```
mean = sum(xs)/len(xs)
var = sum( xs - np.mean(xs)**2)/len(xs)
std = math.sqrt(var)
```

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

Normalize data = z = $\frac{x - \min(x)}{\max(x) - \min(x)}$

& A ∩ B = intersection. A x B

| A ∪ B = union, of A, of B

P(A∩B)/P(B) = voorwaardelijke kans op A

```
def gcd(x,y):
    while y!=0:
        (x,y)=(y,x%y)
    return x

def ggcd(a,b):
    return (abs(a*b)/gcd(a,b))
```

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

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

`pd.cut(df.age,[0,18,30])`

of `qcut(df.age, 2)`

`.str.strip()` `.str.split()`

from collections import defaultdict, Counter

`get_dummies()`

`monte.str.get_dummies('|')`

monte = `['B|C|D', 'B|D']`

. any char except new line

\w matches alphanumeric chars: [a-zA-Z0-9_]

\W matches non-alphanum chars: [^\w]

\d matches digit: [0-9]

\D matches non digit: [^\d]

\s matches whitespace char [\t\n\f\r\p{Z}]

\S matches non-whitspace char [^\S]

[] Character class. Matches any charachter contained between the square braces

[^] Negated Char. matches any char that is not contained between the square braces

* matches 0 or more repetitions of preceding symbol

+ matches 1 or more repititions of preceding symbol

? makes the preceding symbol optional

{} Braces. matches at least 'n' but not more than 'm' reptions of preceding sym. {n,m}

(xyz) Char group. matches the chars xyz in that exact order.

| Alternation. matches either chars before ?or after the symbol

\ Espape next character. u can use: [](). * ? ^ $

^ matches the beginning of input

$ matches the end of input

TN = is neg, predic neg

TP = is pos, predic pos

FN = is pos predic neg

FP = is neg, predic pos

match()
extract()
findall()
replace()
contains()
count()
split()

```
pd.crosstab(df.sex, df.survived, margins=True)
df.pivot_table(c='class', i='sex', v='survived', afunc='count'
Counter(df.sex) OF df.sex.value_counts()
aggfunc={'survived':sum, 'fare':'mean'}
```

recall = pakken

sep

```
titanic.groupby(['sex','class'])['survived'].aggregate('mean').unstack() = .pivot('survived', index='sex', columns='class')
```

df['col'].str.count('word').sum()

1. df.col.value_counts()

2. df.groupby('col1')['col2'].count()
df.group(func, axis=0).mean()
df.groupby(['col1','col2'])['col3'].mean()

3. {s: df[df.col==s].col.count() for s in titanic.col.unique()}

4. df.pivot_table(index='col1', values='col2', aggfunc=len)

5. pd.crosstab(df.col1, df.col2).sum(axis=1)

2. value van col1 in value col2 by value col3
df.pivot_table(index='col1', columns='col2', values='col3', (aggfunc=sum/len))

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

df → df.pivot(index='foo', columns='bar', values='baz')

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Stacked → stacked.unstack()
multi index → -1 index
.unstack(colname)
df.unstack().stack() = df.sort_index()

pd.crosstab(df.col1, df['col2']) | df.iloc[0] = mijn waardes

3. mean and median of age per sex for col survived:
df.pivot_table(index='sex', columns='survived', values='age', aggfunc=[np.mean, np.median])

4. Relatie tussen 2 columns → pd.crosstab
pd.crosstab(df['col1'], df[col2])

5. pd.testing.assert_frame_equal(df.query("..."), df[df.col])

6.
```
def tel(file):
    telling = defaultdict(int)
    with open(file) as f:
        for letter in f:
            for c in list(letter):
                telling[c] += 1
    tseries = pd.Series(telling).sort_values(ascending=False)
    Return tseries
tel(file).plot(kind='bar', figsize=(....), logy=True);
```
telt frequentie van elk letter in file
→ maak pd series

