return file
```

```
np.arange(0, n**0,0), np.array([1,2,3]), np.([1,2,3], [4,5,6]]), a.shape = (3,3) of (2,3)
```

axis=0 ↓↓
axis=1 →

```
a.size = 3, a[0,:] := all -> 1,2,3, a[1,3] -> 6, a[1,-2] -> 5, [0:1:6:2] -> 2,4,6
veranderen = a[0,2] = [1,2], np.min(a), np.max(a), np.max(a, axis=1),
np.zeros(5) -> [0,0,0,0,0] np.zeros([2,3]) [[0,0,0],[0,0,0]]), np.ones, np.full((2,2), 99)
                                                                  [[99,99],[99,99]])
np.random.Rand(4,2) -> [[0.81, 0.43],[0.69, 0.61], ",",]
array.ravel, np.sum(a), a.reshape((3,1)) -> [[1],[2],[3]]
np.sum(a, axis=0)        np.arange(P.shape[0])
np.vstack(a,b), np.h.tack(a,b), np.any(a>50, axis=0), np.all(a>50, axis=0)
a>50, a[a>50] -> [60,92,80]), ((a>50)&(a<100)), (~((a>50)&(a<100)))
n = int(L.size **0,5)          false, false, true      true, false, false
np.reshape(L,(n,n)), L[L%n==0], np.mean(L, axis=0), L.shape[1], a.T
```

3

```
pd.read_csv(luktofile(), index_col=0, sep='\t', names=['jaar'], skip
pd.crosstab(kvk['jaar'], kvk['partij']), pd.a.filter
pd.dropna(subset=['Partij'], inplace=True
str.lower().str.replace("\W","")
str.replace("|".join(woordenlijst),"")]
kvk[~kvk['Partij'].str.contains("vragen")
.unique
kvk[kvk>4]
value_counts().index[0], values
plot(kind="bar")
str.count('\?')
str.len()
.mean(), .median(), .mode()
corr = cor(df.length, method='pearson')
```

