Numpy functies

np.log2()
np.absolute()
np.nan
np.linespace(..., ..., ...)
np.arange(start, stop, step)
np.median()
np.sum()

Matplotlib

df.plot()
df.plot.set()
df.plot.scatter()
df.plot.set_xlabel('...')
df.plot.set_ylabel('...')
df.col.hist()

n	stay	switch
3	perc.	perc.
4	perc.	perc.

↳ kijken: stay & switch
y-as: perc.
x-as: 3,4

Subsets

sub = df[df.col == ...]
sub2 = df[(df.col1 = ...) & (df.col2 > ...)]
(df.col = ...).sum()/mean

Pandas functies

Pd.isnull()
Pd.notnull()
Pd.read_csv()
Pd.read_excel()
Pd.concat
↳ can be used for chunks if chunk-size is passed to Pd.read_csv
Pd.crosstab()
↳ Pass series for index, series for cols & margins = True for 'All'
↳ crossdf['...'].unstack() for same result as pivot_table()

Pandas series

df.col.unique()
df.col.value_counts()
df.col.dtype
df.col.to_datetime()
df.col.str.contains('...')
↳ boolean, dus kan nog .mean()/.sum() achter.
df.col.str.strip()
df.col.idxmax()
df.col.astype(type)
↳ change type of col
df.col.values
↳ get values from a col as an array
df.col.index
df.col.str.findall(Pattern)
df.col.min/max/sum/mean/mode

Plotly

df.col.iplot(kind=...)
df.col.value_counts().iplot()
df.crosstab.iplot()

Seaborn

sns.set() → to use seaborn on matplotlib.pyplot as plt
sns.boxplot(df)
sns.pairplot(df, hue='categoreen column')
sns.distplot(df)
↳ combination of hist & kdeplot
sns.lmplot()
sns.heatmap(df.corr())
sns.jointplot()
sns.scatterplot
sns.kdeplot()
sns.FacetGrid(df, row, col)
↳ after this, use grid.map(plt.hist, bins=)
sns.plot.add_legend()
sns.factorplot()

%magic %cp %time %%time %timeit %ls

Linux comman

Counter(list)

Strings

s.len
s.get
S.split('seperator')
S.count('...')
S.find('...')
#1 '.join(list)
S.lower()
S.replace()
S.splitlines()
S.startswith(
S.strip()
S.upper()

regex

re.compile("|".join(list_of_patterns))
re.findall('pattern', String)
re.sub('pattern', 'replace str', 'search str')
[...]
↳ choice
[^...]
↳ any except within
[a-z]
↳ any characters between
. → any single char.
\s → any white space
\S → any non white sp.
\d → any digit
\D → any non digit
\w → any word char.
\W → any non word cha
(a|b) → match either a or b
a? → 0 or one of a
a* → 0 or more of a
a+ → one or more of a
a{3} → exactly 3 of a
$ end of string
^ start of string

Pandas dataframe functies

df.iterrows()
df.sort_index()
df.set_index()
df.reset_index()
df.Applymap(lambda x: *functie wat je met x doet en waar je x mee vervangt*)
df.shape()
df.pivot_table(index='colname', columns='colname')
df.groupby('colname')
df(.col).dropna(subset='...')
↳ subset to define in which col to look for missing values
df.sort_values()
df.count() → non na values
df.round(int/dict/series)
df.mean
df.corr()
df.to_pickle('filename')
↳ save df as pickle file
df.describe
df.fillna()
df.head()/df.tail()
df.drop()
df.copy()

Pandas subsets

columns: df[['col1', 'col2', ...]]
single column: df['col'] as series
row: df.loc['row_index']
rows: df.loc[['row1', 'row2', ...]]
range of rows: df.loc['idx1':'idx4'] → t/m
rows & columns: df.loc[[rows][cols]]
rows w/ step: df.loc['idx1':'idx4':2]
row & column: df.loc['row', 'col']
all rows & column: df.loc[:, ['col1', 'col2']]
index selecting: df.iloc[slicing as np slicing]

Theory1

Accuracy = (TP+TN) / alles
Precision = TP / (TP+FP)
↳ = P(echte ja | uitslag ja)
recall = TP / (TP+FN)
vb) accuracy 95%
occurence 1 op 1000
hoeveel mensen met positieve test zijn/er besmet?
TP = 0.95 * 0.001
FP = (1-0.95) * (1-0.001)
Precision = TP / (TP+FP)

Theory2

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

broadcasting

→ complement van precision
= 1 - (TP/(TP+FP))
= FP/(TP+FP)

five different ways how many men and women were on boat

titanic.sex.value_counts()
titanic.group_by('sex')['survived'].count()
{s: titanic[titanic.sex==s].sex.count() for s in titanic.sex.unique()}
titanic.pivot_table(index='sex', values='survived', aggfunc=len)
Pd.crosstab(titanic.sex

str.contains	str.extract	str.startswith	re.sub	re.findall
\d	digit	{3}	exactly 3 x	
\w	letter	{2,4}	2 to 4 x	
\s	whitespace	{3,}	3 or more x	
\D	non-digit	*	0 or more x	
\S	non-whitespace	?	once or none	
[ ]	one of	^	start of string	
[a-z]	a to z	$	end of string	

```
def strcount(file):
    i = defaultdict(int)
    with open(file) as f:
        for line in f:
            for letter in line:
                i[letter] += 1
```

omgekeerd sort = hoog → laag dus ascending = False

nltk.tokenize.wordpunct_tokenize()

`df[df[x] < df[x].quantile(.99)]` → verwijder 1% outliers

predicted values \ actual values	P	N
P	TP	FP
N	FN	TN

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

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

import seaborn as sns.

y = sns.load_dataset("x")

y.groupby(["x"]).sum/count() → dataframe gegroepeerd

y.sort_values('z', ascending=False) → geef dataframe gesorteerd hoog-laag o.b.v. z

y['z'].value_counts() → hoevaak z voorkomt in y

y.loc[y['z'] == 'thiez'] → dataframe met alleen de regels waar 'Thiez' in z voorkomt

y['z'].values → Returns array of values in z.

y[['z', 'w']] → maak dataframe y with columns z and w

np.mean, np.median, np.std

Pd.DataFrame.from_dict(x, orient='index')

.sort_index()

.idxmax() → hoogste index waarde

pd.Index(a) & pd.Index(b) → intersect

pd.Index(a) | pd.Index(b) → union

pd.Index(a) ^ pd.Index(b) → symmetric difference

y[(y)>0.3)] = masking

y.T → transpose

y[y.x > 100]

.fillna(0)

y.dropna()

y.set_index("x")

---

pd.read_csv('x', encoding="ISO-8859-1", sep=';' of sep='\t', index_col=1)

.capitalize() → make a capital letter of the first letter of str

---

Pivot voorbeeld: maak pivot die meest voorkomende materialen geeft van de 10 meest voorkomende kunstenaars. Index moet de naam kunstenaar zijn, kolomme materiaal en value aantal keer dat de kunstenaar het gebruikt heeft

Antw: - top10 = kunst[['Naam_kunstenaar', 'Id']].groupby('Naam_kunstenaar').count().sort_values('Id', ascending=False).iloc[:10].index.values

- kunst.pivot_table(kunst[kunst['Naam_kunstenaar'].isin(top10)], values='Id', index='Naam_kunstenaar', columns='Materiaal')

↳ aggfunc=len is count, sum = values optellen

---

pd.crosstab(y.z, y['w'])

---

count how many time each word occurs

from collections import defaultdict

with open(bestand) as f:

tellings = defaultdict(int)

for l in f:

for c in list(l):

telling[c] += 1

tellings = pd.Series(telling).sort_values(ascending=False)

return tellings

---

axis=1 (rij) =0 (kolom)

---

precision = TP / (TP + FP)

recall = (TP / (TP + FN))

Rij selecteren DF.ix[...]