7.
```
df = pd.read_csv(file)
df.col1 = df.col1.astype(str)
df.col1.str.contains(r'^[Ss]ex\b').sum())
df.col1.str.startswith('sex ').sum()
ef = df.col1.str.findall(r'[Ss]ex\w+')
ef [ef.str.len()>0]
Counter([s for l in ef.values for s in l])
```

numpy:
np.count_nonzero
df[col].to_dict(

5.
```
pd.read_csv(loadfile(), sep='\t')
df.loc[df[col] == 'value'] → alleen rijen met bepaalde value uit één column
df.sort_values(by='column', ascending=False)
set(tuple(zip(df.col1, df.col2)))
df1.join(df2, how='right', rsuffix="my")
```
df.drop([col', 'col2'], axis=1)

7.
```
pd.read_excel(file).set_index('col')
soup = BeautifulSoup(open(file)).read(), features="xml")
soup.findAll('string')
for key, value in dict.items():
    col.append(dict[key][N].count('string'))
```

*Importing Data

Pd.read_csv(filename)
Pd.read_html(url)

*Exporting Data

df.to_csv(filename)

*Inspecting data

df.head()
df.tail()
df.shape()
df.info()
df.describe()
s.value_counts(dropna=False)

*Selection

df[col] → series
df[[col1, col2]] → df
s.iloc[0]
s.loc[index]
df.iloc[0,:] → first row
df.iloc[0,0] → first elem of first row

*Data Cleaning

df.columns = ['a','b'] (Rename col)
Pd.isnull() & Pd.notnull()
df.dropna() (rows) df.dropna(axis=1) (columns)
s.fillna(x) or s.fillna(s.mean())
s.astype(float)
s.replace(1, 'one') or s.replace([1,3],['one','three'])
df.set_index('column_name')

*Filter, sort and Groupby

df[df[col] > 0.5] Rows where col > 0.5
df[(df[col] > 0.5) & (df[col] < 0.7)]
df.sort_values(col), ascending = True/False
df.groupby(col)
df.pivot_table(index=col1, values=[col2, col3], aggfunc=mean
df.apply(np.mean) col or df.apply(np.max, axis=1) row

*Join / Combine

df1.append(df2) rows
Pd.concat([df1, df2], axis=1) columns

Oefentoets

1. 3 manieren om te ontdekken hoeveel mannen en vrouwen aan boord waren
   - titanic.sex.value_counts()
   - titanic.groupby('sex')['survived'].count()
   - titanic.pivot_table(index='sex', values='survived').sum(axis=1)

2. How many survived for each gender
   (titanic.pivot_table(index='sex', columns='class', values='survived', aggfunc=sum) /
   " " " " " " " " " " = len)
   Pd.crosstab(titanic.sex, titanic['class'])

3. Gem en mediaan leeftijd overlevenden en niet overl.
   titanic.pivot_table(index='sex', columns='survived', values='age', aggfunc=[np.mean()...

4.
```
def teldeletters(bestand):
    telling = defaultdict(int)
    with open(bestand) as f:
        for l in f:
            for c in list(l):
                telling[c] += 1
    telling series = pd.Series(telling).sort_values(ascending=False)
```
⇒ teldeletters(bestand).plot(kind='bar', figsize=(15,7), logy=True)

Precision = TP/TP+FP | Recall = TP/TP/FN

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=len)

pd.crosstab(titanic.sex, titanic.survived).sum(axis=1)

	predicted	
actual	TN	FP
	FN	TP

titanic.sex.value_counts() (Precision = TP/TP+FP F1 = 2x prec)

(titanic.pivot_table(index='sex', columns='class', values='survived', aggfunc=sum) /

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

pd.crosstab(titanic.sex, titanic['class']) (F1 = 2x (precision x recall) / precision + recall)

Titanic.pivot_table(index='sex', columns='survived', values='age', aggfunc=[np.mean, np.median]) —— Relatie tussen Pclass en Class → pd.crosstab(titanic.pclass, titanic['class']) → T = titanic.dropna(subset=['age']) [abs(df.wordcount - df.my)]

