Day 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Month 1 2 3 4 5 6 7 8 9 10 11 12
Subject

Sensitivity, recall, hit rate (TPR)

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Precision (PPV)

$$PPV = \frac{TN}{N} = \frac{TN}{TN + FP}$$

Accuracy (ACC)

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$$

F1 score

$$F1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$$

Nuttige functies numpy

np.concatenate
.vstack/hstack
.r_
.where() returns indices
.

select <=> pandas select

pandas pivot crosstab

margin = total
agg = np.[mean, sum, median]

np.array [[row, col]]

np.tile
np.reshape
np.repeat

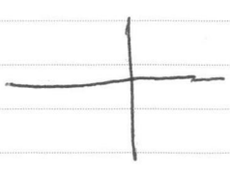
from numpy import doc
np.lookfor('query')

FP = ~~1~~ 1/100

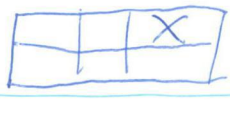
$$\frac{TN}{NN + FP}$$

~~99~~ 0.98
99

h



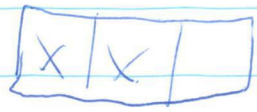
NP. array



NP
- .shape
- .size
- .add(a,b)
- .arange(0,100,10)
- a +-*/ b
- .reshape (keep)
- .resize (delete)

a[0,2]

a[0:2]



a[a<2]

.sort()

from numpy import doc

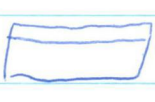
? np.doc

np.lookfor

Pandas

df = pd.Dataframe([[1,2,3],
[4,5,6],
[7,8,9]], index=[1,2,3],
columns=[a,b,c])

PD
- .query('a >= 100')
- .rename('val', 'value')

- .melt(df)  to 
- ~~pivot(df)~~
- df.pivot(col=, val=)  to 

	a	b	c
1	1	2	3
2	4	5	6
3	7	8	9

- Concat([df1, df2]) axis=0 rows
axis=1 columns

a+b = error?
↓
broadcasting rule

- .drop (columns=)
- .dropna()
- .sort_values
- .sort_index
- .head
- .tail
- .fillna(x)
- .nlargest(n, 'value')
- .nsmallest(n, 'value')

df[df.length > 7]

df[df['length'] > 7]

- .value_counts()
- .unique()
- .count

sum, mean, min, max, (med), std

- .plt.hist
- .read_html(link)
- .groupby(by="col")

CHEAT SHEET #1

NP (y, x) , np.zeros() - np.ones() - np.full() -> can shape
↳ rij ↳ kolom
np.arange() -> cannot shape, is linear & np.linspace() (evenly spaced between given values).
else: np.random.random() - np.random.normal() - np.random.randint()
=> creating arrays
x2[0,0] = 12 -> changing value of that index to 12 - x2.astype('float64') -> got to assign to new variable - x[start:stop:step] - x[::-1] -> all elements reversed - x[:3, ::2] -> all rows, every other column
accessing column: x[:, 0] -> first column of x - accessing row: x[0, :] -> first row of x. - if you modify a subarray, the original is changed -> use .copy
combination of np.arange().reshape() -> can get a multidimensional something
np.concatenate(array1, array2, axis=1) : joining two or more arrays
np.hstack() & np.vstack() ↳ add that array according to columns
np mathematics : x1 - x2, x1 + x2, x1 / x2, x1 * x2, np.sqrt(), np.log(), -> must be the same shape, np.floor_divide(), %
Pythons built-in absolute value abs() -> np.absolute()
np.reduce() -> applies a given operation to elem of an array until only a single result remains., using np.accumulate() stores all intermediate results of the operation = np.multiply.outer(x,x) (x = np.arange(1,6)) -> creates multiplication table.
x3.min(axis=0) -> min value per each column. - np.sum/prod/mean/std/max/min/var/argmin/argmax/median/any/all have nansafe versions, just put nan in front ex nansum ↳ finds index
Broadcasting: array-array is element by element basis if arrays are same size
(m,n) + (1,n) -> copies m times to (m,n) fill use [:, np.newaxis]
" (m,1) -> copies n times to (m,n) np.sum((inches > 1) & (inches < 1))
↳ how many entries
np.sum(x<6) -> counts how many are bigger than 6
np.sum(x[x<6]) -> counts values in x, smaller than 6 [:, np.newaxis]
fancy indexing: shape of result reflects shape of indexing array ipv. shape of array being indexed - np.sort(x) & np.argsort(x), np.partition(x,3) -> smallest 3 values to the left LET OP AXIS! # 2 [] als je lists in maakt meegeeft

PD df.index = df.values - df.columns = list(series.items()) - df['new column'] = ..., slicing by explicit index series["a":"c"] -> final index = included <-> implicit index - data.loc[1] -> finds explicit index <-> data.iloc[1] finds implicit index - add new column: data['density'] = data['pop'] / data['area']
swap rows & columns: df.T - passing a single index to df.values gives you a row
passing a single index to df passes a column, use iloc to slice a df. - the ix indexer allows you to do both! data.loc[data['density']>100, ['area','density']] -> columns you want in resulting dataframe. 'Indexing refers to columns & slicing refers to rows
A.add(B, fill_value=0) -> makes sure adding doesn't result in NaN's use A.stack.mean() to get the overall mean, row use: A+B column use A.add(B, axis=0)
isnull(), dropna(), fillna(), notnull()
↳ axis="columns drops all columns containing null values, how='all' -> drops where whole row or column is nan, unstack() multiindex -> regular and vice versa
data.sort_index() - pd.concat([...], axis=..., 'ignore_index=True/False', join='inner'], .append() pd.merge(on=.., left_on="employee", right_on="name"), df.join -> merge that defaults to joining on indices.
df.set_index('F'), pd.merge(df1, df3, left_index=True, right_on='name'), merging df's with not much in common? pd.merge(df1, df2, how="outer") also left and right, merge with overlapping column names) pd.merge(df1, df2, on="name", suffixes=["_L","_R"]), pd.read_csv - df.value_counts
lambda x: x*10, isnull().any() - check if any value is missing, np.argsort() / sort
df.sort_index(by='F'...) - df.plot(kind='barh') - df.groupby('opleiding')['aantal_inwoners'].sum() -> how much
pd.to_datetime()
df[df.place.str.contains('amsterdam')
precision: how many of pred of class C were correct TP/(TP+FP) - recall: how many instances of class C predicted to be C TP/(TP+FN), accuracy: how many off all values were correct TP+TN/(TP+TN+FP+FN)
sort_values(by='F') regex: str.replace('[^a-z0-9]',' ')/[^a-zA-Z0-9]+
multi-index: pop[:9, 2000] str.replace("all|in|between","")

	pos	neg
pos	TP	FP
neg	FN	TN

TP: hans opdicts * accuracy
FP: (1 - hans opdicts) * (1 - accuracy)

from numpy import dot
np.lookfor('')

gsd(ic, g'/.k)

TP/TP+FP

Precision: how many of the predictions for class c were correct (Hoeveel van het totaal van ziek/niet ziekte heeft en pos/negatief getest, daadwerkelijk ziek/niet ziekte)

Recall: how many instances of class c predicted tobe c (TP/TP+FN)
hoeveel van de True class zijn 'opgepikt' door de test.

Accuracy: how many of all predictions were corred.