```
pivot_table(d, "count", index="name
", columns="Sex", aggfunc=sum,
margins=True
```

(df - df.mean())/df.std()

	pred. yes	no
actual yes	TP	FN
no	FP	TN

prec = TP:(TP+FP), Rec = TP:(TP+FN)
acc = (TP+TN):(TP+TN+FP+FN)

.sort_index(), sort_values(by='all'
ascending=True

Spiekbriefje Jop Rijksbaron 11bss1u

np.array
np.arrange
np.count_nonzero
np.concatenate([arr1, arr2])
np.save np.reshape([1,(x,y))
np.suit
np.ravel(arr)
[:, np.newaxis]
[row:column]

arr.transpose
arr.reshape
arr.flatten
arr.shape[0]

arr.size

np.add
np.subtract
np.negative
np.multiply
np.divide
np.floor_divide
np.power
np.abs
np.sqrt

np.sum → nansum
Prod
Mean
Min
Max
std
var
argmin
argmax
median
Percentile
any
all
~~abs~~

nancumsum np.cumsum(axis)

[[[!= 0]

Pandas

data[1:3] implicit data.loc[1:3] explicit data.iloc[1:3] explicit index

Df.
- add
- subtract
- multiply
- divide
- floordiv
- mod
- pow

df.dropna, fillna()
df.drop
pd.concat
pd.merge
df.append

pd.crosstab
pd.
df.mean
df.mode
df.median
df.std

.T
.stack
unstack

kolom = C.loc[Rij].idxmax()

vraag10 = {rij, kolom}

rij = df.max().idmax()

df.str (lower / upper / contains / len / strip / replace
df.sort_values([columns]).ascending([true, false])
df.value_counts().index.to_list()[0]
groupby.apply.unstack

Resc = ((df['assignment.mean'] < 5,5) | (df[ ] < 5,5)).mean * 100

for i in range(L.shape[0])

L[:, i][np.isnan(L[:, i]) = np.nanmedian(L[:, i])

Regex

<.*?>

[^\w\s]

\w letter
\s whitespace
\n newline
\t tab
$ end of string
^ start
\d decimal

P · accuracy (1-P)·(1-acc)

	TP	FN
P·(1-acc)	FP	TN (1-p)·acc

?

accuracy = (TP+TN) / (TP+FP+FN+TN)

Precision = TP / (TP+FN)

recall = TP / (TP+FN)

$F_1 = \frac{2 \cdot recall \cdot Precision}{recall + Precision}$

precision = TP/(TP+FP) recall = TP / (TP+FN) Acc = nT

np.into(np.array.dtype) → b.astype(int)

np.exp(b), np.add(a,b), np.divide(a,b)

b.cumsum(axis=1), a.corrcoef() np.hsplit(a,3)

np.copy(a), = a.copy(), = a.view()

a.sort(), a.sort(axis=0), np.concatenate((a,d),axis=0)

np.flatten() == b.ravel(), .resize(). append, insert, delete

np.zeros, .ones, .arange(10,25,5), .full(), .random.random(), .empty

diagonal(s): d = np.arange(s.shape[0]), return s[d,d]

np.isnan(L[:,i]), np.nanmedian(L[:,i]), np.nonzero()

np.arange(1,4)[:, np.newaxis], np.sort() axis=0

np.all(), np.any(), np.absolute == np.abs, IPython: help(len)

pd.melt(df), pd.concat([df1, df2], axis=1) data[-1] = laatste

df.sort_index(), df.reset_index(), df.drop(colums=[a, b])

regex: (\), 'Length$' = eindigt op length, '^Sepal' = begint met Sepal

df.drop_duplicates(), df.sample(), df.nlargest(n, 'value')

df.loc[:, a:b], df.iloc[:, [1,2,5]], df['w'].value_counts()

df['w'].nunique(), df.describe(), df.iat(), dropna / fillna()

pd.merge(df1, df2, how="left/right/inner/outer", on="x1")

df1[~df1.x1.isin(df2.x1)] pd.MultiIndex()

(wel dat niet) df.plot.hist(), df.plot.scatter(x=a, y=b)

(df[assignments] > 9).sum().sum() np.linspace(0, 1, 5)

c = df[as].corr(), x = c[c!=1].max().idxmax()

y = c[c!=1].loc[x].idxmax(), pd.Series(data=values, index=labels)

pivot[["F","M"]].min(axis=1) ~~sum()~~.idxmax()

groupby → apply(functie) → unstack() gebruik .values ipv tolist()

df[df.x1.astype(str).str.contains("string")] → Colleen Int()

.loc = rows/cols met labels vd index, .iloc = rows/cols met index posities

%.ls -lh, %.rm <file>, %.time(it), %.prun

.str.replace("|".join([...]))

ggd(g,k): if k==0: return k else: return ggd(k, g%k) kgv(a,b): return a*b / ggd(a,b)

F1 = 2 x (Prec. x recall) / (Prec. recall)

Tips: .TAB , ? , *? , %magic , %time(it) , %ls (-lh) , %rm , %prun , %cp , %cd , %mkdir

Precisie = TP/(TP+FP) , Recall = TP/(TP+FN) , Acc. = (TP+TN)/All

PD Pandas: Series = 1D labeled array, DF = 2D".

pd.read_'type'(file,..) | dimensies: .shape | dubbels: len(set(df.index)) == len(df.index) | orden op index: . namen en hoe vaak dubbel: dubbel = df.index.value_counts() -> dubbel[dubbel>1] | sort_index(inplace=True) rm spaties en "" + lowercase: df.index.str.lower().str.strip().str.replace("'",'').str.replace('"',''). beperk tot substring: df[~df.index.str.lower().str.contains('sub')] | df.describe() | sort_values() new kolom √ van** van verschil: df['n'] = np.sqrt((df.vers1 - df.vers2)**2) | trek gem. van waardes in kolom van elke waarde af / normalisatie: M = df.mean() -> (df[M.index] - M) | plot iets op df: df.kolomnaam.sort_values().plot() | kolom > iets in gemeente + hor. barchart: df[df.kolom.>0] .gemeente.value_counts().sort_values().plot(kind='barh') | Vervang NaN: df = df.fillna(0) rm rows/cols met NaN: new = df.dropna(axis=0/1, how='any') | Perc.: "opdrachten".mean() * 100 | Z-normalisatie: z_df = (df - df.mean()) / df.std() | corr. tussen opdr.: c = df[["..."]].corr() -> c = c[c!=1] = rij = c.max().idxmax() -> kolom = c.loc[rij].idxmax() -> vr = {rij, kolom} | KGV: return int(abs(a*b)/ggd(a,b)) | (Numpy): normaliseer kolom tot 1: (P / (P.sum(axis=0))) | np.diagonal(s)

NP Fill NaN by median: for x in range(L.shape[1]): -> L[:,x][np.isnan(L[:,x])] = np.nan-median(L[:,x]) -> return L | P.flatten() | return values: np.sum(P<n) | vind niet nul: L[np.nonzero(L)] | RMSE: np.sqrt(np.mean((L[:,1] - L[:,0])**2)) | np.arange(1,4)x2 [:, np.newaxis] | sort array L naar R: np.sort(L, axis=1) | test waardes: np.all(np.abs(L) > 10) | select last 3: x[-3:] | select last 3 cols of first 2 rows: x[:2,-3:] | reshape into 1D: x.reshape((x.size,)) | change elem. by ddiff**2 from mean: (x - x.mean())**2 | bereken std: np.sqrt((vraag hiervoor).mean()) == x.std() | bool mask: x[x%3==0] | fancy index diagonal: d = x.shape[0] -> dd = np.arange(d) -> x[dd,dd] | sort sepal lengths for species iris.groupby('species')['sepal_length'].max() | Tokenize text, count tokens, create Series, sort: C = Counter(text.split()) -> CS = pd.Series(C).sort_values(ascending=False) | find tokens: CS[cs * CS.index.str.len() == 24].index() | perc. unique words 1x en perc. of all: (CS==1).mean(), (cs==1).sum()/CS.sum() | and met meeste: bd = pd.read_...(file)[1] -> bd.groupby(0)[1].count().sort_values().tail(1) | npsetdiff1D() | kruistabel: pd.crosstab df.pivot_table() | 2 -> deelstaten | with open(text, encoding='utf-8') as f: -> t = f.read() -> t_2 = re.Split(r'\W+', t) -> pdf = Counter([len(w) for w in t_2]) -> return pdf |

Text

Theory slices + indices = views, no copies. Broadcasting: 2 arrays differ in dimensions, kleinere is padded met 1s als shape niet matcht, array met shape==1 is stretched. sizes niet zelfde en niet==1 -> error.

Python reversed: x[::-1, ::-1] | x[:,0] eerste column / x[0,:] eerste row | np.concatenate -> join arrays along axis | isnull(), notnull() | df[[opdrachten]]>9).sum().sum()

↓ (>5.5 | <5.5)

Numpy

Data types:
- Checking: arr.dtype
- Changing: arr.astype(...)
- int, float, str

Operations:
- arr1 +, -, /, * arr2
- np.exp, sqrt, log, sin, cos (arr)

Creating arrays:
- 1D: np.array([1,4,5])
- 2D: np.array([(1,4,5), (2,3,6)])
- , dtype = ...
- np.arange(10,25,5) → [10, 15, 20]

cols, axis 1 →
rows, axis 0 ↓
They 'collapse' along this axis

Inspecting:
- arr.shape → (2,3) row, column
- len(arr) → 2
- arr.size → 6

Comparing:
- Element-wise:
  - arr1 == arr2 → [T, F, T], [...]
  - arr1 > arr2 → ...
- Array-wise: np.array_equal(arr1, arr2)

Slicing:
- select: arr[1,2] row 1, column 2
- slicing: arr[0:2, 1] row 0/1, column 1
  arr[1] row 1
  arr[:, 1] column 1
- bool indexing: arr[arr > 3] → [10 .....]

Functions:
Or in the form: np.sum(arr, axis=...)

arr.sum(axis=...)
arr.min(...)
arr.max(...)
arr.mean(...)
arr.median(...)
arr.sort(...)
X = arr.view()
x = arr.copy()

Manipulation:
- arr.T → rows become cols, cols become rows
- arr.reshape(...,...) → new arr, same data
- arr.resize(...,...) → same arr, new shape

np.append(arr, x, axis=...)
np.insert(arr, index, x, axis=...)
np.delete(arr, indexes, axis=...)
np.concatenate((a,b), axis=a)
np.vstack((a b))
np.hstack((a,b))

Creating dataframe:
df = pd.DataFrame([4,7,10],
[5,8,11],
[6,9,12],
index=[1,2,3],
columns=['a','b','c'])

Pandas

Functions:
high to low

df.sort_values('price', ascending=False)
df.rename(columns={'y':'year',...})
df.drop(columns=['length',...])
df.sort_index() / df.set_index('A')
df.reset_index() → index becomes 0,1,2,... again

- df['A'].value_counts()
- df['A'].nunique()
- df.dropna() / df.fillna(value)
- df.sum, count, median, ['A']→ mean, min, max()
- len(df) → # of rows

New column / row:
- col: df['new-col'] = [..., ..., ...]
- row: df.append({'a': 4, 'b': 7, ...}, ignore_index = True)

Logic:
df.column.isin(values)
pd.isnull() / pd.notnull()
operators: &, |, ~ (not)
df.any(), df.all()

Subset (rows):
- df[df.length > 7]
  df.iloc[10:20] (by position)
  df.nlargest(n, 'value') } selects & sorters
  df.nsmallest(n, 'value') }
  df.drop_duplicates()
  df.head(n) / df.tail(n)

Subset (columns):
- single col: df['length'] or df.length
- multiple cols: df[['length',...]]
  df.iloc[:, [1,2,5]]
  df.loc[df['a'] > 10, ['a','c']]
  df.loc[:, 'a':'c']
  df.filter(['a','c','d'])

Regex:
. = anything
'\a': containing 'a'
'a$': ending in 'a'
'^a': starting with 'a'

Plot:
df.plot.barh, line, etc.

Combining:
on top of / each other
pd.concat([df1, df2])
next to each other: axis=1
pd.merge(df1, df2, how='inner', on='col')
Filtering: df1[df1.A.isin(df2.A)]

Grouping:
df.groupby(col)[other_col].count()
index / value
.size()
or .max(), .min(), etc.

Pivot table:
df.pivot_table(index='', columns='', values='', aggfunc=np.sum)

Reading Files:
pd.read_csv(parameters!)

String operations:
S.str.count(r'(^F.*)').sum()
S.str.match(r.) → returns bool
S.replace('...', '', inplace=True)
S.str.findall(r.)
S.str.contains(r.) → returns bool
S can be df[df['A']...]
For example

iPython
% + tab → all magic commands

Numpy cheatcheat

np.**? pd.**? dt.**?

- np.arange(start, stop, step) → make numbers between start and stop with step space between
- np.linspace(start, stop, number=) → make x numbers between start and stop
- np.array([(1,2,3),(4,5,6)])
- a.shape & a.size + a.ndim + a.dtype + a.astype(int) + np.info
- np.concatenate(a)
- np.**? → all functions

Reading files in python → CSV, xml, JSON, PDF, HTML

- general logic
  - 1. open file → 2. read using special purpose module → 3. convert to dict or list
  - spreadsheet to list of lists or list of dicts using colums and indexes + headers
- Specific formats
  - CSV → pandas
  - xml → lxml library and xpath
  - JSON → json.load + json.dump
  - PDF → pdfminer
  - HTML → Beautifulsoup and requests
- Read from disk
  - %ls .. /Data/ → what files are in folder?
  - f = open('../Data/Monumentale_Bomen.csv')
    - f.readline().split(';')
  - with open('../Data/...csv') as f:
    - allLines = []
    - for line in f:
    - allLines.append(line.split(';'))
  - dict = json.loads(test)
  - titles = tree.xpath('//item//title//text()')
  - magic commands
  - %ls .../Data/ → mijn_files
  - for f in mijn_files:
  - !wc -w "$f"

Robert45631#

np practice

- np.full((3,3), True, dtype=bool)
- np.where(arr%2==1, -1, arr) → copy
- np.r_[np.repeat(a,3), np.tile(a,3)]
- np.intersect1d(a,b)
- np.setdiff1d(a,b)
- P/P.sum(axis=0) → normalize colums

-L[((L<-10)|(L>10)]

Numpy

- return P/P.sum(axis=0) → normalize colums to 1
- np.diag
- def rmse(L):
  - test = (L[:,1] - L[:,0])**2
  - return np.sqrt(test.mean())
- np.arange(1,4)[:, newaxis]
- np.sort(L)

np practice

- rmse: test = (L[:,1] - L[:,0])**2
  - return np.sqrt(test.mean())
- a = np.arange(1,4)[:, np.newaxis] b = np.arange(1,4)
  - array([[2,3,4], [3,4,5], [4,5,6]]) → broadcasting
- Q: From array return values < -10 and values > 10
  A: return L[(L<-10)|(L>10)]

Theorie

- precision = TP/(TP+FP) → how many positives are correct?
- recall = TP/(TP+FN) → how much is detected?
- accuracy = TP+TN/(TP+FP+FN+TN) → reliability of test
- F1 = 2*(recall*precision)/(recall+precision)

	True	False	
pos test	TP 90	FP 990	1080
neg test	FN 10	TN 8910	8920
	100	9900	10000

N = 10000
P = 0,01
precision = 90/(90+990) = 0,083
accuracy bad for skewed test (close to 1)

Theorie