Wanneer is adult male waar? (True) → T.pivot_table(index='age', columns='sex', values='adult_male', aggfunc=sum)['male'] → pd.testing.assert_frame_equal(T.query('sex == 'male' and age >= 16'), T[T.adult_male]

teldeletters functie (prince.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 ← tellingseries = pd.Series(telling).sort_values(ascending=False) — return tellingseries — teldeletters(prince.csv).plot(kind='bar', figsize=(15,7), logy=True); || prince = pd.read_csv(bestand)

prince.text = prince.text.astype(str) # prince.text.str.contains(r'\b[Ss]ex\b').sum() ### of prince.text.str.startswith('Sex').sum() + prince.text.str.startswith('sex').sum() || ef = prince.text.str.findall(r'[Ss]ex\w+) — print(ef[ef.str.len()>0].head(10)) — print(ef[ef.str.len()>0].count()) #4: print alle woorden wat die 'sex' bevatten → from collections import Counter print((Counter([s for l in ef.values for s in l]))) #5 o dingen die woorden & cijfers bevatten: → ef = prince.text.str.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') print(ef[ef.str.len()>0].head(10)) — print(Counter([s for l in ef.values for s in l])) || # percentage klinkers = (100* prince.text.str.findall(r'[AEIOUYaeiouy]').str.len() / prince.text.str.len()).describe() Normalise = lijst(df - df.min()) / (df.max() - df.min())

Znormalise = (T60 - T60.mean()) / T60.std() pd.merge(df1, df3, left_on="employee", right_on="name").drop('name', axis=1) .apply[lambda x: x[0]]

drinks.groupby('continent').beer ...

```
df.groupby(column)[column].max()   "nearest"
df.sort_values(column, ascending=False, inplace=True)
df.column.value_counts()   "how many"
df.groupby(column)[column].count()   "how many"
df.pivot_table(index=column, values=column, aggfunc=len)
pd.crosstab(df.column, df.column).sum(axis=1)
df[column].sum()   "column sum"   Fill_value=0
df.sum()"series with column name and sum"   aggfunc=[np.mean, np.median]
df.sum(axis=1)"series per row sum"
df[column]=...   "define new column"   .std()"standard dev"
df.corr  df.quantile(0.99)   (df[df<q]).corr()
df.stack()  df.unstack()   "per column, per index"
df.loc[:,(df>1).any()]   "columns with any values>1"
df.loc[:,(df>1).all()]   "columns with only values>1"
df.set_index([column,column])   "Multi index"
```

---

		true condition positive	true condition negative
predicted condition	positive	TP	FP
	negative	FN	TN

Precision = TP / (TP+FP)

Recall = TP / (TP+FN)

Accuracy = TP+TN / all

F = 2 * (precision * Recall) / (precision + Recall)

```
.                any char except newline
a                character "a"
ab               string "ab"
a|b              a or b
a*               0 or more "a"'s
\                escapes a special char
+                1 or more
?                0 or 1
{2}              exactly 2
{2,5}            between 2 and 5
{2,}             2 or more
{,5}             up to 5
[a-d]            1 char between a/d
[^a-d]           char except a/d
\d               one digit
\s               one whitespace
\w               one word
```

```
.contains(R"a")
.findall(R"a")
.startswith(R"a")
```

```
df.values[0]   "to get row"
df[column]   "to get column"
df.iloc[:3, :2]"first 3 rows, 2 columns"
pd.read_csv("file")
pd.read_excel("file", index_col=1)
df1 & df2   "intersection, zellfolge"
df1 | df2   "union, alles"
df1 ^ df2   "verschieden"
df.dropna()   "filtered data"
df.fillna()   "filled null data"
df.merge()   "merge by column"
df.join()   "join by row"
df.set_index(column, inplace=true
pd.DataFrame.from_dict(d.it, orient
index).fillna(0).astype(int)
df.drop([value, value])   "drop from row
df.drop(column, axis=1)"drop from column
```