~~Hoeveel van het totaal van ziekte/niet ziekte heeft en pos/neg uitslag~~

Belangrijk bij skewed data:
anders krijg je hoge acc bij zeggen dat iemand gezond leeft helpt niet dan de kans heel klein is

zicht: P = 10% Acc = 90%

V.b.

	had	nodis	
PT	90 TP	90 FP	180
NT	10 FN	810 TN	820
	100	900	1000

90 / 90+90 = 0.5

810 / 820 = 0,9878

10% (ziekte)

precisie (kans, accuracy):
TP = kans * accuracy
FP = (1 - kans) * (1 - accuracy) FN = (1 - acc) · kans
return TP / (TP + FP)

axis=0 (rowwise) ↓ axis=1 (kolomwise) →

	A	B	
0	↓		axis=1 →
1			
2	axis=0		

df = df[df != 1] -> selecteert alle 1 eruit en zet die op NaN | df.fillna(0, inplace = True) -> NaN opvullen met 0's en opslaan

df.max() -> geeft hoogste waarde per kolom in df (-> axis=0 default)

pd.Value_counts -> Counts unique values

df.idxmax() -> geeft index id van hoogste waarde in kolom (of rows afh. v. axis)

df[["col1", "col2", etc]] -> selectie op kolommen

V.b. (van kolom of van rij) df[["col1", "col2", etc]].mean(axis=1)

... = df[(df["col1"] < 5.) | (df["col2"] > 6)] -> selectie op kolommen met voorwaarde (| = een, & = en)

from numy import doc
np.Lookfor(")
Lookfor

matrix normaliseren: P / P.Sum(axis=0)

P = [[1
[1
[... 1

diagonal matrix: P[0,1,2...],[0,1,2...]] kan ook gevuld met D.Shape

Loop door kolommen (matrix): for x in range(len(matrix.shape[1])): (col, # kolommen)
L[:, x][getal] = ...

np.isnan(input check op nan)

Flatten: p.flatten() ndarray naar 1D array

np.array[... ...] = 2 of "_" etc. geeft boolean terug (kijk x/tekst, comma etc.

L[np.nonzero(L)] -> Selectie op de numpy array (net als pandas)

L = [[1, 3, 4, 5],
[2, 6, 1, 8]]
↓ tweede
L[:, 1] -> eerste kolom
L[:, 0] -> best eerste kolom

a = np.array(1, 4)
b = np.array(0, 1, 4)[:, np.newaxis]

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

% ds dh directory, hidden files
% run ... Verwijder ...
% time % timeit % prun

NP | X[start, stop, step] (-1 for reverse)

Calc.

mean/mode/median
add/substract/mod/power
sum/min/max (arg/nan)
any/all/percentile
sort/partitioning (arg for k-nearest)

reorder

random/copy
arange
reshape/concatenate
shape
(V/H)(stack/split)
transpose
roll
linspace
full

test

unique
reduce/accumulate
outer

Doc import doc
Loop for " "

Pd (Series / DataFrame)

read_csv
crosstab/pivottable
dropna
value_counts
sort_index / sort_values
mean/mode/median
corr
groupby. func_
nlargest

STR

replace
lower
strip
contains
count
len

T

ix (index int en lettres mix)

add

pd.Series.str.IETS ?

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

np.zeros ((3,4))
np.ones ((2,2))
np.full ((2,2),7)

a.shape = array of dimensions
len(a) = length of array
b.ndim = num array dimensions
e.size = num array elements

▽ horizontaal = colums : axis = 1
• verticaal = Rijen : axis = 0

a - b → np.substract (a,b)
b + a → np.add (b,a)
a / b → np.divide (a,b)
a * b → np.multiply (a,b)

np.exp (b)
np.sqrt (b)

a[::-1] > reversed array a

a[a<2] > select elements from a less than 2

a.sum ()
a.min ()
b.max (axis = 0)
↳ max of ROW
a.mean ()
np.std (b)

i.transpose () → Rijen en kolommen verswisselen

a.sort () → sort an array
c.sort (axis=0) → sort the elements of an array's axis
np.sort (c, axis=0) → sort each column of c

g.reshape ((3,3))
np.append (h,g)
np.delete (a,[1])
np.concatenate ((x,y), axis=0)
↳ samenvoegen arrays

np.abs ()
↳ geen negatieve

np.mean (x) / np.mean (x[0])

np.mean (x, axis = 1)
↳ gem.

a.sum ()

.mean (1)
= gerekend over kolommen

np.intersect1d (a,b)
> get common values

np.setdiff1d (a,b)
> from a remove all b

np.where (a == b)
> positions where a & b match

P is een multidimensionale array
P.flatten () → nu 1 dim

np.all → all elements along a given axis must be true
np.any → any

L[np.nonzero (L)] → alle elementen die niet nul zijn uit L

x.namen.value_counts ()
naam1 370
naam2 198

df["Ass_mean"] = df[df.colums[0:6]].mean (axis=1)

b = np.arange (1,4)[:, np.newaxis]
array ([1],
[2],
[3],) ipv. (1,2,3))

a[:,[1,0,2]]
kolommen nieuwe volgorde

x.min (axis = 0)
↳ min / column

x.plot (kind = "Barh");

np.sum (P<n)
= hoeveel waarden kleiner dan P

np.isnan
> bool, Returned True / False

all = df[df.colums[0:6]]

df.iloc [2,1]
↳ selecting specific data

df.loc [df['a'] > 10]
↳ selecting raws meeting logical condition

greater_than_g = all [all > g].count ().sum ()

df.sort_values (['Name', 'Age'], ascending = [False, True]

df['W'].value_counts ()
↳ count number of raws

df["New_Column"] = (a.4 * 2) + (0.6 * 1)

df.sort_values ['Name', 'Age']
df.sort_index ()
df.drop (columns = ['Length']
pd.concatenate ([df1, df2], axis = 1)
↳ Reshaping
df3 = pd.merge (df1, df2)
↳ combine datasets

data [(data > 0.3) & (data < 0.7)]

data [~ data.isin (data2)]
↳ data die niet ook in data2 staat

df.groupby (['Animal'].mean)

df.fillna (0)
df.dropna (subset = ["Age"], inplace = True)

df.values () > Raw data
df.transpose () > Rijen x kolommen omdraaien

len (df) > num of ROWS

str.lower() / str.Replace ()

X = X.drop (6, axis = 1) → kolom 6 weghaler

KVR = KVR[~ KVR.Party.str.contains ("vragen", na = False)]

KVR.Party.value_counts().sort_index ()

np.array([1,2,3])

a.ndim → Dimensions

np.info

a[2] = 1 2 3 2e item

a[1,2] = 1 2 3 / 4 5 6
↑ Rij ↑ Kolom

B[0:2, 1] = 1 2 3 / 4 5 6
↑ Rij ↑ Kolom

a[0:2] = 1 2 3
0 T 2

a[::-1] = Reverse

np.arange(3) = np.array[0,1,2]
Start, Stop, step

np.zeros((3,2)) = [1,2] [3,4] [5,6]
↑ Rij ↑ Kolom

a[:,-1] = 1 2 / 3 1
↑ Rij ↑ Kolom

np.zeros = Van array

np.mean

np.transpose = [[1,2],[3,4]] = [[1,3],[2,4]]

np.ndenumerate = [[1,2],[3,4]]
(0,0) 1 (1,1) 4
↑ POS ↑ nR

pd.read_csv(link, sep=',', nan)

kar.value_counts = 2.0 ⇒ 2x etc

Van = kar.loc[kar["naam"].min

Pd.Crosstab = [Rij, Kolom]

kar.dropna(subset=["kar"])

kar["naam"] = kar["naam"].str.lower()

.str.replace('W','').str.replace('1','')

R = [kar.str.contains("p")]

kar.loc[kar("p") > 9]

Van = kar['p'].str.count('1?)