```
def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b, a%b)
def lcm(a,b):
    return abs(a*b)/gcd(a,b)
```

Pandas

- pd.crosstab(col1, col2)
- cross.plot(kind='bar')
- cross.plot.barh()
- kvr.str.lower().str.replace('[^\w]|allen','')
- kvr = kvr[~kvr.partij.str.contains('vragen', na=False)]
- vragen['pvdd'].sort_values(ascending=False)
- kvr[['Antwoord']].applymap(lambda x: len(x))
- kvr[['vraag']].applymap(lambda x: str.count(x,'?'))
- vraag.hist(bins=50)
- vraag.corr().loc['kar'][0]

- df.sort_values(['All','Name'], ascending=[False,True])
- df.query('ratio<0,1 and ratio>-0,1')
- df.index.values
- error['rel_error'].idxmax()
- .sum(skipna=True)

Pandas 2

- ax = df.plot(kind='line')
- ax.set_xlabel('Year')
- ax.set_title('Title')
- ax.legend(['F','M'])
- str='Title...%ss'%(var)

- ind = [3,7,4]
- x[ind]
- fancy indexing
- x[row1, np.newaxis, col]

```
df = pd.DataFrame(
    {'a': [4,5,6],
     'b': [7,8,9],
     'c': [10,11,12]},
    index=[1,2,3])
```

- df.groupby(by='col')
- df.groupby(level='ind')
- pd.merge(a, b, how='outer', on=

pd.read_csv(File, sep='\t', index_col=0, header=None, skipinitialspace=True, compression='gzip')
tab, space — eerste colom index — zelf kolomnamen bedenken — regex

.columns['1','2'] ...

pd.crosstab(kvr.~~partij~~ jaar, kvr.partij)[partijen] — ijzer

[ ] vs [[]] — pSeries DF — [sS] either — r'\b[Ss]ex\b'

.str.replace/lower/contains — \W niet woordelijk — \w woordelijk

Kvr = kvr[~kvr.partij.str.contains('vragen')] — Acc = TP+TN/(TP+FP+FN+TN) — prec = TP/(TP+FP) — recall = TP/(TP+FN)

Nieuwe kolom — hier

kvr['deelv'] = kvr.vraag.str.count('\?')

pd.pivot_table(df, index, values, colums, aggfunc, margins=True)

sort values(by=['All','name'], ascending=[F,T]

import copy
copy.deepcopy(x)

pivot.drop[pivot.ratio<0.1].index — ambigynames

kvr.Partij.value_counts().index.tolist() hoogste partijen

L[:,1] kolom 1 — L[L%n==0] — L[3:7, 3:7] kolom rij vierkant — Alles deelbaar door n — L[:,i]+=i elke ide kolom + i

np.arange(0, 7, 7x144) start stop stap

reshape(-1, i) rij kolom — zoekt zelf uit

pivot... values ['D','E'] ... aggfunc={'D':np.mean, 'E':min...

np.random.randint(low, high, size)

plt.plot(x-as lijst, lijstvalues, label)
plt.show()

.split('str')

DF.groupby([kolom1, kolom2])
.apply(lambda x: x[:n].count,sum)
.unstack()

- <TAB> = mogelijkheden van functie

np.random.randint(begin, eind, size=(kolom, lengte))

ndim = number dimensions
shape = lengte per level (dimension)
size = totale lengte (len)

slicing: x[start:stop:step]
[row, column]

Reshape: (rijen, lengte)

joining arrays = np.concatenate([x,y], axis)
↑ vstack & Hstack

split arrays: np.split

ufunc:
- add, -substract
- negative, -multiply
- divide, -power (^2)
- mod

np.count_nonzero(if) → hoeveel van iets
np.argwhere() → selecteer alles waar voorwaarde

np.nan...() → negeert nan values

np.all/any(array)

np.random.random((x,x))
↳ array w random values

np.set_printoptions() → beetje weergeeft

pd.DF(dict) → series → DF

df[~df.isin()] → where functie
pd.concat([ser1, ser2], axis=1)
↳ Hstack
pd.Series([list comprehension])

pd.<TAB> - mogelijkheden

index_col → index by col

df[[kolomen]] → waarde van alleen die kolomen

.isnull - .notnull

MultiIndex → SF 2000 3 mil
2010 5 mil
.unstack → dataframe

pd.concat(ignore_index)
↳ verzint nieuwe index voor overlappende

pd.concat(keys=[":"])
↳ laat zien welke waarde van welke df is

pd.merge() → merge df's
↳ Left/right_on = kolom waarop moet mergen

.Groupby.count() = hoeveelheid

pd.pivot_table(index, kolom, waardes, aggfunc
↓ functie

pd.crosstab() → laat relatie zien tussen klassen

		predicted P	predicted F
actual	P	TP	FN
	F	FP	TN

Accuracy = How many of all predictions were correct $\frac{TP+TN}{TP+TN+FP+FN}$

Precision = how many of the predictions were correct $\frac{TP}{TP+FP}$

Recall = how many of the instances were retrieve $\frac{TP}{TP+FN}$

$RMSE = \sqrt{\frac{\text{sum}(\text{predicted}-\text{actual})^2}{n}}$

#KijkGOED naar de vraag

%timeit 'rest' to time rest

Numpy

Create np.zeros
ones
Full
random.randint
normal
.arange(start, stop, step)

A.size
A.shape
A.sort
A.argsort
np.argwhere

Index 2D[row, col] slice = 2D[start:stop:step, ''] (start:stop:step, '')
A[::2] = every other val
A[::-1] = reverse
A[:2, :3] First 2 cols, First 3 row

Computation Arr1 + Arr2 → calculates element wise

+ np.add
- Subtract
/ divide
// Floor-divide
*-1 negative
** Power
* multiply
% mod
exp
Power
log / log2 / log10

operators (can be given along different axis/series etc.

np.Sum
- min
- max
- mean
- median
- Std
- var
- Percentile

- argmin
- argmax
  → gives indexes

- nan'operator'

Min, Max, etc

Reshape, concat, split A.Flatten → (n,)

A.reshape(3,3) = 1D len(9) → 2D len(3,3)
A[:, newaxis] = makes array (10,)

np.concatenate([A, A]) = 2x as long
'' ([A, A], axis=1) = extra column

np.vstack / np.hstack

A1, A2, A3 = np.split(A, [3, 6])
→ [1,2,3], [4,5,6], [7,8,9]

np.vsplit
np.hsplit

Broadcasting

works when array's:
1. are equal (dimensions)
2. one of them == 1

so (9,) and (9,2) will not work

$SD' = (x - mean)^2$
$SD = \sqrt{SD'.mean()}$

$Precision = \frac{TP}{TP+FP}$

$Recall = \frac{TP}{TP+FN}$

$Acc = \frac{TP+TN}{All}$

ziekte komt voor 1 op 10
Accuracy = 90%.
$P = (0.1 \times 0.9) / ((0.1 \times 0.9) + ((1-0.1) \times 0.1))$

numpy

np.array() / np.arange()
np.ones() / np.zeros() / np.full() / np.linspace()
np.random .random() / .choice() / .randint() / .seed() / shuffle()
.ndim / .shape / .size
.reshape() / x[np.newaxis, :] / np.flatten() / np.transpose()
np.concatenate() / np.vstack() / np.hstack() / np.repeat()
np.split() / np.(h/v/d)split()
np.add() / substract() / multiply() / divide() / mod()
np.negative() / abs() / exp() / log() / loge() / power()
np.reduce() / np.accumulate() / np.cumsum() = (all vs steps)
np.outer() (multiply table)
np.sum() / prod() / mean() / mode() / std() / var() / min() / max()
np.argmin() / ~~np.idxmin()~~ / median() / percentile() (pleur non encor)
np.any() / all() -- &, |, ~(not) / .isin()
np.sort(axis=..) / np.argsort() -- axis=0 = v / axis=1 = h
np.dtypes({'names': (''), 'formats': ('')})
np.diagonal() / np.setdiff1d()

x[y, x] : = hele ry -1 = achterste x[:-1] = reversed x[x>3]