Series naar DF: pd.DataFrame({"kol1": ser1, "kol2": ser2})

DF.describe() → count, mean, std, min, max

df = Dataframe
Se = Series
xy = var(int)
pd.<tab> ?

Algemeen
pd.Series(data, index='index')
pd.DataFrame(data, columns=[list], index=[List])
[df/Se].[index/columns] => namen van I en C
df.[size/shape/ndim/dtype]
Se[(Se > 0,3) & (Se < 0,8)] masking
df.Loc[(df.col > x) & (df.col < y), [list col]) masking
df.transpose => switch cols and index
sum(), mean(), median(), min(), max()
df.dropna().describe()
count() => totaal aantal
defaultdict() => makkelijk toevoegen (int/float/str etc)
with open(file) as f: regel voor regel
for i in f:
pd.read_[csv/excel](file)
df.value_counts()
df.sort_[index/values]
df.query()
pd.crosstab
df.isin()
with BZ2File(file) as f:

Groeperen
df.groupby('key').sum()
df.groupby('key').aggregate['min', np.median]) meerdere functies
df.groupby('key').apply(eigen functie)
df.pivot_table('col', index='col', columns='col')

Functies op DF
df.col [/*+-%**] df.col
df.substract(df.col, axis=0) => op kolommen

None/np.nan
isnull(), notnull() = bool, alles niet null
dropna(), fillna(method='[b/f]fill', axis=0/1)

samenvoegen
pd.concat(df1, df2), pd.merge(df1, df2)
df1.append(df2)

11-04

Regex
Se.str.contains(r'') => bool ook .startswith()
re.findall(r'', data)

\d, \D = Any getal, geen getal
\s, \S = Any spatie, geen spatie
\w, \W = letter, geen letters
\b = alles behalve newline
+ * ? = 1+, 0+, 1 of 0 keer
{3} = 3x
{3,} = 3+
| or ^ = niet iets
(...) groep

Grouping & Pivot!

planets.groupby(kolom1).sum()

df.groupby('kolom1').

.aggregate(['min', np.median, max]) → drie methoden voor elke kolom

.aggregate({'data1': 'min', 'data2': 'max'}) → methode per kolom

.filter(filter_func) → filteren of gem. groter is dan iets van een cat.

.transform(lambda x: x - x.mean()) → houdt zelfde shape

.apply(norm_by_data2) → alles delen door som van kolom data2

df.groupby(mapping)

mapping = {'A': 'vowel', 'B': 'consonant', 'C': 'consonant'}

df.groupby([str.lower(), mapping]).mean() → hierarchical indexing

L = [0, 1, 0, 1, 2, 0]

df.groupby(L) → index 0, 2, 5 samen

titanic.groupby(['sex', 'class'])['survived'].aggregate('mean').unstack()

= titanic.pivot_table('survived', index='sex', columns='class')

indexen cols kunnen lijsten zijn voor hierarchical

decade = 10 * (planets['year'] // 10)

decade.name = 'decade'

planets.groupby(['method', decade])['number'].sum().unstack().fillna(0)

pd.qcut(titanic['fare'], bij col of index deelt numerieke variabelen in tweeën

pivot_table(agg_func = {'survived': sum, 'fare': 'mean'}

margins = True voor totals

data['2015'] werkt op datetime

str.lower(), str.<TAB>, str.capitalize

Als 'info': ['B|C|D', ...] dan maakt df['info'].str.get_dummies('|') dan dummy variabelen

str.contains('[Bb]reakfast').sum()

dummy_df[dummy_df.isin(frequent_sections)]

pd.get_dummies(categorical_series).astype('bool')

for file in tqdm_notebook(namelist):

fs archive.open(file)

tree = etree.parse(f)

root = tree.getroot()

root.find('handelingen')

.set_index('title', inplace = True)

df['n_sections'].idxmax()

.value_counts → histogram

.quantile(0.99)

& → intersection | → union

^ → symmetric diff.

pd.DataFrame.from_dict(dict1, orient = 'index')

tr.sort_values(by = ['kolom'], inplace = True, ascending = False)

data[data.notnull()] → NaN of None

df1.intersection(df2)

pd.merge(df1, df2, how = 'inner')

'outer' ↓ intersection ↓ union

Theorie

Omgaan met grote bestanden:

1 Niet uitpakken, gezipte file regel voor regel

2 Clean memory als je klaar bent

3 verdeel verwerking over versch. computers.

Z-score:

t60Z = (t60 - t60.mean()) / t60.std()

Missing values:

1 Invullen met constante (NaN of None?)

2 Vervangen door gem (numeriek) of modus (categorical)

3 Vervangen door random waarde

4 Vervangen door inferred waarde

Outliers herkennen:

1 histogram 2 scatterplot 3 |Z-score| < 3

Regex

soup.findall('Affiliation')

parts.RegisteredName.text

[a-zA-Z] een van opties tussen haken

\+ een of meer

* nul of meer

? nul of een

{3} {3,6} 3, 3-6

^ niet of start string

$ einde string

. elk karakter

\S, \s (geen) whitespace

\D, \d (geen) cijfer

(...) wat je wil capturen

re.findall, split, sub

\W, \w (geen) cijfer of letter

\b word boundary

Pandas

```
import numpy as np
import pandas as pd
import seaborn as sns
import re
```

Lees csv file in

```
pd.read_csv(bestand, sep='\t', index_col='...', usecols=['...','...'])= df
```

```
from tqdm import tqdm_notebook
```

Sort op index of kolom

```
df.sort_index(axis=0, ascending=T/F, inplace=True)
df.sort_values('kolom', ascending=True...)
```

kolom als index

```
df = df.set_index('kolom')
```

unieke waarden tellen

```
df['kolom'].value_counts()
```

Dataframes aan elkaar plakken

```
s = pd.concat([df1, df2], join='inner', axis=1)
```

specifieke kolom van specifieke rij

```
peiling.loc['D66']['zetels']
peiling['zetels'].loc['D66']
```

Partij met meer dan 25 zetels

```
zetels = pdf.loc[df['zetels']>25]
```

Data Cleaning

Name columns

```
df.columns = ['...','...']
```

check voor null values

```
pd.isnull()
```

Drop rows with null values

```
pddf.dropna(axis=1)
```

replace null values with mean

```
df.fillna(s.mean())
```

Groupby

selecteer specifieke waarde in kolom

```
nl = df[df['original-lang']='nld']
```

5 meest vertaalde boeken uit nl

```
meest = nl.groupby('target-lan').max()
['num-transl'].nlargest(5)
```

maak hier tuples van

```
nst = set(x for x in meest.items())
```

hoeveel mannen en vrouwen waren aan boord

```
titanic.groupby('sex')['survived'].count()
titanic['sex'].value_counts()
```

Pivot tables

```
pd.pivot_table(df, index='sex', columns='class', values='survived', aggfunc=len)
titanic.pivot-table()
```

hoeveel mensen zaten er per geslacht in elke klasse.

```
titanic.pivot_table(index='sex', columns='class', values='survived') of:
pd.crosstab(titanic.sex, titanic['class'])
```

gemiddelde leeftijd van mensen die het wel of niet hebben overleefd

```
titanic.pivot_table(index='sex', columns='survived', values='age', aggfunc=[np.mean, np.median])
```

wanneer precies is de variabele 'adult_male' waar

```
titanic1 = titanic[titanic['adult_male']>(
titanic1.pivot_table(index='age', values=['adult_male'], aggfunc=sum)
```

preprocess:

```
-csv(bestand)
prince = pd.read
prince.text = prince.text.astype(str
```

regular expressions

bestand inladen:

```
bestand = open('prince_lyrics.csv', encoding='utf8')
```

Hoeveel nummers bevatten het woord _sex_ (met of zonder capital)

```
print(prince.text.str.contains(r'\b[Ss]ex\b').sum
```