Van2 = Van.where(Van <= 50)

mean() median() mode()[0]

Van.corr(kar["p"].str.len()

pd.Pivot_table(df, values, index

np.sum, margin

sort_values["A","B"] asc = [T, F]

.between(-1, 1)

idxmax() = id hoogste kolom

Pandas

- df.sort_values([["col1", "col2"]], ascending=[F,T])
- df.fillna('waardoor?', inPlace=True)
- pd.read_csv('naam', index_col='naam', names=[])
- df[] > of < 'waarde'
- pd.concat([df1, df2]) → append rows
- pd.concat([df1, df2], axis=1) → append columns
- df.rename(columns={'y': 'year'})
- df.dropna() → drop rows where at least 1 element is missing
- df.dropna(how='all') → where all elements are missing
- df.loc[:, 'x2':'x4'] → select all columns between x2 and x4 incl
- df.drop(columns=['a', 'b'])
- pd.isnull()
- pd.merge(adf, bdf, how='left'/'right'/'inner'/'outer', on='x1')
- df['w'].nunique() → # of distinct values in column
- pd.notnull() / notna()
- .str.lower()
- .str.contains('string')
- series.str.startswith('string')
- series.str.replace('iets', 'door_dit')
- Series.value_counts(sort=True, ascending=, dropna=)
- df.drop_duplicates([columns], inPlace=)
- df.iloc[:, [1,2,5]] → select cols in positions 1, 2, 5

Actual \ Predicted	N	P
Negative	TN	FP
positive	FN	TP

$$accuracy = \frac{TP+TN}{total}$$

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

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

lambda x: x + 10

$$= (p \cdot acc) / ((p \cdot acc) + (1-p) \cdot (1-acc)) = \text{precisie ziet}$$

$$(1-p) \cdot acc) / ((p \cdot (1-acc)) + (1-p) \cdot acc)) = \text{precisie niet ziek}$$

Numpy

- np.intersect1d(a, b)
- np.setdiff1d(a, b) verwijder uit arrA wat in B zit.
- np.arange(start, stop, step)
- np.isnan(array)
- np.nanmedian(array, axis=)
- np.nanmean(array, axis)
- arr.flatten() → maakt van MA 1D array
- np.nonzero(arr)
- np.sort(arr, axis=)
- np.sqrt(x)
- np.multiply(x1, x2)
- np.vstack([a, b])
- np.hstack([a, b])
- np.argwhere(x > 1) → geeft indices van values die voldoen
- np.random.choice(5, 3) → kiest 3 dingen uit np.arange(5)
- np.where(condition)
- np.unique(arr, return_index=, return_counts, axis)
- np.linspace(0, 2, 9)
- arr = arr[~np.isnan(arr)]

plt.bar(x, height, color, label)
plt.xlabel('string', fontweight, fontsize)
plt.xticks(np.arange(x), rotation)
plt.barh(x='minister')

Regex

- \d → matches any decimal digit
- \D → matches any char that isn't dec digit
- \w → matches any word char
- \W → matches any non-word char
- \s → matches whitespace chars
- 'Lenght$' → matches strings ending with 'length'
- '^Sepal' → matches strings beginning with 'Sepal'
- ? → 0 or 1 reps
- + → 1 or more reps
- * → 0 or more reps

Numpy	Pandas	Regex
np.setdiff1d	df.columns	1 = match any
np.equal	df.value_counts	a-z = range
np.reshape	df.index	\d = digit
np.transpose	df.values	^ = not
np.argwhere	df.crosstab(df.X, df.Y)	\w = word
np.unique	df.copy	| = or
np.equal	df.dropna(subset=['x'])	* = 0 or more
np.logical	df.astype(str)	. = any char
np.where	df.contains	\D = non digid
np.shape	df.nunique	r' = regex
np.ndim	df.sort_index	\n = newline
np.size	df.loc	
np.idxmax	df.count	
np.nonzero	df.describe	
np.absolute	df.pivot_table	
	df.concat	

rd.multiIndex.from_tuples([('a', 2000), ('b', 3000)])

rijen kolom

L[:, 0]

[:, newaxis]

P / P.sum(axis=0) = normalise

((1-z) * a) / (((1-z) * a) + (z * (1-a))) = nP

(a * z) / ((a * z) + ((1-z) * (1-a))) = P a = aanwezig, z = ziek

$$\text{Precision} = \frac{TP}{TP+FP} \qquad \text{Recall} = \frac{TP}{TP+FN}$$

round(,1)

%ls
%cp ...
%time
%timeit
a=set
a.iTab

Recall = TP/(TP/FN)
precision = TP/(TP/FP)
acc = -- % reliable

	has dis	no dis
pos tst	TP	FP
neg ts	FN	TN

95	495	590
5	9905	9910
100	9900	10000

0.01 x 0.95 / (0.01 x 0.95 + 0.99 x 0.05) = 0.16

.isin()
.Value_counts()
.drop() .dropna() => how='all' = drops rows/cols = all nan thresh=3 (=alleen min met 3)
.unique() subset=['...']
.sort_index() .sort_values() (by=[-,-], asc=[T,F])

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

.to_datetime

from collections imp Counter
%matplotlib inline

.inplace = True
.idxmax
.copy

.merge
.melt
.query()
.unique()
.clip() bv. ondersteboven 10

print("%s - %s" % (per, mis))

plt.plot(kind='bar'/'barh' plt.'hist' .scatter linestyle='steps')
plt.xlabel('...')
plt.legend(['...'])
plt.title('...')
plt.show()

.rank

.Str
.len()
.contains('')
.replace
.Count
.Strip() lower upper strip space

pd.MultiIndex?
pd.merge() -> on='...' -> how='inner'/outer

.loc [:,:]
.iloc [:,:2]
.ix [:,'pop']
.ascending

.nlagest
.nsmallest(n, value)
.filter

aggfunc=sum mean

.Stack()
.unstack() level=0/1
.flatten()

Set((...) & | ^ (...))

.concat(2 df's samenvoegen), ignore_index=True, join='inner'/outer
.append()

.index[0] -> na sort_val. => eerste
.groupby('year','sex')['count'] ->by=/level=
.aggregate('first').unstack()
.index_col = ...
.reset_index()
.fillna(0 of 1) -> method='ffill' -> vult val
'bfill' -> vult val
.mean() etc. maar 0=col 1=ry

apply()

:2 = 2 stappen

isnull()
notnull()

cumsum min max prod

Nieuwe col = df['...'] = ...
.sort_values(['...','...'], accend.=[...,...])