np.where()

pandas

pd.Series() / pd.DataFrame()
.values / .index / .columns / .size / .shape / .ndim
.keys() / .items()
.mean() / .mode() / .median() / .quantile() / .corr() / .log2()
pd.notnull() / isnull() / dropna() / fillna()
.str. len() / lower() / upper() / startswith() / find() / replace() / etc
.value_counts() / .nlargest()
.max() / min() / idx max()
.groupby / .pivot_table() / crosstab()
.dir / .apply
.plot() / .histogram()
.pd.read_csv()
.sort_values()
.loc() / .iloc() / .ix() = explicit / implicit / hybrid
pd.MultiIndex() / reindex() / unstack() / stack()
~~pd~~.reset_index() / set_index()
pd.concat() / .append() / transpose()
& = intersection | = union ^ = symetric difference

df['Index'], df.index df[df>3]
Pd.merge() / .join()
.tail() / .head()

help

shift+tab
?
.tab
%xmode Verbose
" Context
+word?
help(function)

accuracy = (Tp+Tn) / All Tp = R.A (ratio/accuracy), 1 interacting with outside World
precision = Tp/Tp+Fp Fn = R.(1-A) = P 2 preperation
recall = Tp/Tp+Fn Tn = (1-R).A 3 transformation
F1 = 2x(Rec x pres)/(rec+pres) Fp = (1-R).(1-A) 4 modeling & computation
type 1
type 2
5 presentation

Re

- findall() / sub() / replace()
\W = letters/cyfers \d = cyfe
. = of neer + = 1 of meer
{} = exact times
^ = match anything not in set
" ".join(lyst) <.+?>

based on patterns in data predicting a value on new unseen instances of data / classification / regression
unlike statistics → no explaining
RMSE
selection = [min, max, etc]
bagg = random shuffle
breadfirst = knap, dan kruisen etc

Numpy

dir(np)

np.arange (start, stop, step) generate array

np.concatenate join arrays
- vstack
- hstack

h/v split split points as list of indices

np.nan
- sum
- prod
- mean
- std
- var
- min/max
- argmin/max index of value
- median
- percentile rank-based stats
- any*
- all*

- .ndim nr of dims
- .shape size per dim
- .size total size
- .reshape in shape (x,y)
- .flatten 1D array
- .T transpose

Numpy slicing

- np.newaxis
- np.sort
- np.where

1D x[::2] every other el.

1D x[::-1] reverse array

2D x[:2, :3] row, cols

2D x[:, 0] first col.

2D x[0, :] first row

2D x[::-1, ::-1] reversed

Numpy Indexing

x[y-as, x-as]
row, column
axis=0, axis=1

Ufuncs

- np.add +
- np.subtract -
- np.negative -
- np.multiply *
- np.divide /
- np.floor_divide //
- np.power **
- np.mod %

Boolean & Masks

- & and
- ^ xor
- | or
- ~ not

x[x > 5] select values that meet requirement

Fancy Indexing

pass array of indices

(2, 1)(1, 0)
→ [2,1] x [1,0]

X[[2, 0, 1], [1, 2, 0]]

Broadcasting

Apply ufuncs to arrays of different sizes

Pandas

dir(pd) of pd.series

- pd.read_csv inlezen bestand
- .values
- .index
- pd.Series(data, index=index)
- [a:b] incl a, excl. b selection
- pd.DataFrame(input, columns, index=[])
- serie['index'] >>> value
- 'a' in serie T/F boolean
- serie.keys() get indices
- list(serie.items()) returns list of tuples
- serie['index'] = 3 change value of 'index'
- serie[(serie > x) & (serie < y)] masking
- serie[['a','b']] fancy indexing
- df.columnname > series of column, index + value
- df['columnname]"
- df.values display df as 2D array
- df.T Transpose
- df.values[0] returns row
- df.values["index"] returns column
- !df[df[0]] get index for value
- for index, row in df.iterrows()
- fill_value =0 replace nan

& and
| or
~ not

indexers
- .loc Explicit index
- .iloc Implicit index
- .ix combination

Expl. index: last index INcluded
IMplicit: last index EXcluded.

↳ direct masking is interpreted row-wise!

NB. Indexing refers to columns [..., ...]
Slicing refers to rows [... : ...]

happens row-wise due to broadcasting
- add() +
- sub() -
- mul() *
- div() /
- floordiv() //
- mod() %
- pow() **

- .isnull() / .notnull() boolean mask
- .dropna() drops r/c where nan > present (all, any) thresh
- .fillna()
- .sort_index()
- .sort_values() pass list of columns on which to sort
- .unstack
- .stack
- .set_index / reset_index
- .join
- .append

pd.concat standard row-wise

- .describe()
- .count()
- .sum()
- .prod()
- .mad()
- .first, .last
- .mean(), .median()
- .min(), .max()
- .std(), .var()
- .groupby() split according to key, compute and combine
- .pivot_table = multidim > groupby aggregation
- .isin
- .hist
- .str.split / replace / join
- .merge
- .value_counts

Steps of DS

1. Interaction with the world
2. Preparation
3. Transformation
4. Modeling & Computation
5. Presentation

Precision $\frac{TP}{TP + FP}$

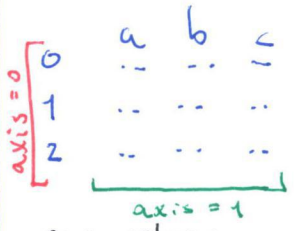
Recall $\frac{TP}{TP + FN}$

	ziek	niet ziek
test p	TP	FP
test n	FN	TN

Predicting:
Classification spam/no spam
Regression value eg. for course

name where age < 10
df[df['age'] < 10]['name']

NB. Broadcasting!



row, column

% ls magic = list met alles wat beschikbaar
% magic
% ls
% cd change dir
% rm remove
% time / % timeit (test)

np.abs
np.newaxis voor kolom: row na komma column
np.sort

vb. array.shape >> 4,
[:, np.newaxis] >> 4, 1
[newaxis, :] >> 1, 4

Accuracy: (TP+TN)/totaal
Precision: TP/(TP+FP)
Recall: TP/(TP+FN)
F1: 2 · pr·re / pr+re

predicted \ actual	ziek	niet ziek
ziek	TP	FP
niet ziek	FN	TN

Ziekte met 95% accuracy, 1/100 personen
N = 10000 P = 0,01
TP 0,01 · 0,95 · 10000 = 95
FN 0,01 · 0,05 · 10000 = 5
TN (1 − 0,01) · 0,95 · 10000 = 9405
FP (1−0,01) · 0,05 · 10000 = 495
Ziekte met 90% accuracy, 1/10 personen
P = 0,1
Precisionziek (P · 0,9) / ((P · 0,9) + ((1 − P) · 0,1))
Precisionnietziek ((1 − P) · 0,9) / ((P · 0,1) + ((1−P) · 0,9))

Z-score: (df − df.mean()) / df.std()
RMSE: np.sqrt(np.mean((L[:,1] − L[:,0])**2))
echt gemeten

```
def GGD(g,k):
  if k == 0:
    return g
  else:
    return GGD(k, g%k)
```

```
def kgv(a,b):
  return int(abs(a·b) /
             ggd(a,b))
```

middlesquare (4x4)
k = L.shape[1]
st = int((k/2)−2)
ei = int((k/2)+2)
L[st:ei, st:ei]

reshape square
array 144 → 12 x 12
L.reshape(int(math.sqrt(L.size)), int(math.sqrt(L.size)))

tafel
eerste n getallen, deelbaar door k
np.arange(0, n·k, k)

euclidean distance
np.linalg.norm(a−b)

[0,1,2 etc] optellen bij L
z = np.arange(0, np.size(L,1), 1)
L + z

Vertical reshape
L.reshape(L.size, 1)

5e herhaling van 1 in array x
np.where(x==1)[0][5−1]

weg nummers only
geen = df.kolom.str.contains('[^0-9.-]')
df = df.loc[~geen]