Hoeveel nummers beginnen met sex?

```
print(prince.text.str.contains(r'^[Ss]ex\b').sum() of
prince.text.str.startswith('Sex').sum() + prince.text.startswith('sex')
```

Hoeveel nummers bevatten sex maar niet letterlijk sex als woord (kolom)

```
ef = prince.text.str.findall(r'[Ss]ex\w+') | print(ef[ef.str.len()>0].head())
print(ef[ef.str.len()>0].count()) | from collections import Counter
print(Counter([s for l in ef.values for s in l]))
```

Regex * = 0 or more + = 1 or more ? = 0 or 1 {2} = exactly two
{2,5} = between 2 and 5 {2,} = 2 or more [ab-d] = one charachter of a, b c d
[\b] = backspace \d = one digit \s = one space \w = one word ^ = start of strin
$ = end of string \b = word boundary . = any character except newline a = characters a
\ = escapes special character. → werkt op pandas Series (ook None)

String operations: str.capitalize() = hoofdletter monte.str.lower() monte.str.len()
monte.str.startswith('T') monte.str.split() **Regex methods:** match() = returns boolea
extract() = matched groups, findall() = for each element, replace() = replace pattern with str
contains() = boolean, count() = count occurances of patterns, rsplit() = excepts regex
extract first name: monte.str.extract('([A-Za-z]+)', expand=False. Find the names
that start or end with a consonant monte.str.findall(r'^[AEIOU]. *[aeiou]$')
Extract last name monte.str.split().str.get(-1) \W = [a-zA-Z0-9_] re.search = match pattern anywhere in string (pattern, string)

Wikipedia bestand per regel inlezen

```
with open(bestand) #(encoding='utf-8') as f:
    for line in f: of:
```

```
for l in tqdm_notebook(f)
split regels: gesplit = l.split('\t')[1:]
dictje = {}
for x in gesplit:
    weer = x.split(',')
dictje[weer[0]] = int(weer[1])
for x,y in dictje.items():
    iets[x]+=1
    iets[x]*=y
for iets in dictje.keys():
```

Precision, recall, accuracy
SOA Test = 95% accuraay
Als je een soa hebt zegt de test in 95% van de gevallen ja
Als je geen SOA hebt zegt hij ook 95 van 100 keer nee
Je test, het is positief: wat is de kans dat je geen SOA hebt
P(geen SOA | test ja) = precisie
Precision: TP / (TP + FP) we willen het complement
FP / (TP + FP)
TP = 0,95 × 0,001 FP = (1-0.95) × 0,999
↓ 1 op 1000 hebben SOA

handige overzichten: dir(str) / dir(pd) / dir(np)
DF uit dictionairy: pd.Dataframe.from_dict(dict, orient='index'
from collections import counter
correlatie: corr1 = df['k'].corr(df['i'])
maak dict van lijsten
lege_dict[kolomlijst] = [lijst, lijs1]
reset index
df.reset_index()
Series = 1 kolom uit df

```
df = pd.Dataframe({'key': ['A','B','C','A','B','C'], 'data': range(6)}, col=['key','data'])
df.groupby('key').sum
```

key	Data
A	3
B	5
C	7

voorbeeld inlezen en letters tellen

```
bestand = 'princelyrics.csv'
def telletter(bestand):
    from collections import defaultdict
    telling = defaultdict(int)
    with open(bestand) as f:
        for l in f:
            for c in list(l):
                telling[c]+=1
tellingseries = pd.Series(telling).sort_values(ascendi False
return tellingseries
```

50 %

Allerlei dir(pd), dir(np) %lsmagic

```
with open('Bestand') as file:
    for page in file:
```

```
Read_csv( )
df.sort_index()
df.sort_values('kolom')
df.set_index('kolom')
df.value_counts()
```

pd.concat, pd.merge → df1.merge(df2)

Selection

```
df['col'] → returns series
df[['col1','col2']] → nieuwe df.
df.iloc[0,:] → first row
```

Cleaning

```
df.columns = ['a','b','c'] → rename
pd.isnull(), pd.notnull() df.dropna()
df.dropna(axis=1) → drop columns
df.fillna(x), s.astype(float)
s.replace(1,'one')
```

Filter

```
df[df['col'] > 0,5]
df[(df['col'] >0,5) &(df['col'] <0,7)]
df.sort_values([col1,col2],ascending=[True,False])
df.groupby(col)
df.groupby([col1,col2])
df.groupby(col1)[col2].mean()/sum()/etc.
df.groupby(col1).agg(np.mean)
df.apply(np.mean), df.apply(np.max,axis=1)
df.pivot_table(index=col, values=[col2], aggfunc=mean)
```

Pd.crosstab?

Pd.DataFrame.from_dict?

→ Dict of Dicts to dataframe

```
pd.DataFrame(dict)
pd.Series(dict)
titanic['sex'].value_counts()
titanic.pivot_table(index='sex', columns
='class', values='survived')
pd.crosstab(titanic[sex], titanic['class'])
index.name = 'naam'
titanic.groupby('sex')[['survived']].mean()
```

de dubbele haken geven een mooi Frame

```
titanic.groupby([sex, class])[survived].aggregate('mean').unstack() → zorgt voor mooi frame
age = pd.cut(titanic['age'],[0,18,80])
titanic.pivot_table('survived',[sex',age], 'class')
pd.qcut(titanic['fare'],2)
```

Regex dir(str):

```
df[kolom'].str.findall(r'blabla').str.len(
```

match(), replace(), count(), len(), lower(), extract(), contains(), split()

```
Prince['text'].str.contains(r'sex').sum(
```

. → alles behalve '\n' (newline)

\w → [a-zA-Z0-9_]

\W → tegen overgestelde

\b boundary between words and not word

\s single white space

\S non white space character.

\t, \n, \r → tab, newline, return.

\d → [0-9] \D → alles behalve

^ = start, $ = end

+ = one or more occurences

* = zero or more

? = match 0 or 1 of thing to the left

[abc] = a or b or c [^abc] alles behalve

re.sub(regex, vervanging, string)

endswith(), str.startswith()

Plotting

```
df.plot(x= y= kind=[line, bar, hist, barh, box,
    scatter, pie], subplots=True or False
    → pakt kolom subplot
use_index=True → sticks for x axis
title='string' grid=False, legend=False
logx=False, logy=False, loglog=False,
xticks=[values for xaxis], yticks=[ ]
sort_columns=False)
```

```
Frequent_sections = [lijst met titels]
dummy_df = df.sections.apply(pd.Series).stack(
dummy_df = pd.get_dummies(dummy_df)
dummy_df = dummy_df[frequent_sec].groupby
    (by='page_id').sum()
```

Vlaag 9

```
chance_dict = dict()
for b in frequent_sections:
    chance_dict[b] = dict()
    for a in frequent_sections:
        if a==b:
            chance_dict[b][a] = 1
        else:
            bkans = sum(dummy_df[b]==1)
            chance_dict[b][a] = len(dummy_df[(
            dummydf[a]==1)&(dummydf[b]==1)])/bkans
```

conditionele kans: kans op A en B als je weet dat B er is. A+B / B

kans: je bent ziek + test goed

Recall = TP/(TP+FN)