// floordiv()
/ div()
** pow()
% mod()

.corr()
.dtype
.describe
.items()

.any()
.all()
.head() .tail()
.shape()

.T .transpose()
.add() fill_value=0
.mean() level='year'

---

np.array([...])
np.arange(10) -> start, stop, step

<=
>=
==
!=

.zeros .ones
.full((3,5), 3.14) empty
.linspace() -> start, stop, hoeveel
dus 1, 2, 3 = 1, 1.5, 2

.eye(3) = 1 0 0 / 0 1 0 / 0 0 1 .eye(3,k=1) = 0 1 0 / 0 0 1 / 0 0 0

.add.at(x, i, 1)

bogosort
while np.any(x>1):
np.random.shuffle(x)

.sort()
.argsort()
.lexsort((2,0)) dus 2 dan 0
np.partition()
argpartition
Sorted()
↳ gean np!

np.random.randint(low=, high=, size=)
.random((3,3)) 2d array
.normal
.rand

.ndim
.shape
.size
.dtype

array
row=[0,1,2]
col=[2,1,3]
row[:,newax] * col
000
213
426

.setdiff1d
.count_nonzero
.logical_and
.unique
.reshape()
.asarray .astype
.isnan
.np.newaxis -> row/col

5: 5,6,7
:5 0,1,2,3,4
::2 0,2,4
1::2 1,3,5
::-1 => reversed 3,2,1,0
5::-2 => 5,3,1

reverse 2d:
x[::-1,::-1]

first row/col
x2[:,0] col
x2[0,:] row

np.newaxis

& | ^ ~ => niet => np.bitwise_not

Broadcasting: -> meeter zelfde lengte nodig
np.arange(3) + 5
0 1 2 -> 5 6 7
np.ones((3,3)) + np.arange(3)
1 1 1 + 0 1 2 => 1 2 3 / 1 2 3 / 1 2 3
np.arange(3).reshape((3,1)) + np.arange(3)
0 1 2 -> 0 + 0 1 2 / 1 + 0 1 2 / 2 + 0 1 2 = 0 1 2 / 1 2 3 / 2 3 4

.hstack
.Vstack
.concatenate(axis=0/1)
dstack = 3th axis?
.vsplit(..., [2]) -> in 2 delen
.hsplit

.absolute/abs
.sin/cos/tan
.exp .log
.exp2 .log2
.power(3,x) .log10
ook inmp.add
np.divide
np.power
np.mod
etc.

.reduce()
.accumulate()
.outer(x,x)

.Cumsum
.corrcoef

from scipy.imp.stats .mode()
modus=

.Sum() .min() .max()
.mean .prod
.nanmean .std .Var
↳ variance

.argmin, max
.median
.percentile

.copy

insert
.delete

PRESICION & RECALL

- EVALUATING (reliability: does not account for errors) Good
  - CLASSIFICATION: ACCURACY, PRECISION, RECALL, F1
  - REGRESSION: RMSE

ACCURACY: How many of all predictions correct

RECALL: How many instances of class C predicted to be C $TP/(TP+FN)$

PRECISION: How many predictions of class C are correct $TP/(TP+FP)$

NUMPY

CREATE: • NP.ARANGE • NP.LINSPACE • NP.FULL((2,2), 7) (constant) • NP.RANDOM.RANDOM((2,2)) • NP.ARRAY([[]])

INSPECT: • A.SHAPE • LEN(A) • A.NDIM • A.SIZE

ARYTH: NP.SUBTRACT • NP.ADD • NP.DIVIDE • NP.MULTIPLY • NP.EXP (sort, log, sin, cos)

• A.SUM() • A.MIN • A.MAX(axis=1) • A.MEAN() • A.CORRCOEF • NP.STD(A)

SLICES: • A[1,2]  • A[[1,0],[0,1,2]] → (1,0),(0,1),(1,2)

MANIPUL: • A.T • A.RAVEL (FLATTEN!) • A.RESHAPE • A.SORT(axis=0) • NP.ARGWHERE

%matplotlib inline

% MAGIC → explains magic commands.

W = !LS → LEN(W) is based on terminal

- Any CHAR consider in word, digit, whitespace
- \w is
- \W is Not word, Not digit, not
- [ABC] Any of ABC
- [^ABC] Not A, B or C
- [A-G] CHAR BETWEEN A-G
- * + ? 0 or more, 1 or more, 0 or 1
- a{2,3} 2 of mean a's

PANDAS

DF = PD.DATAFRAME([[...]], INDEX=[1,2,3], COLUMNS=['A','B','C'])

• DF.SORT_VALUES • DF.SORT_INDEX • DF.RENAME • DF.RESET_INDEX

• DF.ILOC[10:20] (rows; inclusive not exclusive) • DF.LOC[:,'x2':'x4'] • DF.LOC[df.a > 10, ['a','c']]

• DF.W.VALUE_COUNTS • LEN(DF) • DF.W.NUNIQUE • DF.DROPNA • DF.FILLNA(vals)

MISC • DF.GROUPBY(by='cd').SUM() (by) • PD.PIVOT_TABLE • DF.IDXMAX (id with max val)

MERGE • PD.MERGE(a, b, HOW='LEFT', ON='x1')

• PD.PLOT(title='') • DF.PLOT.HIST() • DF.PLOT.BARH()

• DF = DF[DF.NOTNULL()] • DF.A.STR.REPLACE(regex, '')

Precision : $\frac{tp}{tp + fp}$

Recall : $\frac{tp}{tp + fn}$

Accuracy : $\frac{t}{tF}$

Numpy array indexing.

X2 [0, 0]

↑ ↑

Row column

[start : stop : step]

Slicing

One-dimensional

X [:5] eerste vijf

X [5:] alles na vijf

X [::2] om de twee

X [::-1] omgedraaid.

Multi-dimensional

X [:2, :3] stop bij de tweede rij
stop bij de derde kolom

X [:3, ::2] stop bij de derde rij
om de kolom.

X [0, :] rij 0, pak alle kolommen

Filtering pandas

a	0,25
b	0,50
c	0,75

= data [(data > 0,3) & (data < 0,8)]
data.values

pd.read_csv('file', compression='gzip', sep='/t', delimiter='\t', header=None, index_col(0)

.fillna(0) → slaat niet op

df.iloc[:, -4:-1].sum(axis=1).mean(axis=1)

df.sort_index(inplace=True)

inplace=True | = of teveen

(df[[ASS1, ASS2, ASS]] > 9.sum().sum()

perc = ((df["ass"] < 5.5) | (df["ass"] < 5.5)).mean() * 100

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

df.sort_values(['deeltoets_mean', 'names'], ascending=[False, True])

c = df[[ASS1, ...]].corr()

c = c[c != 1]

rij = c.max().idxmax()

kolom = c.loc[rij].idxmax()

vlo = {rij, kolom}

df.groupby('columns')['income'].mean()

df.value_counts().index.

pd.pivot_table(df, columns='sex', index='Name', aggfunc={'count': np.sum}, values='count')

- .iloc[0] impliciet
- .Loc['a'] → expliciet
- .idxmax()
- .isin()

numpy:

def normalise(p):

return (p / (p.sum(axis=0)))

d = np.arange(S.shape[0]) ← pixels

S[d, d], np.diagonal(S)

np.Reshape(L, L.shape[0], L.shape[1])

np.Transpose(L)

np.where( )

np.flatten() → 1d

np.sum(p < n) ← van p, hoeveel daarvan strict kleiner zijn dan n

L[:, c]

L[np.nonzero(L)]

np.arange(1,4)

np.arange(1,4)[:, newaxis]

ACC = Tp/(Tp + TN + FP + FN)

precision = Tp / Tp + Fp

RECALL = Tp / Tp + FN

np.sqrt(np.mean((L[:, i] - L[:, 0])**2))

np.sort(L, axis=1) → sorteert per rij, van laag → hoog

np.all(np.abs(L) > 10)

cito.str.lower().str.strip().str.replace(" ", "").str.replace

regex: .str.replace(R'[^a-z\d]', '') ← match

pd.crosstab(kvr.partij, kvr.jaar)[most_common().index.array]

↳ alle specifieke teveens vallen weg

- .contains('vangen')
- .unique
- .values

np.random.randint(0, n)

- .append

np.arange(n)

np.reshape(L, i, c)