```
Arr = np.array([0,9,1,4,64])
Arr[1:3] → 9,1
Arr[:2] → 0,9
Arr[:-2] → 4,64
Arr[:-2] = 50 → 0,9,1,50,50
Arr[arr<3] → 0,1
Arr[arr<3] = -1 → -1,9,-1,4,64

Arr2 = [[1,2,3],
        [4,5,6]]
Arr2[0] → 1,2,3
Arr2[[0,1]] → 1,5
Arr2[:,0] → 1,4
Arr2[0,0] → 1
Arr2[1,0] → 4
Arr2[1,-1] → 6
Arr2[:2,::3] → hele arr2
Arr2[:2,::2] → [[1,3],
                [4,6]]
Arr2[::-1,::-1] → [[6,5,4],
                   [3,2,1]]
Arr2[2,[2,0,1]] → 6,4,5
```

SLICING

```
create [[2,3,4],
        [3,4,5],
        [4,5,6]]
a = np.arange(1,4)
b = np.arange(1,4)[:,np.newaxis]
a + b
```

rij
np.sort(L, axis=1)
np.sort(L, axis=0)
kolom

np.all((L<-10) | (L>10))
↳ True of False

np.random.randint(n, size=...)
size = 6 → 6 getallen
= (3,4) → 3 hoog, 4 breed
= (3,4,5) → 3 keer 4 hoog, 5 breed
np.random.random((3,3)) 3x3 kolom
np.subtract(a,b) a−b
np.add(a,b) a+b
np.divide(a,b) a/b
np.multiply(a,b) a·b

```
X = np.arange(10)
X[::2] → tafel 2, start = 0
X[1::2] → tafel 2, start = 1
X[::-1] → andersom g t/m 0
X[5::-2] → 5, 3, 1

Y = np.arange(9).reshape(3,3)
Y[:,[2,1,0]] → switcht kolom 0 & 2
Y[[0,2,1],:] → switcht rij 2 & 1
Y[::-1] → reverse rijen
Y[:,::-1] → reverse kolommen
```

fill with median(L)

```
for x in range(L.shape[1]):
  L[:,x][np.isnan(L[:,x])] =
    np.nanmedian(L[:,x])
return L
```

```
smallerthan(P,n):
return np.sum(P<n)
```

notzero(L) alle elementen die niet 0 zijn

```
return L[np.nonzero(L)]
```

NUMPY

```
np.mean(arr,
  axis=0)
```

arr.reshape(x,y)
x − lengte
y − breedte

np.sum(x<6, axis=1)
↳ in each row

np.transpose(L)
10,15 / 33 34 → 10,33 / 15,34

reshape x2 → x1
x2.reshape(x2.size)

variance
(x1 − x1.mean())**2 = V

std
np.sqrt(v.mean())
== x1.std()

divisible by 3
x1[x1 % 3 == 0]

diagonal
d = x2.shape[0]
dd = np.arange(d)
dia = x2[dd,dd]

```
pd.read_csv('file', sep='\t', index_col=0,
header=None, names=[''], compression='gzip',
skipinitialspace=True)
df.kolom.value_counts() hoogste bovenaan
df.sort_values(['kolom1','kolom2'],
  ascending=[False,True]) alfabetisch
                  ↓ hoog naar laag
df['kolom'].corr(df['kolom2'])
df['kolom'].str.count(r'\?')
str.len(), str.lower(), str.replace("","")
df.dropna(subset=[], inplace=True)
df.drop(columns=[])
pd.concat → samenvoegen df's
pd.merge
len(set(df.kolom)) = df.kolom.unique().size
                     aantal unieke in kolom
df.groupby('kolom')['lengte'].max()
  ↳ max lengte van kolom
restrict df[df['kolom']>5]
c = Counter(text.split())
cs = pd.Series(c).sort_values(ascending=..)
cs[cs.index.str.len()==24].index
  ↳ tokens which occupy 24 char in total
unique words percentage
cs[cs==1].sum()/cs.count() of
(cs==1).mean()
percentage all words
cs[cs==1].sum()/cs.sum()
cs.value_counts().sort_index().plot(
  kind='bar')
each number i occurances, how many tokens in text occur i many times
df.pivot_table(index='Name', columns='sex',
values='count', aggfunc=sum, margins=True).
sort_values(by=['All','Name'], ascending=[False,True])
pd.crosstab(df['kolom'], df['kolom'])[index]
  bijv iets met value counts
str.replace(r"[^A-Za-z0-9]+", "") verw. spatie & tekens
df['kolom'].str.len() of mean/mode etc.
df[df[''].str.contains('vr')==False] = df
  ↳ verwijder rijen waar vr in voor komt
df[~df[:,:].duplicated()]
```

PANDAS

precisie = $\frac{TP}{TP+FP}$

recall = $\frac{TP}{TP+FN}$

acc = $\frac{TP+TN}{TP+TN+FP+FN}$

F1 = 2* $\frac{precision * recall}{precision + recall}$

dir()
help()
?
??
.<TAB>
*?

predicted \ actual	has disease	no disease
has disease	TP	FP
no disease	FN	TN

```
np.array([], dtype=)
np.zeros()
np.ones()
np.arange()
np.linspace()
np.full()
np.random.random()
np.empty()
np.arange()[:,np.newaxis]
np.random.randint(n, size=...)
random.choice(a)
```

```
a.shape
len(a)
a.ndim
a.size
a.dtype
a.dtype.name
a.astype(int)
```

```
a-b   np.subtract(a,b)
a+b   np.add(a,b)
a/b   np.divide(a,b)
a*b   np.multiply(a,b)
a==b
```

```
a.sum()
a.min()
a.max()
a.mean()
a.median()
a.corrcoef()
np.std(a)
np.log(a)
np.argmax()
np.any()
np.all()
np.exp()
```

```
np.transpose(a)/a.T
a.ravel()/a.flatten()
a.reshape()
a.resize()
np.append(a,b)
np.insert(a, 1, 5)
np.delete(a,[1])
np.concatenate((a,b), axis=)
np.vstack()/np.r_[a,b]
np.hstack()/np.c_[a,b]
np.hsplit(a,3)
np.vsplit(a,2)
```

```
a.sort(axis=)
np.sort(a)
np.argsort(a)
np.partition(a,3)
```

X[row, colum]

```
   0  1  2  3  4  5  6  7  8  9
0 [[1  8  2  3  2  9  9  1  9  1]
1 [5  6  5  3  6  8  2  6  8  1]
2 [1  3  7  4  5  3  8  7  3  3]
3 [7  4  4  9  9  6  6  1  4  9]
4 [6  8  9  6  8  4  3  1  6  4]]
```

ex[:,4]
ex[:-2,9]

axis=1 row
axis=0 column

& intersection
| union
^ symm diff

```
df['w'].value_counts()
len(df)
df['w'].(n)unique()
df.describe()
sum()
count()
median()
apply(function)
min()
max()/idxmax()
mean()
var()
std()
mode()
df['w1'].corr(df['w2'])
```

```
df['w'].str.count(r'\?')
str.len()
str.contains()
str.lower()
str.upper()
str.replace()
df['w'].isin(values/column)
pd.isnull(obj)
pd.notnull(obj)
&, |, ~, ^, df.any(), df.all()
df.dropna(subset=[])
df.fillna(value)
str.split('\')
str.get('')
pd.concat()
```

```
df.values
df.index
df.columns
df.sort_values('', ascending=]
df.rename(columns={'y': 'year}
df.sort_index()
df.reset_index()
df.drop(columns=[])
df.shape()
df.tail()
df.head()
df.plot(kind='barh') → logx=True
list(df.items())      logy
```

html: r'<[^>]*>'
leestekens: r'\W|\s'

kolomnormalisatie: a/a.sum(axis=0)
rijnormalisatie: a/a.sum(axis=1)
Z-normalisatie: (df-df.mean())/df.std()
RMSE: 1. np.~~sum~~mean((y-echt - y-gemeten)**2)
2 np.sqrt(antwoord 1)

reverse columns: a[:, [2,1,0]] / a[:,::-1]

reverse rows: a[[2,1,0], :] / a[::-1, :]

max length for each species: iris.groupby('species')['length'].max()