Precision = TP/(TP+FP) test is positief + je bent ziek

kans

Laguages={l2.split(';')[0]: int(l2.split(';')[1])
for l2 in l.split('\t')[1:]}

```
titanic.groupby('sex')['survived'].count()
[s:titanic.sex==s].sex.count() for s
in titanic.sex.unique()}
titanic.pivot_table(index='sex',
values='survived', aggfunc=len)
pd.crosstab(titanic.sex, titanic.surv).
sum(axis=1)
#per geslacht, klasse, overleefd, 2 manieren
(titanic.pivot_table(index='sex',columns=
'class', values='survived',aggfunc=sum)/
zelfde maar met len
titanic.pivot_table(index='sex', columns=
'class', values='survived')
pd.crosstab(titanic.sex, titanic[class])
gem. mediaan leeftijd wel & niet overleefd
titanic.pivot_table(index='sex', columns=
'survived', values='age', aggfunc=
[np.mean, np.median])
adultmale?
= titanic.dropna(subset=['age'])
.pivot_table(index='age', columns='sex',
values'adultmale', aggfunc=sum)['male']
pd.testing.assert_frame_equal(t.query("sex
== 'male' and age >= 16")) + [t.adult_male])
def teldeletters(bestang):
from collections import defaultdict
tel=defaultdict(int)
with open(bestand) as f:
for l in f:
for c in list(l):
tel[c]+=1
telser=pd.Series(tel).sort_values(ascending=
False)
return telser
prince=pd.read_csv(bestand
prince.text=prince.text.astype(str)
prince.text.str.contains(r'\b[Ss]ex\b').sum
... str.startswith('Sex ').sum()+
... str.startswith('sex ').sum()
# niet sex. wel sexy
no=prince.text.str.findall(r'[Ss]ex\w+')
[no.str.len()>0].head
Counter([c for l in no.values for c in l])
# woorden met letter en cijfer
findall(r'\b[A-Za-z][A-Za-z0-9]*[0-9][A-Za-z0-9]*\b | hetzelfde switched
klinkers
100*prince.text...findall(r'[AEeiouy]').
str.len()/prince.text.str.len()).describe()
```

```
← kan met median()
for(method,group) in planets.
 groupby('method'):
 print('{0:5} shape={1:3}'.
 format(method, group.shape)
in .aggregate kan (min, max)
of {'data1':'min','data2':'max'}
filter function
def filt(x):
 return x[data2].std()>4
df.groupby('key').filter(filt)
substract groupwise mean
df.groupby('key').transform(
lambda x: x-x.mean())
apply
df.groupby('key').apply(norm)
def normx(x)
 x[data1]/=x[data2].sum()
 return x
split key
L=[grouping keys, len gelijk an df]
df.groupby(L).sum()
map index to group
df2=df.set_index('key')
mapping={'A':'dick','B':etc
df2.groupby(mapping).sum()
[, kan ook str.lower in]
count discovered planets by
method and decade
dec=10*(planets['year']//10)
dec=dec.astype(str)+'s'
dec.name='decade'
planets.groupby(['method',
decade])['number'].sum().
unstack().fillna(0)
Ook by pivot te gebruiken
als index PIVOT
geeft hetzelfde:
A.groupby(['sex','class'])
['survived'].aggregate('mean').
unstack
A.pivot_table('survived',
index='sex',columns=
'class')
bin age
age=pd.cut(A['age'],[0,18]
A.pivot_table('surv',['sex',age],
'class')
add tax, quantile cut
fare=pd.qcut(A['fare'],2)
in aggfunc kan zo'n dict
{'surv':sum,'fare':'mean'}
margins=True geeft totaal
```

```
extract first names
monte.str.extract('([A-Za-z]+)',
expand=False)
start^ and end$ with consoa.
monte.str.findall(r'^[AEIOU].*
[^aeiou]$')
same as str.slice(0,3)
monte.str[0:3]
last name each entry
monte.str.split().str.get(-1)
split column into two
monte['info'].str.get_dummies('|')
longest ingredient list
recipes.name[np.argmax(recipes.ingr.
                       str.len())]
how many cinnamon as ingr.
rec.ingr.str.contains('[Cc]inna').sum()
is ingredient in the list?
import re
spice=pd.datafr(dict((spice,rec.ingr.str.
contains(spice1,re.IGNORECASE)) for
spice1 in spice_list))
recipe with tomato, fish
spice.df.query('tomato&fish')
recipes.name[].index]
.match, extract, replace, contains,
count                    true of false
```

```
MS 2 oper kolom mean kolom
carsdf.fillna(carsdf.mean())
carsdf.notnull, isnull.mean()
vb data cleaning
carsdf.country.replace('!','')
idem... (r'U.*','USA') → USA
normaliseren, elke waarde min() laten
(df-df.min())/(df.max()-df.min())
2-score
(tbo-tbo.mean())/tbo.std()
MS 3
clean columns=[w.split(':')[0].
replace('_','_') for w in columns]
[4:]+['churn']
churndf=pd.read_csv('file',heade=
None, names=cleancolumns)
omdat True==1
churn.corr()[churn.corr().round(2)==-0.7]
aantal True krijgen
churn[20].str.strip()=='True.').sum())
set to boolean type)
churn[20]=  zonder sum
mele deel is True?
churndf[20].sum()/...  .count()
%. international plan
churn[4].value_counts()/.....sum()*100
crosstab
```

```
pd.crosstab(churn[20], churn[4])/66
churn[4].value_counts()
pd.cut(churndf[7],3,labels=
['low','medium','high']
file naar dict
D=dict()
with open('fib.tsv') as f:
 for line in f:
  LL=line.split('\t')
  D[LL[0]]={i.split(':')[0]:
   i.split(':')[1] for i
   in LL}
nieuw kolom met aatal chaa.
df['new']=df.name.str.capitalize().str.len()
alle recepten zonder breakfast
A=recipe.dropna(subset=['description']
A[~A.description.str.contains(r'[Bb]reakfast')]
load planets dataset
sns.load_dataset('planets')
verst liggende planeet per methode
indices=planets.groupby('method')['distance'].
idxmax().dropna().astype(int).values
planets.loc[indices]
loading in big files
import gzip
with gzip.open('input.gz','r') as fin:
    allines=[]
    for line in fin:
        print 'Regel', c,':',line[:100:10]
        allines.append(line.split(':'))
reader=pd.read_csv('person.csv.gz',
                   sep=';',
                   header=0,
                   encoding='utf8',
                   compression='gzip',
                   skiprows=[1,2,3],
                   index_col=0,
                   na_values='NULL',
                   low_memory=False,
                   usecoles=mycols,
                   chunksize=10000)
df=pd.concat([chunk.dropna(axis=0,
'index', subset=['birthdate']) for chunk
                  in reader])
```

```
chance_dict[b]=dict()
for a in frequent_sections:
 if dummy_df_test[b].sum()>0:
  chance_dict_test[b][a]=dummy_df...[dum...[b]==True][a].sum()/dummy... [b].sum()
 else:
  chance_di...[b][a]=0
nd_chance=pd.dataframe(chance_dict).sort_index(axis=1)
```

```
# iris=iris.drop('species',axis=1)
# y_iris=iris.species
# logistic function
def logfunc(x):
 return 1/(1+np.exp(-x))
```

week 7 HW:

```
gendercount={p:Counter(genderlist[p][:N] for p in genderlist}
ker=pd.dataframe.from_dict(gendercount,orient='index').fillna(0).astype(int)
ker['vrouw']=((ker.female/kr.sum(axis=1))*100).round()
kr.sort_values('vrouw',ascending=False,inplace=True)
D={p:secret[translate[p]][df.loc[p].letels] for p in translate}
A=pd.dataframe.from_dict(D,orient='index')
A.columns=['vrouwen']
df=df.join(A)
df['man']=df.letels-df.Vrouwen
```

```
title=df.loc[df['n_sections'].idxmax()].title
sns.heatmap(churndf.corr())
with open(load_userlang(), encoding='utf8') as f:
i=0
editors=defaultdict(int)
edits=defaultdict(int)
multilingual=... same
edit_by_multilinguals=...
pairs=...
for l in tqdm_notebook(f):
 ll=l.split('\t')
 taken={ll.split(';')[0]:int(ll.split(';')[1]
        for ll in l.split('\t')[1:]}
 if sum(taken.values())>=5 and len(taken)>
  taken={x:Y for x,Y in taken.items() if Y>
  for x in taken:
   if taken[x]>1:
    editors[x]+=1
    edits[x]+=taken[x]
   if len(taken)>1:
    multilingual[x]+=1
    edits_by_multilinguals[x]+=taken[
  for x in taken:
   if len(taken)>1:
    for i in combinations(taken,
     pairs[tuple(sorted(i))]+=1
```

voor waardelijke kans:

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

SOA-test 95% accuraat, toch geen SOA? 1 in 1000
je wil complement van $\frac{TP}{TP+FP}$ dus 1- of $\frac{FP}{TP+FP}$

TP = P(ja, ~~geen~~ SOA) = P(SOA) * P(test ja) = P(ja|SOA) * P(SOA = 0,95 * 0.001

FP = P(ja, geen SOA) = P(geen SOA) * P(ja) = P(ja|geen SOA) * P(geen SOA) = (1-.95) * 0.999

te veel

Pivot tables

df.pivot_table(index='...', columns='...', values='...', aggfunc='...')
↳ je kan ook twee indexen meegeven
↳ aggfunc = mean by default

Hoeveel mannen en vrouwen aan boord?
titanic.pivot_table(index='sex', columns='class', values='survived', aggfunc=sum)

Hoeveel mensen de ramp hebben overleefd per geslacht, per klasse.
① titanic.pivot_table(index='sex', columns='class', values='survived', aggfunc=sum) / titanic.pivot_table(index='sex', columns='class', values='survived', aggfunc=len)
② titanic.pivot_table(index='sex', columns='class', values='survived')

Gemiddelde én mediaan per geslacht van overlevende / niet-overlevende
titanic.pivot_table(index='sex', columns='survived', values='age', aggfunc=[np.mean, np.median])

Aantal 'woord' in 'original' kolom
raw['original'].str.count('woord').sum()

Selecteren van waarde op basis van 2 kolommen
df.loc['kolom 1']['kolom 2'] of df.loc[kolom1, kolom2]

Series maken
pd.series(data='df['...'])

df.sum(axis=0) → som per kolom
df.sum(axis=1) → som per rij

Pandas subsets
df[['col1', 'col2']] columns
df['col1'] single column as series
df.loc['index-of-row'] row
df.loc[[index1, index2]] rows
df.loc[[rows][cols]] rows & columns
df.loc['row1', 'col1'] row & column

Bestand per regel inlezen
```
with open(bestand) as f:
    for l in f:
```

Detective werk
```
Prince = pd.read_csv(bestand)
Prince['text'] = Prince['text'].astype(str)
Prince['text'].str.contains('....').sum())
Prince['text'].str.startswith('....')
Prince['text'].str.findall('....')
```

5 ways to count men and women
```
titanic.sex.value_counts()
titanic.groupby('sex')['survived'].count()
{s: titanic[titanic.sex==s].sex.count() for s in titanic.unique()}
titanic.pivot_table(index='sex', values='survived', aggfunc=len)
pd.crosstab(titanic.sex, titanic.survived).sum(axis=1)
```

Min-max normalization
(df.min()) / (df.max() - df.min())

z-score
(df.col - df.col2).mean() / df.col.std()

Plots
Outliers vinden
df.plot.scatter(x=col1, y=col2, title=...)
Regressielijn → data=df, fit_reg=True
sns.distplot(df)
df.plot(kind=bar)

Missing values
Replace by:
- constant → creation of info
- mean/mode of field → overoptimistic confidence levels
- Random variable → spread will be reduced
  ↳ spread should remain closer to original

Bigfiles
```
import gzip
with gzip.open(bestand) as bla:
    for line in bla:
```

Filling missing values
df[col].fillna(df[col].mean(axis=0)) → mean
df['col'].fillna(random.random())

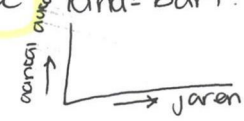
Regex

\n	Newline
[...]	Range or character class
[^...]	not " "
.	Any character except newline
\w	Word character
\W	nonword character
\b	word boundary
\B	non word boundary
i	case insensitive matching
(...)	group subpattern into \1 \2
\d	digit character
\D	nondigit character
\s	whitespace character
\S	non whitespace character
^	the start of the line of text
$	the end of line of text
*	match 0 or more items
+	match 1 or more items
?	match 1 or 0 items
{n}	match exactly n items

Regex methods
re.match()
extract()
findall()
replace()
contains()
count()
split()

Defentie
```
def telderletters(bestand):
    from collections import defaultdict
    telling = defaultdict(int)
    with open(bestand) as f:
        for L in f:
            for c in list(L):
                telling[c] += 1
    t_series = pd.series(telling).sort_values(ascending=False)
    return t_series
```

carsdf.year.value_counts.plot(kind=bar).sort_index



By value_counts plot → sort_index

↳ expliait index
data.loc[0]
↳ implicit index / Python index.

pd.read_csv(bestand, index_col='...')
.set_index()
.sort_values(by=...
df.columns=[....]
↳ rename columns

pd.series:
- value_counts()
- min()
- max()
- sum()
- mean()
- mode()
- median()
- abs()

df[[col1, col2]].corr()
df[col].mean()
df[col].mode()
df[col].median()
df.sort_values
df[col].min/max/sum
.astype(str/float/in

df['kolom'].nlargest(5, columns=...)

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

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

pd.crosstab(col1, col2)

Min-max normalization

$$X^* = \frac{X - \min(X)}{\text{range}(X)} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

↳ is by min waarde gelijk aan 0

z-score

$$X^* = \frac{X - \text{mean}(X)}{SD(X)}$$

↳ is by mean waarde gelijk aan 0.

z-score is niet geschikt om outliers te identificeren omdat mean & SD worden beïnvloed door outliers.

Be carefull with using correlated variables
- at best → overemphasize one data component
- at worst → unstable & unreliable results.

OF /series
.count()

Str methods

len()	startswith()
lower()	endswith()
upper()	isupper()
index()	isnumeric()
strip()	split()

Frederique Henraat
1129265

HEZELBURCHT

Data Science CheatSheet

kolom selecteren
DF[["..","..."]] , DF["..."]

rij selecteren
DF.ix[[...]] , DF.ix["..."]

DF.ix[["..","..."],["..","..."]]
→ kan ook zonder list als enk.

→ put this in pd.Series() to get a series object

Series naar DF => pd.Dataframe({"col1": ser1, "col2": ser2})
dict naar DF => pd.DataFrame(my_dict), key = index
DF.set_index("name")
DF.sort_values(by = "name") | DF.sort_index()
DF.add_suffix(str)
DF.value_counts() => telt hoe vaak waarde voorkomt in Series
DF.describe() => count, mean, std, min, 25, 50, 75, max
↳ categorical series => count, unique, top, freq
DF.plot(x = column, y = column, kind = "") ← plotten!!!
pd.pivot_table(), pd.crosstab(row, column), DF.groupby()
↳ margins = True => column & row with "All"
aggfunc = len => ipv count

get string from Series => .values[index]
remove items from serie1 not in serie2 => serie1[~serie1.isin(serie2)]
get index with max value in Series => serie.idxmax() (also column)

loop door bestand => with open(file) as f:
dataset seaborn inladen => sns.load_dataset("...")
exel file lezen => pd.read_excel
csv file lezen => pd.read_csv("...", sep = "...")