Regex

^ - start of string

$ - end of string

\s - whitespace

\d - digit

\w - word

pd.read_csv()

- .contains()
- .startswith()

pd.pivot_table()

- pd.crosstab()

From collections import Counter

* - 0 or more

\+ - 1 or more

? - 0 or 1 Counter(

. - any character except new line

(a|b) - a or b

(...) - group

[abc] - range

[^abc] - not a or not b or not c

[a-z] - lower case

[A-Z] - upper case

[0-7] - digit from 0 to 7

{3} - exact 3

5 manieren mannen en vrouwen

```
Titanic.sex.value_counts()
Titanic.groupby('Sex')['Survived'].cou
dict(Titanic.loc[:,'Sex'].values())
pd.crosstab(Titanic.sex, Titanic.survived,
            .sum(axis=1)
```

Pandas

.drop(['a'], axis=1, inplace=True)

axis=0 kolom
axis=1 rij

Pd.Read_CSV() sep='\t' index_col skiprows=1

Rename columns: df.columns = ['a','b','c']

Select: df[[df.Kolom == 'value']]
df.loc[[df.kolom == 'value')& ()]

Toevoegen: df['naam'] = df.b/df.a

Index: df.set_index('naam').index

Sorteren: df.sort_values(['kolom', 'kolom'], ascending=['false', false]

Tellen: df.value_counts('Kolom')
~~waardoor~~ tellen van kolom, hoe vaak aantal komt iets voor?

Boolean mask: X[X % 3 == 0]

samenvoegen: pd.merge(x,y) on= how='innee'
pd.concat([x,y], ignore_index=True), join=inner)

x.append(4) df1.join(df2)^Rsuffix

1-1, many-1: 1/2 heeft duplicaten, many-many

Groupby: df.groupby('key').sum()

	key	data
0	A	0
1	B	1
2	C	2
3	A	3
4	B	4
5	C	5

key	data
A	3
B	5
C	7

df.describe()

df.groupby('Species')['length'].max()
* geeft grootste waarde van kolom lengte met specie erbij
What are the max lengths for each specie?

pivot: pd.pivot_table(df, index='sex', columns='class') aggfunc=np.sum/len values

class / sex	first	second	third
female	0,9	0,4	0,1
male	0,7	0,5	0,1

Plotten: df.plot(kind='barh') x = df.kolom

aanpassen: df.kolom = df.kolom.str.lower()

nan: df.fillna()
df.dropna()
df.isnull() } Boolean
df.notnull() }

meerdere kolommen
df[['kolom', 'kolom']].sum

type
df.kolom.dtype = str

Pd.crosstab(df.kolom, df.kolom)[lijst] margins=True

Rename: df = df.rename(translate, axis='index', inplace=True)
5.replace('i', 'one')

Margins=True wanneer je ook alluit

Literatuur

datum
datetime -> df.kolom.to_datetime()

```
import gzip
with gzip.open('input.gz','R') as in:
    for line in fin:
        print(line)
```

5 verschillende manieren om aantal mlf te tellen
- tit.sex.value_counts()
- tit.groupby('sex')['survived'].count()
- {s: tit[tit.sex == s].sex.count() for s in tit.sex.unique()}
- tit.pivot_table(index='sex', values='survived', aggfunc=len)
- Pd.crosstab(tit.sex, tit.survived).sum(axis=1)

Hoeveel nummers bevatten het woord sex?
prince.text.str.contains(r'\b[Ss]ex\b').sum())

Hoeveel ~~beginnen~~ beginnen met dat woord?
prince.text.str.startswith('Sex ').sum()

dataframe van dic: dic={'Party': party,
df = pd.DataFrame.from_dict(dic) {'male', man}

gemiddelde aantal van kolom
df.kolom.mean()

meer dan mediaan
len(df.loc[df.kolom > df1])