L.size

for loop

maxline = ' '

for line in fin:

if len(line) > len(maxline):

maxline = line

Last column = L[:, -1]

n = len(L[0])

L[int(n/2) - 2 : int(n/2) + 2, int(n/2) - 2 : int(n/2) + 2]

buren:

x_lim, y_lim = grid.shape

grid[(x + xx) % x_lim, (y + yy) % y_lim

for xx in (-1, 0, 1) for yy in (-1, 0, 1)

if not (xx == 0 and yy == 0)]

[:, :2][:, :2]

np.arange(0,0,step)
Array.reshape((n,m))
~np.array(L'/.N, dtype=bool)
np.transpose(array) [[1,2],[3,4]] → [[1,3],[2,4]]

np.size
np.ndim
np.astype

Aggregate functions

a.sum() sum array
a.min()
b.max(axis=0) max value per row
NP.Sort(axis)

manipulation

np.concatenate( )
np.vstack() vertical
np.r_[e,f] stack vertical rowwise
np.hstack
np.column_stack
np.hsplit(a,3)
np.vsplit(a,3) split at index

For y,x in zip(Array, Array)
np.linspace(0,2,9) → [0, 0,25, 0,5 ...]
np.full((2,2),7) → [[7,7] [7,7]]
np.eye((2,2)) → identity matrix
np.random.random((2,2)) → random values
np.ones()
np.zeros()

slicing

L[0:3,1] row 0/m 3, pak value 1
L[:] pakt alles
L[:1] 1ste row
L[L<2] alle values <2
L[[0,1],[1,0]] pakt L[0,1] en L[1,0]

broadcasting

Arange(3)+5
0 1 2 + 5 5 5 = 5 6 7

np.ones((3,3)) + Arange(3)

1 1 1 0 1 2 → 1 2 3
2 2 2 0 1 2 → 2 3 4
3 3 3 0 1 2 → 3 4 5

+ 0,1,2

0 0 1 2
1 + 0 1 2 → 1 2 3
2 0 1 2 → 2 3 4

Syntax
Index = pd.MultiIndex.from_tuples([(d,1)..])

reshape
pd.melt(df) → columns to rows
df.pivot_table() spredt rows to columns
pd.concat([df1,df2]) append rows, axis 1 for colum.

observations:
df[df.Length>7] df.drop_duplicates()
df.head(n) df.nlargest df.nsmallest()
df.filter(regex) df.loc[df["a"]>5, ["a","b"]]

summarize
df['w'].value_counts()
df['w'].nunique()
min()
max()
qvantile()
var()
apply(np.sqrt)

df.sort_values('col')
high to low: ascending = False
df.rename(columns={'y':'yea
df.reset_index()
df.drop(columns=['length'])

df.dropna()
df.fillna()

df['Area'] = df.length * df.heigh
pd.qcut()

adf[adf.x1.isin(bdf.x1)]

df.Plot.hist() / .Plot.bar() / Plot.barh
Pd.crosstab()
df.Pivot_table()

df.sort_values(by=[]) altijd

str.lower
kvr[...].str.replace
str.contains
str.len
str.count

df.div(sum, ...)

acc = (TP+TN)/total

recall = TP/(TP+FN)

Precision = TP/(TP+FP)

Pandas

`df[[kolommen]]` projectie v.d. kolommen

`df[[kolommen]] > x)` booleanmask op de projectie

`df.column.mean()` gemiddelde over kolommen

`df.column.mean(1)` gemiddelde over rijen

`df.column > x).mean()*100` berekenen percentage m.b.v booleanmask

`variantie = ((df - df.mean())^2).mean()`

`std = np.sqrt(variantie)`

`corr_matrix = df[[kolommen]].corr()` correlatie matrix van df[[kolommen]]

`corr_matrix = corr_matrix[corr_matrix != 1]` corr_matrix zonder items die met zichzelf correleren

`rij = corr_matrix.max().idxmax()` key die hoort bij de hoogste corr waarde in de rijen

`kolom = corr_matrix.loc[rij].idxmax()` " ↑ " " " " " " " kolommen

What are the max sepal_lengths for each species? `df.groupby('species')['sepal-length'].max`

Numpy

`axis = 0` → kolom | `axis = 1` → rij

Fancy indexing: `array[[rij, rij, rij], [kolom, kolom, kolom]]` 1e lijst plaats rij, 2e lijst plaats kolom

Normaliseren: `(array / (array.sum(axis=0)))` elke waarde delen door gem. van de kolom.

`np.arange(S.shape[0])` maakt een lijst van waardes gebasseerd op aantal rijen in een array

`array.shape` → (aantal rijen, aantal kolommen)

2D-array → `L[0]` = eerste rij

↳ `L[:,0]` = eerste kolom

`L[:,x][np.isnan(L[:,x])]` → checkt voor alle waardes in kolom x of deze nan zijn

`array.reshape((array.size, ))`

`np.arange(1,4)` | 1 rij, 4 kolommen

`np.arange(1,4)[:, np.newaxis]` | 1 kolom, 4 rijen

(1 op 10 ziek)

accuracy = 90%

Precision = TP/(TP+FP)

Recall = TP/(TP+FN)

	Z	NZ	
P	1 · 0,9 (TP)	9 · 0,1 (FP)	1,8
N	1 · 0,1 (FN)	9 · 0,9 (TN)	8,2
	1	9	10

precision = $\frac{tp}{tp+fp}$

recall = $\frac{tp}{tp+fn}$

F1 = $\frac{2 \cdot prec \cdot recall}{}$

accuracy = $\frac{tp+tn}{tp+fn+tn+fp}$ $prec \cdot rec$

np.random.seed(a)

np.random.randint(max, size=amt) (3,4)

np.reshape(array, (4, X)) [[[1,2,3] → X, [4,5,6]] ↓ Y]

eye(amt^2, dtype=int) = [[1,0], [0,1]]

resize(= affects og

read_csv(file, skipinitialspace, sep)
- values
- index
- loc / iloc
- series([1,2,3], index=['A','b','c'])
- head()
- drop([index or column])
- size
- apply(operation)
- merge

Magic
%ls -lh: files in dir + size
%rm file
% time
% timeit
% prun

x.? istrue for dtype

append
insert
delete
concatenate
vstack = stack vertically

.dropna()
.replace()

a.shape
a.ndim
a.size
a.dtype.name
a.astype = cast to type

.sort : return none
.sorted : return copy
.argsort : index

a = np.sort(5)

df.query(sepal >5 and width<@a')

sort_values([cols], ascending=[False, True])

argmax vs max

sns.heatmap

np.zeros(len, dtype=int)
np.multiply.outer(a,b)
np.add.reduce(a,b) reduce dim
np.add.accumulate
np.power
np.absolute
np.cumsum(axis=1)

.transpose

Regex
^ start string
$ end
\s white space |\S
\d | \D
* 0 or more
+ 1 or more
? 0 or 1
. 0 or 1
{3} 3
{3,} 3 or more