dict.items() => k, v
dict(zip(list1, list2)) <= dict van 2 lijsten
Als index tuple is, naar multi
pd.MultiIndex.from_tuples(index
join 2 DF's DF1.join(DF2)

precision = TP/(TP+FP)

recall = TP/(TP+FN)

intersection = &
union = |
symmetric diff = ^

DF.column.contains(r'') . sum() . str → change to string
. startswith , . endswith \b . blankspace [Ss] . one or the other
. findall list to string: "".join(list)

df.ndim/shape/size/dtype

x[2,0]=12
x[::2]
x[1::2]
x[::-1]
x[:,0] first col
x[0,:] first row

dir(pd)
dir(np)

LOAD DATA

pd.read_csv(loadfile(), sep='\t')
with open(file) as f:
sns.load_dataset() usecols=[]
pd.read_excel()

import re -defaultdict
import tqdm from tqdm import tqdm_notebook

ur[arr.y.2==1]
#extract add nrs.
np.concatenate([x,y,m])
np.vstack([])
np.hsplit()
array + 5 = ufuncs
abs(x)/np.absolute()
np.sum()/min()/max()
np.any()/np.all()
np.argsort()
np.sort(axis=0/1)
np.log2(x)
2log(8) => 2^3=8
np.percentile(series, q=[25, 50, 99])
np.argwhere(ser.y.3==0)

verwijder outliers

df.loc[df[x] < df[x].quantile(0.99)]

plot

plt.hist(name)
df.head(10).plot(kind='barh')
df.plot.scatter(x=, y=, fit_reg=True, style='o')
sns.distplot(np.log(x))
df.plot(kind='line', loglog=True, use_index=False)
histogram => value_counts()

create set of tuples

set(zip(df.x, df.y))

REGEX.

re.match() contains()
extract() count()
findall() split()
replace() rsplit()

word.split('') .title
.startswith .capitalize
.endswith .swapcase
.replace

strip() .islower()
(strip() .isspace()
Rstrip()

.replace('\W', '')

m=re.search(' .*?
m.group html

TRANSFORM

pd.DataFrame(data)
df -> ser: .squeeze .to_frame()
df=pd.DataFrame({'col1': ser1, 'col2': ser2})
df=pd.Dataframe(ser.values.reshape())

pd.crosstab(df.x, df.y).sum(axis=1)
df = pd.Dataframe.from_dict(x, orient='index')
df.name_unique.size

re.sub(x,y,str)
re.split

df.pivot_table(index=
values=
columns=
aggfunc=
[np.mean, np.median])

[] set of chars * zero or more
. any char + one or more
^ startswith {2} exactly 2
$ endswith | either..or

ind A & ind B = intersection
A|B = union
A^B = symmetric difference

data['a': 'c']
data[['a','c']]
data.loc[data.x>2, ['y','c']]
.index[0]

# relatie tussen x en y
pd.crosstab(df.x, df.y)

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

df.dropna(axis='columns')
df.fillna(0)
pd.concat([x,y])
df1.append(df2)
pd.merge(df1, df2, on='employee', left_on='emp.', right_on='emp', left_index=True, suffixes=['_b','_a'])
df1.join(df2)

with open(file) as f:
c=0
e=defaultdict(int)
for l in tqdm_notebook(f):
....
c+=1.

df.column.contains(r"\b").sum

aggfunc(len)
ipv count.

string from series:
.values[index]

titanic.dropna(subset=['age'])

serie.idxmax()

AGGREGATIONS na groupby..

count() min() max()
first() std() var()
last() mad() prod()
mean()
median() sum()

df.groupby('key').aggregate(['min','max'])
({data1: min, data2: max})

margins=True
↓ alles

first row:
iloc[0]

.filter(filter_function)
# drop data based on
.apply(function)

r'\b[A-Z0-9]'

prince.text.str.contains(r'\b[Ss]ex\b').sum

percentage klinkers:
prince.text.str.findall(r'[AEIOUYaeiouy]').str.len().describe

Readfiles | help(pd.Series.loc); dir(pd/np); dur(str);

```
df = pd.read_csv('file.csv')
     pd.read_excel('file.xlsx')

with open('file') as file:
    for page in file:
df.sort_values(by='col')
df.sort_index()
with open('file') as f:
    for line in f:
        regels = line.split('\t')
        languages = {i.split(',')[0]: int(i.split(',')[1])
                     for i in regels}
        if sum(languages.values()) >= 5 and len(languages) <= 5:
            languages = {language: value for
                         language, value in languages.items()
                         if int(value) >= 2}

default dict
for language, value in languages.items():
    editors[language] += 1
    edits[language] += value

pd.dataframe.from_dict(dict, orient='index')
samenvoegen = pd.concat([dataf1, dataf2], join='inner', axis=1)
merge = df1.merge(df2, how='inner'...)
```

Regex findall; match; extract;

```
df.text = prince.text.astype(str)
bevat sex → prince.text.str.contains(r'\b[Ss]ex\b')
begint met sex → prince.text.str.contains(r'^[Ss]ex\b')
of ...startswith('sex').sum()+...startswith('sex').sum()
Songs met woorden waar sex in voorkomt
ex = prince.text.str.findall(r'[Ss]ex\w+')
ex[ex.str.len()>0] → laat zien of .count()
Counter([s for i in ex.values for s in i])
woorden met letter en cijfer erin
... findall(r'\b[A-Za-z0-9]*[0-9]+[A-Za-z0-9]*[A-Za-z][A-Za-z0-9]*\b
|\b[A-Za-z0-9]*[A-Za-z][A-Za-z0-9]*[0-9][A-Za-z0-9]*\b')
klinkers vinden: findall(r'[AEIOUYaeiouy]')
```

```
df.replace(1, one
df.dropna(subset=...)
df.describe()
df.corr(col1, col2)
df.min/max/std
S['col'].str.lower()
df.head  str.replace
df.tail
df.shape
df.info
```

. → any char except line break
\w → word char, digit, _
\s → whitespace
\d → digit 0-9
\. → period
$ → end of str
^ → start
| → either or
+ → one or more
* → zero or more
? → once or none
\t\n\r → tab, newline, return
[abc] → a/b/c [^ABC] → alles behalve
re.sub(reg, vervanging, string)

selecteren |

```
df[df['col'] = 'voorwaarde'].sort_values('col1', ascending(True)
df.loc[row, col] → df[:, col] of gewoon df['col']
integer: df.iloc[0] → df.iloc[[0]]
by label → df.loc[[0], 'colname']]
by label/posit → df.ix[[index], 'colname']
bool index → s[~(s>1)] = Series s where val is not >1
s[(s<-1) | (s>2)] = s where val is <-1 or >2
df[df['Population'] > 1200000] → Filter df
df.loc[:, (df.any() | .all())] → sel col any / sel col all
```

from pandas.util.testing import
test if equal → assert_frame_equal(df1, df2)
stack() → level 1 → stack col to row
level 0 → upper cols stack to row
unstack() → row to col
crosstab
pd.crosstab(df.col1, df.col2) zelfde als
groupby(['col1','col2'])[col2].count().unstack().fi
grouping → pd.crosstab(df.col, [df.col1, col2])
query → df.query("col1 < col2 < col3")
↳ and, or; col1 in [val, val, val]

tellen | len(df[df['col'] == 'value']) | df.groupby('value').size()
count_values() | df.pivot_table(index, val, aggfunc='len')
dictcom = {x: df[df.col = x].col.count() for x in df.col.unique()}
of meer voorwaarden Pd.Crosstab(df.col1, df['col2'])
ext class sex class
per df.groupby(['sex', 'class'])['class'].count().unstack()
hoeveel deel overleefd van sex per class
titanic.pivot_table(index='sex', columns='class', values='survived'
gem en medi van gelacht van overlevers- niet overlevers
titanic.pivot_table(index='sex', columns='survived', values='age', aggfunc=[np.mean, np.medi..])
rijen met nan eruit:
df = titanic.dropna(subset=['age'])