```
def ggd(g, k):
    if k==0:
        return g
    else:
        return (k, g%k)

def kgv(a,b):
    return int(abs(a*b)/ggd(a,b))
```

middlesquare:
k=L.shape[1]
st = int((k/2)-2)
ei = int((k/2)+2)
L[st:ei, st:ei]

vertical reshape
L.reshape(L.size,1)

euclidean distance
np.linalg.norm(a-b)

notzero(L):
return L[np.nonzero(L)

diagonal
d = xz.shape[0]
dd = np.arange(d)
dia = xz[dd,dd]

```
pd.read_csv('file', sep='\t', index_col=0, header=None, names=[''], compression='gzip', skipinitialspace=True)
os.listdir()
```

```
df.pivot_table(index='Name', columns='Sex', values='Count', aggfunc=sum, margins=True).sort_values(by=['All', 'Name'], ascending=[False,True])
```

```
df.groupby(['','']).apply().(sum().)unstack()
```

as 1 column

L[:,-1]
L[::2]
pL[::-1] = reverse

Shape = rij, kolom Recall = TP/(TP+FN) axis = 0 → kolom S.max(). idxmax()
Size = datacellen precisie = TP/(TP+FP) axis = 1 → rij ↓ max waarde ↓ locatie max waarde

df = pd.read_csv('file', index_col='names')

file[:,2010] rijen met index 2010

df.sort_index(inplace=True) of df = df.sort_values(['...', 'names'], ascending = [False, True])
(pivot. sort)
a[(a<=3) & (a==5)]
file[(file[...]=='Piraat') | (file['...']=='2010')]['Vragen']

groupby

1880 F 100
M 80

all boys/jaar file[file['sex']=='M'].groupby('Year')['births'].sum() | groupby(['Year','sex']) kan ook
(index, value)

pivot FMAIL df.pivot_table(values='count', columns='Sex', index='name', aggfunc=sum, margins=True)
name =

crosstab x = file['party'].value_counts()[:10] → pd.crosstab(nur['Jaar'], nur['Party'])[x.index]

nur.dropna(subset=['...'], inplace=True) | nur['...'].str.lower() | .str.replace('.','') | .str.count('\?')
df.fillna(0, inplace=True)

Z-norm (df - df.mean(axis=1)) / df.std() RMSE → np.sqrt(np.mean((L[:,1]/L[:,0])**2))

mean df[['...','...']].mean(1) | (df[['...','...']]>9).sum().sum() pd. serie

% / kans ((df['...']>5.5) | (df[...]>5.5)).mean()*100 | # narauwers per woord serie = serie.map(lambda x: len(x))

min * 100 uitvinden johns # births file[file[sex]=='F'].loc['John']['births'].plot() | file[~file.index.str.lower().str.contains('schaai')]
not

df.isnull().values.any()

kans = 0.1 % 10
accuracy = goed / totaal TP precision = (kans*0.9)/((kans*0.9)+((1-kans)*0.1)) TN prec. = ((1-kans)*0.9)/((kans*0.1)+((1-kans)*0.9))
als recall = 0.95 kans = % 100 precision = (0.01*0.95)/((0.01*0.95)+(0.99*0.05))

numpy → arange, reshape, transpose, sort, intersect1d, np.all, size, shape setdiff1d

gem. alleen voldoendes → L[L≥5.5].mean() | kans op voldoende (L≥5.5).mean() | vertical shape np.reshape(L,(np.size(L),1))

+1 +2 +3 +4 L + np.arange(L.shape[1]) | np.arange(0, 0*n, 1*0) | kolom normaliseren → L / L.sum(axis=0)

loop kolommen for x in range(L.shape[1]): L[:,x][np.isnan(L[:,x])] = np.nanmedian(L[:,x])

test all values Return np.all(L[(L>10) | (L<-10)]) | %3 L[L%3==0] | np.where[a==b]

reshape((4,-1)) -1 doet automatisch # kolommen

correlatie → df[['...','...']].corr() dan c = c[c!=1] rij = c.max().idxmax() kolom = c.loc[rij].idxmax()

swap → x = np.arange(9).reshape(3,3) return np.array(x[[1,0,2],:])

median → float(np.median(df['...'])) | ratio → pivot['...'] = np.log2(pivot[M] / pivot[F])

reshape square → x = np.sqrt(len(L)).astype(int) return np.reshape(L,(x,x))

reshape paren → np.reshape(x, (int(np.size(x)/2), 2)) | np.concatenate((x,y,z), axis=1) pee rij merge arrays

vertically stack → np.vstack([x, grid]) | np.vsplit(grid, [2]) | np.ones((3,3)) | np.arange(3)[:, np.newaxis]
hstack h

random → np.random.randint(0, 10, (2,3)) | x = array, i = array mag dan x[i] = 99 of x[i] == 10

distance points → i = X[:, np.newaxis, :] - X[np.newaxis, :, :] i.shape

oneven getallen → x[x%2==1] = -1 | [np.repeat(a,3), np.tile(a,3)] | 2d array floats tussen 5-10 np.random.uniform(5, 10, size=(5,3))

items of A that are not in B pd.to_datetime(a) A[~A.isin(B)] | pd.Series(np.union1d(a,b)) | file['name'].value_counts() algesorteerd!

convert array df → df = pd.DataFrame(file.values.reshape(7,5)) | stack df = a.append(b) horizontaal pd.concat([a,b], axis=1) verticaal

hoofdletter zin → a.map(lambda x: x.title()) | iris.groupby('species')['sepal.length'].max() max per species

tokenize → C = Counter(text.split()) CS = pd.Series(C).sort_values(ascending=False)

column with min by max per row → df.apply(lambda x: np.min(x)/np.max(), axis=1) % 15 % cp

eucl distance → np.linalg.norm(a-b) | column next max # missing values x = df.apply(lambda x: x.isnull().sum()) x.argmax() 'Luggage.room'

	predicted negative	predicted positive
negative	TN	FP
positive	FN	TP

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{precision} = \frac{TP}{TP + FP} \qquad \text{recall} = \frac{TP}{TP + FN}$$

$$f = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$STD = \sqrt{\frac{\sum (x_i - \mu)^2}{n}}$$

square = x^2
rooted = $\sqrt{}$
squarerooted = $\sqrt{}$

*.values → Series to Numpy
*.T → transposes a Dataframe

+ add
- substract
* multiply
/ divide
// floor divide

% lsmagic = list of magic
% magic = handbook of magic
% ls = list of files in directory
% cd = change current directory
% cp = kopeer file naar andere file
% rm = remove file
% time, % timeit, % prun

np.random.randint
np.newaxis
np.where
np.repeat
np.concatenate
np.sort

df.sort_index
df.sort_values

pd.Series.value_counts

Regex

\ escape character
\d any number
\s space
\w any character
. any except newline
\b white space around word
\n new line
\S space
\t tab
\r return

numpy

create 1D Array = np.arrange(10)
create bool array = np.full((3,3), True, dtype=Bool)
extract odd numbers = arr[arr %2==1]
create new array with condition = np.where(..., ..., arr)
reshape array = np.reshape(r, c) # -1 → automatisch
stack arrays vertical = np.concatenate([a,b], axis=0)
stack arrays horizontal = np.concatenate([a,b], axis=1)