[^abc]
[a-q]

```
pd.read_csv(linktofile(), compression='gzip', header=None, names=['Jaar', etc.], sep='\t', quotechar='"', error_bad_
        lines=False, skipinitialspace=True)
top10 = kvr['Partij'].value_counts()[:10].index.values
Partijlo = kvr[(kvr['Partij'] == 'PVDA') | etc]
Crors = pd.crosstab(partijlo.Jaar, partijlo.Partij)

cross.plot()
partiesleft = len(set(kvr['Partij']))
kvr Partij.str.replace('[^\w\s]', '')      of str.lower of (' ', '')
Kvr = kvr[pd.notnull(kvr['Partij'])]
kvr = kvr[~kvr['Partij'].str.contains('vragen')]
Kvr.Partij.value_counts().sort_index()
                    index_col of sort_index(inplace=True)
pd.pivot_table(df, values=['Count'], index=['Name'], columns=['Sex'], aggfunc=np.sum, margins=
                    True, dropna=False).sort_values(by='All', ascending=False)
pivot['ratio'] = np.log2(Pivot['Count']['M'] / Pivot['Count']['F'])
        set(pivot[(pivot['ratio'] > -0.1) & Pivot['ratio'] < 0.1)])

df = df.fillna(0)
df['ass_mean'] = df[['ass_week1', etc]].mean(1)      of .corr()
greater = ((df[['ass.1', 'etc']] > 9).sum().sum()    sum groter dan 9
df['weighted_mean'] = 0.4 * df.ass_mean + 0.6 * deel_mean
perc = ((df['ass_mean'] < 5.5) | df['deel_men'] < 5.5)).mean() * 100
Z_df = (df - df.mean()) / df.std()
        df.sort_values(['deeltoets_mean', 'names'], ascending=[False, True]

c = cc[c != 1]
rij = c.max().idxmax()
kolom = c.loc[rij].idxmax()      vraag10 = (rij, kolom)
```

Recall = TP / (TP + FN) → How many of the 'true class' are picked up by the test

Precision = TP / (TP + FP) → How many instances of class C predicted to be C, how many of those with a positive test do have the disease? (patients).

Accuracy = (TP + TN) / all → How many of all predictions were correct?

$$\sqrt{\frac{\sum (x_i - \bar{x})^2}{n}} = SD$$

np.random.randint (start, stop, (rows, columns)) (shape)

ex = (1, 10, (5, 10))

	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	4],
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[0,1] ex[:-2, 9] ex[1:5, 0:3] ex[2, 4:] ex[:, 3] ex[4, :]

Select single value by row and column: df.iloc[[0], [0]]

select single value by row and column labels: df.loc[[0], ['Country']]

boolean indexing: Series S where value is <-1 OR > 2: S[(S<-1) | (S>2)]

boolean indexing: Series S where value is not > 1: S[~(S>1)]

df[df['Population'] > 1200000000]

S.drop(['a', 'c']) → drop values from rows (axis = 0).

df.drop('Country', axis=1) → drop values from columns (axis=1)

df.sort_index() → sort by labels along an axis

df.sort_values(by='Country') → sort by the values along an axis

print working directory: %pwd (this is the path we are in)

list working directory contents: %ls

change directory: %cd Pictures (gaat naar pictures directory)

make new directory: %mkdir | Remove file: %rm

copy files and directories: %cp | Remove directory if empty: %rmdir

move file to new directory: %mv file directory | %man !

%time, %timeit, %prun, %memit, %mprun

np.size?

pd.value_counts?

pd.DataFrame.drop? / help(pd.DataFrame.drop)

pd.Series.str.[TAB]

str.*find*?

L.mean(axis=0) doet voor elke rij het gemiddelde.

L.mean(axis=1) doet voor elke kolom het gemiddelde.

Boolean mask: x1[x1 % 3 == 0] (alles wat deelbaar is door 3 blijft over in x1.

Boolean indexing: L[L=5.5].mean() gemiddelde van voldoendes

Hoeveel voldoendes? (L>=5.5).sum() want True == 1

Kans op voldoende? (L>=5.5).mean() want de mean over een boolean array geeft de ratio van trues.

Fancy indexing: select elements (1,0), (0,1), (1,2) and (0,0) from b.

b[[1,0,1,0],[0,1,2,0]]

lengtes = A[:,0]

aantal = A[:,1]

gemiddelde = (lengtes * aantal).sum() / aantal.sum()

modus = A[aantal.argmax(), 0]

Sd = np.sqrt(((aantal * (lengtes - gemiddelde)**2).sum() / aantal.sum())

hoeveel hebt van elke waarde? np.unique(grid, return_counts=True) geeft een histogram.

iris.query('sepal_length > 5 and sepal_width < @a')

Return iris samples of those samples with a sepal length larger than 5 and sepal width smaller than the square root of S: a = np.sqrt(5) print(a)

What are the maximal sepal length for each species: iris.groupby('species')['sepal_length'].max()

Sort iris first by sepal_length, and then by sepal_width. We want that the sort is in reverse order (so the longest values are on top), but that ties in sepal_length are broken by sepal_width in standard order: iris.sort_values(['sepal_length', 'sepal_width'], ascending=[False, True])

create the dataframe X containing all the numerical data in iris and the Series y which contains the species information: X, y = iris.drop('species', axis=1), iris.species

How are number per species: iris.species.value_counts().plot(kind='bar')

np.setdiff1d

axis = 0, gaat over de columns ↓
axis = 1, gaat over de rows →

top = kvr.Partij.value_counts().head(10)
cross = pd.crosstab(kvr.Jaar, kvr.Partij)[top.keys()]

remove = ["[^a-zA-Z0-9]", "\s", "alle", "beide", "groep", "fractie", "fr", "lid"]
kvr.Partij = kvr.Partij.str.lower().replace(regex=remove, value="")
kvr = kvr[~kvr.Partij.str.contains("vragen", na=False)]
kvr.dropna(subset=["Partij"], inplace=True)
parties_left = kvr.Partij.unique().size

kvr["deelvragen"] = kvr.Vraag.str.count("\?")

Pearson correlation
numpy = np.corrcoef(x, y)[0,1]
pandas = x.corr(y, method="pearson")
import scipy.stats
scipy.stats.pearsonr(x, y)[0]

df["assignment_mean"] = df[df.columns[0:6]].mean(axis=1)
all_assignments = df[df.columns[0:6]]
greater_than_9 = all_assignments[all_assignments > 9].count().sum()

df = df.sort_values(["deeltocts_mean", "naam"], ascending=[False, True])

RMSE = np.sqrt(np.mean((predictions - targets)**2))
voorspelt - gemeten

# vraag 10 = set()
c = df[df.columns[0:6]].corr()
c = c[c != 1]
row = c.max.idxmax()
col = c[r_max].idxmax()
vraag 10 = (row, col)

np.where
np.argwhere

normalise = p/p.sum(axis=0)

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

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

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

Pandas

open files: -> pd.read_csv('file.csv', sep="\t", encoding='utf-8')
-> with open('file') as f:
from web -> f = requests.get(url)

- precision accuracy = $\frac{\text{True positive}}{\text{TRUE Positive + False Positive}}$
- recall = $\frac{\text{True positive}}{\text{True Positive + False Negative}}$

Reshaping: pd.concat, pd.melt
select col: df.loc
merge: pd.merge(df1, df2, how='inner', on='x1')
df.groupby()

Numpy:
inspecting: a.shape, len(a), a.ndim, e.size, b.astype(int)
Sort: a.sort(axis=0)
slicing: a[row][col]
indexing: a[a<2] → boolean
Fancy Indexing: [..., ...][..., ...]