concat([df1, df2], axis=0,1)
append: df1.append(df2)
Pd.merge(df1, df2)
merge on index → left_index=True

```
error plot vrg 4 w 7
error = df.Percentage - df.Percentage laag
percs = df.Percentage
percs.plot.bar(yerr=error)
```

```
def teldeletters(bestand):
    from collections import defaultdict
    telling = defaultdict(int)
    with open(bestand) as f:
        for l in f:
            for c in list(l):
                telling[c] += 1
    tellingseries = Pd.Series(telling).sort_values()
    return tellingseries
teldeletters('bestand').plot(kind='bar', figsize(15,7), logy=true);
```

pivot
Pd.pivot_table(df, values, index, columns, fill_value, aggfunc=np)
→ spredd rows into columns
aggfunc applies to value, columns are optional → segment

```
gemiddelde aantal woorden per sectie
mean = (df['n_words'] / df['n_sect'])
log = np.log2(mean)
plot = sns.distplot(log)
```

```
boven 20
df.loc['col'['x'] > 20]
gem zonder nan df.mean(skipna=T)
gender count
{p: Counter(genderlist[p][:N] for p
            in genderlist}
PVV 25 zetels, hoeveel vrouw
Counter(mvdict['PVV'][:25]['female
```

Series.index
Series.values
DataFrame.columns
data.items() → tuples
data['a':'b'] → includes b.
data[['a','b']]

Index as ordered set:
x = 1,3,5,7,9
y = 2,3,5,7,11
→ x & y : 3,5,7
→ x | y : 1,2,3,5,7,9,11
→ x ^ y : 1,2,9,11

data.loc[1:3] → index-based
data.iloc[1:3] → python-based
data.values[0] → first col.
data.loc[data.dev>100, [x,y]]
a.add(b, fill_value=0)
→ sub(), mul(), div(), mod(), pow()
• notnull() → all non nan values.
• dropna / .fillna().
MultiIndex.unstack() → back to df.
df.set_index([arr]) → multi index
df.append(df2)
pd.merge(df1, df2, on='..')

• query("query") → @var
• sort_values() by=
• mean(axis='columns') → mean over rows
• count() → number of items.
• pivot_table('survived', index='sex', columns='class')
• pivot_table(index='sex', columns='class', aggfunc='sum').

col.str.
• match → bool
• extract → matched groups
• findall → list
• replace
• contains → bool
• count → occurences of pattern.
• split
• rsplit
• get_dummies → indicator var split.

• sort_index()
• sort_values()
crosstab
hist
df.col.value_counts().plot.barh().

df.iterrows()
pd.merge
df.col.quantile
df = pd.DataFrame.from_dict(dict, orient='index', columns=[])

REGEX
^ start
$ end
\w \d \s
\W \D \S
[abc] [^abc]
[a-z]
\t \n
(?=abc)
(?!abc)
* 0+
+ 1+
? 0/1
?? lazy
\b end of word
re.sub
re.findall
re.match
re.search

~ → not

defaultdict
↳ from collections

elem.findtext
f = archive.open(file
root = etree.parse(f).getro
node.find
node.findall

BS.find('').text
BS.findall

5 ways:
• value_counts()
• groupby('')[''].count()
• pivot_table aggfunc=len
• crosstab().sum(axis=1)

df.col.str.contains().sum()

str.replace(old, new)

Counter.most_common()

index_col sep.

filteren met bools:
df[filters.values]

pd.read_excel(file, usecols='A:B')

df.sort_values(index) ... week 5

```
(df.col1 == 'test').sum()   True False
df[(col1 == 'test') & (df.col2 > 5)]   DF
df[['col1', 'col2']]   DF of 2 cols
df.values   DF in list without index
list.items   for error 'to many values to unpack'
df.sort_values('col1')
pd.Series(df.col1, index=df.index)
df.drop(columns=['coly','colx'], axis=1)
pd.DataFrame.from_dict(dict1, orient='index', columns=['col1'])
pd.concat([dict1, dict2], axis=1, join='inner')
df1.join(df2)
((((abs(df[col1] - df[col2]))/df[col1]).mean()*100)   mistake %
```

df.mean() / df.median() / df.mode() / df.count() / df.first() / last() / min() / max() / std() / var() / mad() / prod() / sum()

week 6

```
df.loc?
df.col1.quantile(q=0,99)
df_no_na = df.dropna()
```

week 7

```
df.fillna(0, inplace=True)
df.col1 = df.col1.astype(int)
df['Percentagecol'] = round((df.col1/(df.col1+df.col2)*100),0)
df['mannen'] = df.zetels - df.vrouwen
```

Tentamen 1

```
1) df = pd.read_csv('xxx.csv', index_col='names')
   df.sort_index(inplace=True)
2) df = df.fillna(0)
3) df['assmean'] = df[['ass1', 'ass2']].mean(axis=1)
4) (df[['ass1','ass2']]>9).sum().sum()
5) df['wmean'] = 0,4*df[['assmean']] + 0,6*df[['examen']]
6) z_df = (df - df.mean()) / df.std()
7) df.sort_values(['col1', 'col2'], ascending=(False, True))
8) corr = df[['col1', 'col2']].corr()
   corr = corr[corr != 1]
   rij = corr.max().idxmax()
   kolom = corr.loc[rij].idxmax()
   vraag = (rij, kolom)
```

Rec: TP/(TP+FN)
Pres: TP/(TP+FP)

	soa	geen soa
voorsp w	TP	FP
voorsp n	FN	TN

Hoofdstuk 3

```
df.loc[:3, :2]
df.isnull()
pd.merge(df1, df2)
df.unstack()
df.groupby('col1').aggregate([min, np.median, max])

titanic.groupby(['sex', 'class'])['survived'].aggregate('mean').unstack() == titanic.pivot_table('survived', index='sex', columns='class')
titanic.pivot_table('survived', index='sex', columns='class')
age = pd.cut(titanic['age'], [0, 18, 80])
titanic.pivot_table('survived', index=['sex', age], columns='class', margins=True)   margins gives totals
births.pivot_table('births', [births.index.month, births.index.day])
ser = pd.Series(['Jan Smit', 'John Haas'])
ser.str .lower.len.strip.startswith.split.get.capitalize
```

Oefententamen

```
1) titanic.sex.value_counts()
 - titanic.groupby('sex')['survived'].count()
 - {s: titanic[titanic.sex == s].sex.count() for s in titanic.sex.unique()}
 - titanic.pivot_table(index='sex', values='survived', aggfunc=le...)
 - pd.crosstab(titanic.sex, titanic.survived).sum(axis=1)
2) titanic.groupby(['sex', 'class'])['survived'].sum(axis=1)
   titanic.pivot_table(index='sex', values='survived').mean().unstack()
3) titanic.groupby(['sex', 'survived'])['age'], columns='class', aggfunc='m...
4) pd.crosstab(titanic.pclass, titanic.class).aggregate([np.mean, np...
5) test = titanic.dropna(subset=['age'])
   test = test[test.sex == 'male']
   pd.crosstab(test.age, test.adult_male)

6) bestand = 'prince_lyrics.csv'
def teldeletters(bestand):
    from collections import defaultdict
    telling = defaultdict(int)
    with open(bestand) as f:
        for l in f:
            for c in list(l):
                telling[c] += 1
    tellings = pd.Series(telling).sort_values(ascending=False)
    return tellings
teldeletters(bestand).plot(kind=bar, logy=True);
7) prince = pd.read_csv(bestand)
prince.text = prince.text.astype(str)
```

A: hoeveel nummers bevatten "sex"

```
prince.text.str.contains(r'\b[Ss]ex\b').sum()
```