repeat element 3 times = np.repeat(a,3) # (111222333)
repeat array 3 times = np.tile(a,3) # (123123123)
build arrays quickly = np.r_[a,b]
get common items = np.intersect1d(a,b)
from a remove all b = np.setdiff1d(a,b)
get all matches = np.where(a==b)
get between range = a[(a>=5) & (a<=10)]
check values between 2 arrays = np.vectorize(functie, otypes=[float])
swap columns = ~~arr[:,[2,0,1]~~ arr[:,[2,0,1]] # swap rows
reverse rows = arr[::-1]
reverse columns = arr[:,::-1]
random floats = np.random.uniform(5,10, size=(5,3))
limit decimal = np.set_printoptions(precision=3)
correlatie = np.corrcoef
check for any nan = np.isnan(A).any()
get unique values in array = np.unique(a)
sort 2D array by column = A[A.argsort()]
max value each row = np.amax(a, axis=1)

pandas

dataframe van 2 series: pd.concat([A,B], axis=1)
give name to series A.name = "..."
from ser1 remove items in ser2: ser1[~ser1.isin(ser2)]
get percentile: np.percentile(A, q=[0,25,75])
calculate frequency: a.value_counts
reshape series to df with shape: pd.dataframe(ser.values.reshape(7,5))
get all values for condition: np.argwhere(ser % 3==0) # of andere conditie
take items from positions: ser.take([0,2,5,8])
stack two series: pd.concat([ser1,ser2], axis=0 of 1)
get loc from items in ser2 in ser1: [pd.Index(ser1).get_loc(i) for i in ser2]
map every element in series: ser.map(lambda x: functie)
make datetime: pd.to_datetime(a)

only columns from read = pd.read_csv(..., usecols=[...])
reverse rows of df = df.iloc[::-1, :]
combine 2 df = pd.merge(df1, df2, how='inner', left_on['fruit'], right_on['name'], suffixes['left'
files inladen = pd.read_..., index_col=...
unique values = a.index.value_counts
order by index = a.sort_index.

check duplicates: len(set(city.index)) == len(city.index)
kwadraat = ** 2, wortel = np.sqrt(a).
plot = .plot() of .plot(kind='barh')
%%time = tijdmeten
*pivot_table = A.p_t(index='', columns='', values=''), aggfunc [np.mean, np.median]
cross_tab = pd.cross_tab(a,b)

parse HTML

with gzip.open(..., 'r' as f:
url = "..."
html_doc = request.get(url)
soup = BeautifulSoup(html_doc.content)
find all comments
comments = soup.findall('li', class_="comment",
beautify struct. ....prettify()
make tuples list [(soup.title, c.p.text etc)]
df = pd.Dataframe(tuples)
df.columns = ['A', 'b'] index=false
naar excel of csv
df.to_excel, df.to_csv

regex

.findall('')
'' = r'\b[A-Za-z0-9]\b'

PRECISION + RECALL

THERE IS A TEST FOR A DISEASE WHICH HAS AN ACCURACY OF 95%. THE DISEASE occurs IN ONE IN EVERY 100 PERSONS. THIS ACCURACY MEANS THAT THE TEST IS 95% RELIABLE.

HIGH ACCURACY (OR RELIABILITY) DOESN'T SAY much WHEN:
- WITH HEAVILY SKEWED DISTRIBUTIONS (E.G. DISEASE →)
- YOU CAN GET A HIGH RELIABILITY BY THE LAW OF LARGE NUMBERS ON THE FALSE CLASS
- EVERY TIME YOU PREDICT SOMEONE DOES NOT HAVE THE DISEASE → ACCURACY DOESN'T SAYING THAT NO ONE HAS THE DISEASE GIVES AN ACCURACY OF 95%
- BUT ALSO A PRECISION OF 0 AND RECALL OF 0 TOO.

RATIO → 1:10 ACC → 90%

P = 1

PRECISION/RECALL = CP x .9) / ((P x .9) + ((1-P) x 1))

PRECISION/RECALL = ((1-P) x .9) / ((P x .1) + ((1-P) x .9))

RATIO → 1:100

P = .01

PRECISION/RECALL = CP x .9) / ((P x .9) + ((1-P) x .1))

N=10,000	HAS DISEASE	NO DISEASE	
POS. TEST	95	495	590
NEG TEST	5	9405	9410
	100	9900	

	ACTUAL POS	ACTUAL NEG
PREDICTED POS	TRUE POS	FALSE POS
PREDICTED NEG	FALSE NEG	TRUE NEG

PRECISION	RECALL
TRUE POS / (TRUE POS + FALSE POS)	TRUE POS / (TRUE POS + FALSE NEG)
TRUE POS / TOTAL PREDICTED POS	TRUE POS / TOTAL ACTUAL POS

F1 = 2 x (P x R) / (P + R)

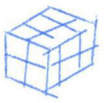
PMMS

NUMPY

ARRAYS

1D: [1 | 2 | 3]

2D: AXIS = 1 / AXIS = 0



3D

CREATING ARRAYS

a = NP.ARRAY([1,2,3])

b = NP.ARRAY([(1.5,2,3), (4,5,6)], dtype = float)

INITIAL PLACEHOLDERS

NP.ZEROS((3,4)) — CREATE AN ARRAY OF ZEROS

NP.ONES((2,3,4), dtype=NP.int16) — CREATE AN ARRAY OF ONES

d = NP.ARANGE(10,25,5) — CREATE AN ARRAY OF EVENLY SPACED VALUES (STEP VALUE)

NP.LINSPACE(0,2,9) — CREATE AN ARRAY OF EVENLY SPACED VALUES (NR OF SAMPLES)

e = NP.FULL((2,2),7) — CREATE A CONSTANT ARRAY

f = NP.EYE(2) — CREATE A 2x2 IDENTITY MATRIX

NP.RANDOM.RANDOM((3,2)) — CREATE AN ARRAY WITH RANDOM VAL

NP.EMPTY((3,2)) — CREATE AN EMPTY ARRAY

INSPECTING YOUR ARRAY

A.SHAPE	ARRAY DIMENSIONS
LEN(A)	LENGTH OF ARRAY
b.NDIM	NR OF ARRAY DIM
c.SIZE	NR OF ARRAY ELEMENTS
b.DTYPE	DATA TYPE OF ARRAY ELEMENTS
b.DTYPE.NAME	NAME OF DATA TYPE
b.ASTYPE(INT)	CONVERT AN ARRAY TO A DIFFERENT TYPE

ASKING FOR HELP

NP.INFO(NP.NDARRAY.DTYPE)

ARRAY MATHS MATCHES

G = A - B / NP.SUBTRACT	SUBTRACTION
G = B + A / NP.ADD(B,A)	ADDITION
A / B / NP.DIVIDE(A,B)	DIVISION
A * B / NP.MULTIPLY(A,B)	MULTIPLICATION
NP.EXP(B)	EXPONENTIATION
NP.SQRT(B)	SQUARE ROOT
NP.SIN(A)	SINUS
NP.COS(B)	COSINUS
NP.LOG(A) / NP.LOG2(A)	LOG / LOG2
E.DOT(F)	DOT PRODUCT

SORTING ARRAYS

A.SORT() — SORT AN ARRAY

C.SORT(AXIS=0)

COMPARISON

A == B → [FALSE, TRUE, TRUE] — ELEMENT-COMP

A < 2 → [FALSE, TRUE, FALSE] — ELEMENT-COMP

NP.ARRAY_EQUAL(a,b) — ARRAY-COMP

AGGREGATE FUNCTIONS

A.SUM()	ARRAY-WISE SUM
A.MIN()	ARRAY-WISE MIN. VALUE
B.MAX(AXIS=0)	MAX VALUE OF ARRY ROW
B.CUMSUM(AXIS=1)	CUM SUM OF THE ELEMENTS
A.MEAN()	MEAN
B.MEDIAN	MEDIAN
A.CORRCOEF()	CORR COEFF
NP.STD(B)	STANDARD DEV.

SUBSETTING, SLICING, INDEXING

a[2] → 3

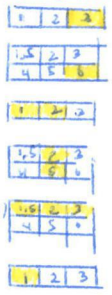
b[1,2] → 6.0

a[0:2] → [1,2]

b[0:2,1] → [2.,5.]

b[:1] → [[1.5,2,3]]

A[A[2]]



ARRAY MANIPULATION

A.T / A = NP.TRANSPOSE(B)	PERMUTE ARRAY DIM
B.RAVEL()	FLATTEN ARRAY
G.RESHAPE(3,-2)	RESHAPE
B.RESIZE((2,6))	RETURN NEW ARRAY W/ SHAPE (2,6)
NP.APPEND(H,6)	
NP.INSERT(A,1,5)	
NP.DELETE(A,[1])	
NP.CONCATENATE	
NP.VSTACK((A,B))	
NP.R_[0,F]	
NP.HSTACK((E,F))	
NP.COLUMN_STACK((A,D))	
NP.C_[A,E]	
NP.HSPLIT(A,3)	ERROR(NP.ANY(L)>10)
NP.VSPLIT(C,2)	NP.ALL

*USE INFO

NONZERO: L[NP.NONZERO(L)]

2D ARRAY LOOP LOGIC