- vul NaN -> df.fillna(0)
- pd nieuwe kolom: df["new_colum"] = ...
- boolean indexing: df["column"] > 4
- Z-df = (df - df.mean()) / df.std() ← Z-normale waa

~~Precision~~:
P = .1
Precision = Fout x TP / ((F x 0.9) + ((1-F) x .1)

$\frac{0.9}{0.9+0.}$

Prec. = $\frac{TP}{TP+FP}$ Rec. = $\frac{TP}{TP+FN}$ Acc. = $\frac{TP+TN}{total}$

np.into(np.array.dtype) → b.astype(int) ? → np.int64, float32

np.: add, substract, divide, multiply, sqrt, log, log2

np.: sum, min, max, cumsum, corrcoef, corr, std

np.: zeros((3,f)), ones((2,3,f)), arange(10,25,5), linspace(0,2,9)

np.: any, all .ravel() == .flatten() , reshape vs. ~~resize~~

np.: sort(), sort(axis=0), concatenate((a,b),axis=0)

diag: return L[range(L.shape[0]), range(L.shape[0])]

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

np.arange(1,f)[:,np.newaxis] , np.split()

%ls -lh , %rm <file> , %%time, %time it , .<TAB> , ?

index labels ↓ index positions ↓

df[x == 2] vs. df.loc[x == 2] vs. df.iloc[1:3]

pd.isnull() vs pd.notnull() , df.sort_values, df.sort_index

df.dropna(subset=["x"]) , df.nlargest , ns.smallest , df[x].value_counts

df.plot(kind = bar/barh) , df.plot.hist , df.plot.scatter , df.fillna()

pd.merge(a, b, how = inner/outer/left/right, on = x1)

df.drop_duplicates() , pd.Series(data = values, index = labels)

(df[assignments] > g).sum().sum(), data[-1], data[::2]

df.groupby → .apply() → .unstack() → .head()

df[~ df.x1.isin(df2.x1)] , .index , .values

~~df.str~~ wel of niet df.x1.str.replace("|".join([\s,\W],"u"))

df[~ df.x1.str.contains("string")] , pivot[[f,M]].min(axis=1)

C = df[assignments].corr(), x = C[C != 1].max().idxmax()

y = C[C != 1].loc[x].idxmax() , np.transpose == np.T , np.abs()

pd.merge(), pd.concat() , df.reset_index() , pd.melt

np.copy(a), a.copy() , a.view() , random_ <TAB>

divisible: L[L % n == 0] , round(float, 1)

from numpy import doc → np.doc? → np.lookfor

^ starts , ends $, '\.' alle . , '(^(?!species$).*)' alles behalve species

\d digit , \W not \w , \s whitespace , {3,4} 3,4,3+x

Numpy

Maken

```
np.random.randint(0, 100, (shape))
np.arange (start, stop, steps)
```

Informatie

```
a.shape - dimensies
a.shape[0] - rij
a.shape[1] - kolom
a.size - elementen
a.dtype - element en
a.astype(int) - convert
```

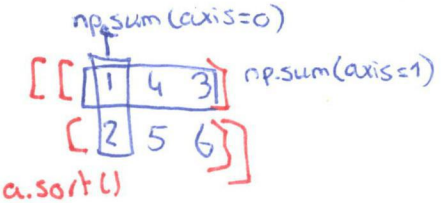
Manipulate

```
np.transpose(a)
np.insert (a, 1, 5)
np. append()
np.delete()
np.reshape()
```

Math

```
a.sum()
a.min()
a.max()
a.cumsum()
a.mean
a.median
np.std(a)
```

} over hele array



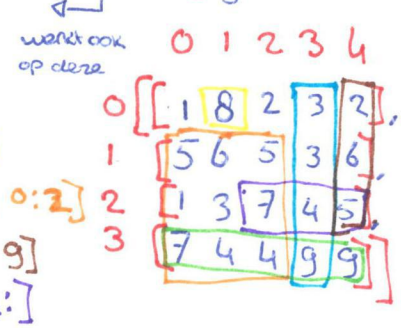
a.sort()

a.sort(axis=0) kolommen

(axis=1) rijen

Index Slicing

```
a[0,1]   a[1:3, 0:2]
a[:,3]   a[:-2, 9]
a[3,:]   a[2, 2:]
```



a.reshape(size,) → wordt 1D

np.arange(1, 4)[:, np.newaxis - 1D in kolommen ← nieuwe axis

< -10 of > 10: maak er absoluut van! np.abs

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

recall = $\frac{TP}{TP+FN}$ (actual TP)

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

np.var(a)

np.std(a)

Rmse = np.sqrt(np.mean((L[:,0] - L[1,0]) ** 2))

waardes kolommen

Jupyter handig

function? - documentatie

help(function) - documentatie

function?? - source code

Regular expressions

1 of df.str.replace(re, ng)

\w words

\W - not-words

[0-9] - numbers

[a-z] - lowercase

[A-Z] - upper

^hoi - begint met hoi

\s space

\? alles met ?

@[...] niet.

%lsmagic

Pandas

Inlezen

```
Pd.read_csv('naambest', sep,
            index_col, skipinitialspace)
df.sort_index()
df.sort_values('kolomnaam', ascending=false)
df.rename(columns={'dd': 'new'})
```

Displays

boolean!

```
df[df.length > 7]   - rows that meet criteria
df.drop_duplicates()
df.drop(columns=['x', 'y'])
df.sample(frac=0.5)
          n=10
df.nlargest(n, values)
```

Subsets

```
df[['naam', 'mean']]
df.filter(regex)
df.loc[:, [1, 2, 5]]
```

Summarizing

```
df[kolom].value_counts
df.sum()
df.count()
df.describe()
df.agg(['count', 'min', 'max'])
```

for each in

what is total

df.groupby('kolom').kolom.sum()

voor totalen bij elkaar

df.pivot_table("values" = , index= , columns=
aggfunc = 'mean', fill_value,
margins=false)

df.dropna(subset['kolom', 'kolom2'])

Naam moet in beide sets voorkomen!

Pd.merge(df1, df2, how='left' on='kolom1'

intersection ↳ optional

Pd.crosstab(df['kolom'], df['kolom2'])

↳ altijd count

Rmse

df['kolom1'], ['kolom2'], cross()

Pd.set_option('display.max_columns', 10)

df.isna(), isin(), contains() → all boolean

lambda x, y: x * y → shortcut for function

numpy

Arrays are created with np.array([ ])
np.arange(10, 25, 5) creates an array of evenly spaced values (5 is the step value)
Inspecting your array is done by
a.shape array dimensions
len(a) length of array
b.ndim number of array dimensions
e.size number of array elements
b.dtype datatype of array elements.

a[2] subsetting, select element on 2nd index
b[1,2] equivalent to b[1][2] or [row][column]

$$\text{precisie nederlich} = \frac{(1-p)\times 0.9}{(p\times 0.1)+((1-p)\times 0.9)}$$

a[0:2] slicing for a/ in a row
a[0:2, 1] slicing for a/ in a column
a[:, -1] get the last column

$$\text{precisiecreek} = \frac{p \times \text{Precisie Zielk}}{\underset{(p)}{(p\times 0.9)}+\underset{(p)}{((1-p)\times 0.1)}}$$

a[a<2] boolean indexing
a[["fill in columns, row, etc"]] select a subset of the matrix's rows and columns. = fancy indexing

i = np.transpose(b) } permute array
i.T } dimensions
np.intersect1d() or np.union1d() or np.setdiff1d()
Flatten the array np.flatten() or b.ravel()
Reshape the array g.reshape(3,2)
concatenate arrays np.concatenate((a,d),axis=0)
axis has to be specified.
np.arange(start, stop, step) optional[:, np.newaxis]

g = a - b subtraction
np.subtract(a,b)
g = a + b addition
np.add(a,b)
g = a / b division
np.divide(a,b)
g = a * b multiplication
np.multiply(a,b)

$$\text{nwogbegaafd} = \frac{\underset{(p)}{0.01}\times \text{precisie nwogbegaafd}}{\underset{(p)}{(0.01\times 0.9)}+\underset{(p)}{((1-0.01)\times 0.01)}}$$

np.exp(b) exponentiation
np.sqrt(b) square root
np.log(b) logarithm
e.dot(f) dot product

a == b returns True or False
a < 2 returns True or False
a != 2 returns True or False
np.array_equal(a,b)
np.all() or np.any()
a.sum() sum of values
a.min() b.max() returns max or min
a.cumsum() cumulative sum
a.mean() mean
b.median() median
a.corr correlations
np.std() Standaard deviation
np.where() replace values in array or to see indices of matches in arrays
a.sort() sort an array
c.sort(axis=0) sort an array on ax