B: hoeveel nummers beginnen met 'sex'

```
prince.text.str.startswith('sex ').sum + lowercase
```

C: hoeveel nummers bevatten sexy etc?

```
sex = prince.text.str.findall(r'[Ss]ex\w+')
print(sex[sex.str.len()>0].head(10))
print(sex[sex.str.len()>0].count())
```

D: print all die woorden uit

```
from collections import Counter
print(Counter([s for l in sex.values for s in l]))
```

E: U2 4U etc

```
ef = prince.text.str.findall(r'\b[A-Za-z0-9]*[0-9]+[A-Za-z0-9...
print(ef[ef.str.len()>0].head(10))
print(Counter([s for l in ef.values for s in l]))
```

F: % klinkers per track

```
(100* prince.text.str.findall(r'[AEIOUaeiouyY]').str.len() / prince.text.str.len()).describe()
```

Regex

[abc] anything abc | [^abc] anything not abc | a-zA-Z0-9
\s whitespace | \d digit | \w word | \b word boundary |
( ) capture | ^ start of string | $ end of string |
a? zero or one | a* zero or more | a+ one or more

A = alcoholist B = man
$P(A \cap B) = 0{,}0225$
$P(A|B) = \frac{P(A \cap B)}{P(B)}$
Conditionele kans $= \frac{0{,}0225}{0{,}5} = 0{,}045$

$\frac{x - \min(x)}{\max(x) - \min(x)}$ Min-max normalization

$\frac{x - \text{mean}(x)}{\text{STD}(x)}$ Z-score normalization

extra

```
df.Percentage.plot('bar', yerr=(df.Percentage-df.Perc.bag).tolist())
r = (Q3 - Q1)*1,5
Q1 - x en Q3 + x = data
df.col1.fillna((df.col1.mean()), inplace=True)
for b in freqsec:
    chancedict[b] = dict()
    valuedict = {}
    for a in freqsec
        numab = len(dummydf[dummydf[b]==1] & (dummydf[a]==1))
        valuedict[a] = numab / len(dummydf[dummydf[b]==1])
    chancedict[b] = valuedict
condchance = pd.DataFrame(chancedict).sort_index(axis=1)

freqsec = sectietitels(df).nlargest(20).index
dummydf = df.secties.apply(pd.Series).stack
freqsec = [x for x in freqsec.values]
dummycols = pd.DataFrame(columns=freqsec)
for col in freqsec
    dummycols[col] = dummydf.
dummydf = dummycols == freqsec
dummydf = dummydf.groupby(level=0).sum() == 1
chancedict = dict()
```

re.findall   re.sub
re.split   re.match if beginning with (T,F)

NaNs: · replace with constant
* replace with mean/median
with random
imputed value
* changes center and spread
Misclassifications: value counts
outliers: histogram, boxplot, scatterplot
center measures: mean, mode, median
Variability: range (max-min) STD, MAD, IQR
normalization (reduce variation):
· min-max: $X^* = X - min(x) / range(x)$
alles 0-1.
· Z-sc. $Z = (x - mean(x)) / SD(x)$
< mean - > mean +
Kan leiden tot n-verd, if ! skewed.
· Decimal: $X^* = X / 10^d$ → d = decim. max(x)
· Skewness: $\frac{3 \cdot (mean - median)}{SD}$
right mean > median
Transformation (reduce skewness)
· log, $\sqrt{x}$, $1/\sqrt{x}$
outliers: if $z_{sc} > 3$ < -3
· Z-score: [illegible]
· IQR: Q1: 25% Q2: 50 (med)
Q3: 75%
IQR = Q3 - Q1 ↗ Q1 - 1.5 IQR, Q3 + 1.5 IQR
outlier if x located
1.5 IQR below Q1, above Q3
num → cat: binning (K-means e.g.)
cat → num: dummy var.
unsupv.: Clustering/grouping
reduceren complexit.
supv: regressie, classificatie
3: reinforcement learning
evaluatie: vergl. met gouden std.
class.: Acc., Prec, Rec, MRR ...
Regr.: (R)MSE, AME ...
Acc: how many predicted were correct?
Prec. how many pos. pred were corr TP/TP+FP
Rec. how m. pos pred. as pos? TP/TP+FN

	ziek +	−
Test +	TP	FP
−	FN	TN
	alle ziek	gezond

Train, Val, test
weinig data: K-fold
· men/women titanic:
t.sex.value_counts()
T.groupby('sex')['survived'].count()
t.pivot_table(i=sex, v=surv., aggf=len)
d.crosstab(t.sex, t.surv).sum(axis=1)
· per geslacht / per klasse / deel surv.
t.p.t(i: sex, v: sur, c: class)
t.groupby(['sex', 'class'])['survived'].agg('mean')

· gem./med. leeftijd surv = 1
t.p.t(i=sex, [illegible],
c = surv. val: age, aggf [np.mean, np.median])
· tel de letters:
```
from coll.. import defaultdict
with open(file, encoding=utf8) as f:
    for l in f:
        for c in l:
            tel[c] += 1
    tel = pd.Series(tel).sortv...
    Return tel
```
· Regex: .str.match → bool
.extract → strings
. Findall .replace
.contains → returns bool
.count → count occ. of patt.
[abc] inv: [^ABC]
\s: whitespace \d: digit
\w word (a|b) a or b
a?: 0/1 a
a*: 0> a a+: 1> a
a{3:3 ^ start of str
$ end of str \b wordbound.
*? non greedy
· n numers _sex_
Prince.txt.str.contains(\b[Ss]ex\b).sum()
· n numers starts with _sex_
.contains(^[Ss]ex \b)
· n numers bevat _sex_:
.contains([Ss]ex\w+).sum
· Print alle woorden die _sex_ ...
p(Counter(x for y in ... for x in y))
· Perct. klinkers:
findall(aeiouy).str.len() / Pr.txt.str.len()
· (Rij, kolom) · Axis 0: within each col.
1: each row.
Langste str: 1. lang = str()
2. pak regel, meet len() vergl. len(lang)
3. if >, lang = 2 → next
· Age = pd.cut(titanic.age, [0, 18, 80])
t.p.t(surv, ['sex', age], 'class')
· corr: df1.corr(df2)
· Jaccard $\frac{A \cap B}{A \cup B}$ → $A \cup B = (A+B) - A \cap B$
· conditionele kans $P(A|B)$
$= \frac{P(A \cap B)}{P(B)}$ → waarzowel A als B is / → totaal aantal B's

is kans dat A plaats vindt, gegeven dat B ook ..
$\frac{P(hoog\ sal.) \cdot P(vrouw)}{P(vrouw)}$
regressie: sns.lmplot(x, y, data, Fitreg=T
df → multi.indx → set_index([col1, col2],
· Concat: row-wise default
Short: df1.append(df2)
· Merge: join autom. on key or colls.
merge(df1, df2, on = 'col1')
left on, right on ...
default on indexes → df1.join(df2
· Pivot_tables(values, index, colls, agg
· groupby.aggregate(col1: mean, col2: ma
pd.qcut → quantiles
Pivot_tables can be multileveled.
Rgex: re.findall(regex, text) → list
.match → determine if begin of string
.split → list
.sub → replaces
f_s = sectietitels(sf).nlargest(20)
dummy_df = df.sections.apply(pd.Series).stack
cols = [x for x in frequent_sections.value
df2 = pd.DF(columns = cols)
for x in f_s:
df2[x] = d_df
Bool = df2 = freq-sec
dum_df = boolean.gb(level=0).su
chance_dict = dict()
for b in frequent_sections:
temp_dict[a] = len(d_df[(d_df[b] == 1) & (d_df[a] == 1
/ len(d_df[d_df[b] == 1
chance_dict[b] = temp_d
cond_chance = pd.DF(c_d).sort_index(ax=1)
cond_chance.