```
ROW_LEN = L.SHAPE[0]
COL_LEN = L.SHAPE[1]
FOR X IN RANGE(ROW_LEN):
    FOR Y IN RANGE(COL_LEN):
        IF NP.ISNAN(L[X][Y]):
            L[X][Y] = NP.NANMEDIAN(L, AXIS=0)[Y]
```

MORNING COLUMNS

CP / P.SUM(AXIS=0)))

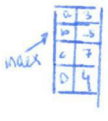
CREATE 3D FROM 2*2D

a = NP.ARRAY((1,2,3))

b = NP.ARRAY((1,2,3)) [:, NP.NEWAXIS]

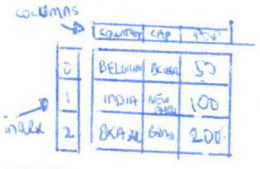
PANDAS

SERIES



s = PD.SERIES([3,-5,7,4], index=['a','b','c','d'])

DATAFRAME



data = {'country': ['Belgium', 'India', 'Brazil'], 'capital': ['Brussels', 'ND', 'Brasilia'], 'population': ['50', '100', '200']}

df = pd.DataFrame(data, columns=['country', 'capital', 'population'])

READING

PD.READ_CSV('file.csv', header=None, nrows=)

PD.READ_EXCEL()

.SORT_INDEX()

SELECTION

s['b'] → -5

df[[1:]]

	COUNTRY	CAP	POP
1	INDIA	NEW DELHI	100
2	BRAZIL	BRASILIA	200

BY POSITION

df.iloc[[0], [0]] → 'Belgium'

df.iat[[0], [0]] → 'Belgium'

BY LABEL

df.loc[[0], ['country']] → 'Belgium'

df.at[[0], ['country']] → 'Belgium'

BY LABEL/POSITION

df.ix[2]

COUNTRY	BRAZIL
CAP	BRASILIA
POPULATION	200

df.ix[:, 'capital']

0 Brussels
1 New Delhi
2 Brasilia

df.ix[1, 'capital'] → 'New Delhi'

BY BOOLEAN INDEXING

s[~(s>1)] — SERIES S WHERE VALUE IS NOT >1

s[(s<-1) | (s>2)] — S WHERE VALUE IS <-1 OR >2

df[df['population']>50] — USE FILTER TO ADJUST DATAFRAME

SETTING

s['a'] = 6

DROPPING

s.drop(['a','c']) — DROP VALUES FROM ROWS (AXIS=0)

df.drop('country', axis=1) — DROP VALUES FROM COLUMNS (AXIS=1)

SORT AND RANK

df.sort_index() — SORT BY LABELS ALONG AN AXIS

df.sort_values(by='country') — SORT BY THE VALUES ALONG IT

df.rank() — ASSIGN RANK TO ENTRIES (by=['a','y'], asc=[False,True])

RETRIEVING INFORMATION

df.shape	(ROWS, COLUMNS)
df.index	DESCRIBE INDEX
df.columns	DESCRIBE DATAFRAME COL
df.info()	INFO ON DATAFRAME
df.count()	NUMBER OF NON-NA VALS
df.sum()	SUM OF VALS
df.cumsum()	CUM SUM OF VALS
df.min()/df.max()	MIN/MAX VALUES
df.idxmin()/df.idxmax()	MIN/MAX INDEX VALUE
df.describe()	SUMMARY STATISTICS
df.mean()/df.median()	MEAN/MEDIAN OF VALUES (AXIS=1 → SAME PER STUDENT)

APPLYING FUNCTIONS

f = lambda x: x*2

df.apply(f) — APPLY FUNCTION

df.applymap(f) — APPLY FUNCTION ELEMENT-WISE

ASKING FOR HELP

help(pd.Series.loc)

DATA ALIGNMENT

s3 = pd.Series([7,-2,3], index=['a','c','d'])

s + s3

a	10.0
b	NaN
c	5.0
d	7.0

* indices don't overlap
THERE'S NO 'B'

FILL METHODS

s.add(s3, fill_value=0)

a	10.0
b	-5.0
c	5.0
d	7.0

s.sub(s3, fill_value=2)

s.div(s3, fill_value=4)

s.mul(s3, fill_value=3)

df.fillna(0, inplace=True)

2 ITEM CORR

c = df[[~ALL OF THAT ITEM]].corr()

c = c[c[]==0]

2y = c.max(c).idxmax()

bottom = c.loc[2y].idxmax()

Utility10 = [2y, bottom]

zdf = (df - df.mean(axis=0)) / df.std()

e = (df['...'] < 5,5) | (....)

OR

FILTER c = c[c[:] != 1]

to perc (...).mean() *100

precisie/recall

pr = hoeveel correct in voorsp. C = $\frac{TP}{TP+FP}$

rc = hoeveel daadwerk C correct = $\frac{TP}{TP+FN}$

acc = hoeveel correct totaal = $\frac{TP+TN}{totaal}$

acc = 0.9 ziek = 0,1

pziek = $\frac{0.1 \cdot 0.9}{(0.1 \cdot 0.9)+(0.9 \cdot 0.1)}$ ⇒

pnziek = $\frac{0.9 \cdot 0.9}{(0.9 \cdot 0.9)+(0.1 \cdot 0.1)}$

%ls -lh = directorysize %paste/cpaste

%%rm file = verwijder %%run = extern

%time %timeit %prun = resources

%%variabele? = type pr[-]/out[2]/-2

%history -n 1-4

pwd/ls/cd/mkdir/mv

% = modulo/remainder

// = floordivision (cuts low round)

NUMPY

D1.shape(values)

D2.shape(row, column)

- shape = vorm → shape[0]/[1] = rijen/kolommen
- size = grootte/aantal values
- reshape( ) → platmaken 2D = 1 waarde met de size of .flatten()
- arange(st., st, int) alleen necessary

veranderen = 1D (-/+ etc) vb X1 - X1.mean

np.sqrt( ) = √

X1.std = stand.

Boolean mask vb. X1[X1 % 2 == 0] → geeft Fals/True

X1 < 3 = False en Trues

np.sum( ) geeft aantal Trues

np.nonzero(): geeft locatie non zero → array[np.nonzero(a)] = de waardes

broadcasting

① matching dimenties kleinste wordt groter

② shape van array met shape 1 wordt gestretched

③ geen shape is 1? → kannie

np.all / np.any > geeft True of False

np.sort(a, axis=1) = sort de rijen op grootte

vb. toets a = np.arange(1,4) b = np.arange(1,4)[:, np.newaxis]

a = [1,2,3] b = [[1][2][3]]

① a = [[1,2,3]]

② b = [1,1,1] [2,2,2] [3,3,3]

a+b = [[2,3,4] [3,4,5] [4,5,6]]

NORMALISEREN opvall.

vb. 2/12, 4/12, 6/12 + = p/p.sum(axis=0)

12 1

RMSE = $\sqrt{\frac{\Sigma \hat{y} - y}{T}}$ ŷ = echt, y = gem aantal, T = totaal waarden

Pandas

pd.read_csv('file') → bekijk opties voor naam indexers.

df.sort_index(asc)

voorwaarde (df[df[lijst k]]) = </> etc.

↳ .sum(). sum() = geeft hoeveelheid

{fillna(value) dropna(zie condities) → NaN} DTI

nieuwe kolom

df[kniaam] vb. DF[[lijst k]].mean(1) geeft gem rij van de kolommen

idxmax return index van max val. OIA

Vb. len((df[k] vw) | (df['k'] vw)) / len(df) = % → & doe per stuk

df.sort_values[lijst, bool lijst]

corr uit DT

C = df[[kol]].corr()

C = C[C! = 1]; voorwaarde

rij = C.idxmax()

kol = C.loc[rij].idxmax()

coord = [r, k]

pd serie

pd Df

df[kol].value_counts(): aantal v. in kolom

z.normalize = X - X mean / X.std()

voor kolom Df['k'] / voor index Df.index / voor values Df.values

- str.len()
- str.lower()
- str.strip

DF[~DF[k].str.contains('w')] → alle zonder dat stukje string

Df[k] = [re.sub(r'[...]', '', str(i)) for i in df[k]]

alleen [.-|...|] = woorden

[^A-Za-z0-9] + = alleen letters & cyfers

groupby?

df.groupby(kolom)[k]

.max/sum etc

↳ geeft max van kl

pd.merge

→ corrects index

on = zelfde kolom

left_on/right_on

np.concatenate

pd.concat