---

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

a multiIndex dataframe is created with pd.multiIndex()

$$mse = \sqrt{\frac{\sum(\text{Predicted}_i - \text{Actual}_i)^2}{N}}$$

df.head()
df.tail()
df.idxmax()

subset variables (columns)
df[['width','length','species']] select multiple columns u specific names.
df.width or df['width']
df.filter(regex='regex')

Reshaping data is done with the following
pd.melt(df) (colums -> to rows)
pd.concat([df1,df2]) (df1+df2)
pd.concat([df1,df2],axis=1) (df1+df2)
df.pivot(columns='var', values='val')
(rows -> colums)

df.sort_values('mpg') order rows by values in a column (low optional arguments apply
df.rename() rename colums.
df.sort_index() sort the index of a dataframe

subset observations (rows)

df['w'].value_counts() Count number of rows with each unique value as variable
len(df) # of rows in df .Nunique()
df['w'] # of unique value) in a column.
df.describe() dexribes the df.

df[df.Length > 7] extract rows that meet logic
df.iloc[10:20] select rows by position
df.nlargest(n,'value') select and order top n entrie
df.nsmallest(n,'value') select and order bottom n entrie

df.loc[:,'x2':'x4'] select all columns between x2 an x4 (in
df.iloc[:,[1,2,5]] select columns in positions 1,2 & 5
df.loc[df['a']>10,['a','c']] select rows meeting a logical condition, and only specific to the column.

df.groupby(by="col") return a groupby object grouped by values in a column.
additional groupby functions =
size() & agg(function)
df.groupby(by="")["col"]
new column? df['volume'] = 'value'

.sum() .mean() .min() .max() .count() .median()
.var() .std() .apply(function) df.dropna()
df.fillna(value) pd.merge() df.intersect1d() df.union1d()
df.difference()

Pandas

Read file: pd.Read_csv(filename, compression, sep)
df.set_index("column name")
sorting: df = df.sort_values(by = [columnname], ascending = [bool])
missing values: df.fillna(value) of df.dropna
mean of colums: df["column"] = df.loc[:, "von column":"tot colum"].mean
= df[["colums"]].mean(axis=1)
filters df = df[df["column to filter"] == 1] df[df.loc[]]
slicing: df.loc[:, "von":"tot"]
Select more columns: df[["name","age"]]
df.sort_index
df.sort_values("name", ascending = false)
df.rename(columns = {'name':'new_name'})
df.drop(columns = ['name', 'age'])
df['name'].value_counts()

pd.merge(data, data, how, on)
- how: left, right, inner, outer on: column

df.groupby(by = "name", level = "ind")
df.Pivot_table(index="name", columns = "age"
aggfunc = {"column": sum})
pd.concat(data, data)

Logic operators

normal ones
column.isn
pd.isnull(obj)
pd.notnull(obj)
df.any()
df.all()

describe data

sum() min()
max() count()
mean() median()
quantile([ ])
std() var()
apply(function)
abs()

numpy

np.shape → (Row, column)
np.argwhere(array + condition) → gives coords
np.isnan → boolian array
np.Reshape(Row, column)
filter → array[array condition]
np.asarray → macht np array von list
sorting = ~~array.sort(axis)~~ np.sort(array, axis)
np.all = ! np.all([condition] & (condition)) → gives bool
np.ndim = array dimentions

array.sum()
.min()
.max(axis)
.cumsum()
.mean
.median
.corrcoef

np.std(array

np.transpose → turn array
np.append
np.insert(array, coord)
np.delete
np.Hstack / np.Vstack
np.hsplit(array, index_to_split)

Slicing array[Rowstart : Rowend, columnstart : Columnend]
np.random.Randint(start, end)
np.arange(start, end, stepsize)

from numpy import doc
? np.doc
np.lookfor

Precision = tp / (tp)
Recall = tp / (tp + fn)
acc = tp + tn / (all)
f1 = 2x ((Pre + Rec) / (Pre x Rec))

acc: $\frac{TP+TN}{Total}$ prec: $\frac{TP}{TP+FP}$ Rec: $\frac{TP}{TP+FN}$

// = floor div

np array => np functie: np.sum, np.max, etc..

np.zeros(length/(dims), dtype) ar.reshape((dims)) np.concatenate([ar1, ar2]).. np.split(grid, [indexes])
↳ hsplit, vsplit

[:5] first 5
[5:] all after 5
[4:7] subarray
[::2] step 2
x[start:stop:step]

np.count_nonzero(statement) counts bools np.ravel(arr)

np.sort(arr), np.argsort(arr) np.arange(n)

arr.transpose() ar.flatten()/np.ravel(arr)

arith: np. add/subtract/negative/multiply/divide/floor_divide/power/mod/abs/sqrt

agg: np. sum/prod/mean/std/var/min/max/argmin/argmax/median/percentile/any/all (let op axis)

↳ nan error on nan's te handlen.

np.cumsum(axis),

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

diff. add/subtract/multiply/divide/floordiv/mod/pow (doe ff ? voor args)

dropna(axis, how, thresh) fillna(method (ffill, bfill))

pd.concat() (check? voor args)

df.append(df2)

pd.merge() (check ? voor args)

values_counts, is_in, cross_tab

.str. lower/upper/contains/len/strip/replace

mean/median/mode/std

.T, .unstack, .stack

idxmax, idxmin, nunique, nlargest

groupby.apply.unstack

\d digit	+ one or more	[] select group
\w word char	{3} 3 times	^ start string
\s whitespace char	{2,4} 2-4 times	[^...] not
\D not \d	{3,} 3 or more	$ end string
\W not \w	* zero or more	
\S not \s	? 0 once or none	
	. any except line break	

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

recall = TP / (TP / FN)

precision = TP / (TP + FP)

dropna(axis = 1)

fillna()

isnull() → boolean

isnotnull → boolean

Numpy

np.arange(start, stop, step)

np.reshape(..., ...)

[[L % n = 0] → deelbaar door n

add_per_colum = np.arange(L.shape[1]) + 1

add_per_row = np.arange(add_per_column(L.T).T

random.choice(i for i in range(5))

beter Niet

→ df[~df['party'].str.contains('vlugen')]

→ len(df['Party'].unique())

with open bestand as f:
  for i in f:
    for j in i:
      print(j)

replace tokens + lowercase

→ df['Party'].str.lower().str.replace("\W","")

remove specific list of words

→ df['Party'].str.replace("|".join(listofwords),"")

⇒ drop rows with non

⇒ df.dropna(subset=['Partij'])

→ count = len } aggrfunction

→ sum = sum }

→ df.[colum].str.count('?') = tel aantal dat vragen

d.crosstab(df['colum'], df['column']).filter(...)

d.read_csv(skipinitialspace=True, sept='\t', names='list of colums'

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

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