~~uit alle kolommen rijen~~
uit alle kolommen values een .halen
df.kolom.str.replace('.','')

head printen van een boel files
!head ../../Data/db/chuven.all
!cat ../../Data/db/chuven.names
chuvendf = pd.Read_csv('../../Data/db/chuven.all', (zijst)

%ls ../../Data/db/chuven*

soup = BeautifulSoup(open(loadlijst()).Read(), f.features='xml'
print(soup.Prettify())
Partijen = [party.text for party in soup.findall('RegisteredName')]
genders = [[j.text for j in i.findall('Gender')] for i in soup.find_all('Affiliation')]

aantalmannen = []
for i in genderlist.values():
    aantalmannen.append(i[:30].count('male'))

Plotten van meerdere kolommen op a-z volgorde
df[[kolom,kolom]].sort_index().plot()

Selecteer alle kollommen behalve 1
df.loc[:, df.columns != 'kolom']

hoevaak komt iets voor in een kolom
~~if df.kolom == 'string'~~ df.kolom.str.contains(' ').sum())

iets droppen kolom die index is gemaakt
df.drop(kolom, inplace=True, axis=1)

```
genderlist = {}
for i in range(len(partijen)):
    if partijen[i] in genderlist:
        genderlist[partijen[i]].append(genders[i])
else:
    genderlist[partijen[i]] = genders[i]
```

Kolom toevoegen hoeveel vrouwen in kamer worden verwacht

```
dicv = {}
for party in df.index:
    for zetel,vrouw in secret[party].items():
        zetels = int(df.loc[df.index==party].zetels.values)
        if zetel == zetels:
            dicv[party] = vrouw
df['vrouwen'] = dicv.values()
```

27975 f
20960 M

```
p = re.compile('\w+')
p.findall('string')
re.split('[\w]+',string)
re.sub('[\w]+','',string)
\w = (a-z, A-Z, 0-9)
\W = not ↑
\d = 0-9
\s = space, tab, newline
```

.findall(r' ')

.contains()

.startswith()

^ = start of a string

\d = one digit 0 to 9

\w = word character

\s = whitespace

\D = not a digit

\W = not a word

A-b_1

ABC

*t=)

+ = one or more

{3} = exactly three times

* = zero or more times

| = or

(...) = group

\t = tab

. = any except newline

[^abc] = not a b or c

[A-Z] = uppercase

[a-z] = lowercase

pd.crosstab(titanic.sex, titanic.class)

df.~~pivottab~~ = pivot_tabel()

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

pd.crosstab()

pd['sex'].str.contains

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='len')

pd.crosstab(titanic.sex, titanic.survived).sum(axis=1)

Pivot

titanic.pivot_table(index='sex', columns='survived', values='age', aggfunc=[np.mean, np.median])
↳ gem & mediaan leeftijd per geslacht van overlevers & niet overlevers

df.pivot_table(index='sex', values='survived', aggfunc=len)
↳ of [..., ...]

df.pivot_table(index=[col1, col2], values='col3', margins=True, margins_name='total')

Plot

df.plot(kind='bar', x='col1', y='col2', title="...")
↳ logy=True

Inlader

var = pd.read_csv(file, sep='\t', index_col=0, use_cols=[])

with gzip.open(file.gz, 'r') as fin: for line in fin: print(line)

---

df[df.col.str.startswith('G').values]

df[df.col.isin(['Eng', 'NL']).values]

Pandas

df.drop_duplicates([col1, col2 ...])

last element
df[-1:]

df.iloc → integer based zoeken, werkt op positie

df.loc → index based

df.query("....", inplace=True) → @var

df.loc[data.col < 2000], [col1, col2]]

nieuwe col maken
↳ df["new col"] = ...
↳ df.eval('D = (A+B)/C', inplace=True)

df.apply(func)
↳ def func(x):
return x

apply → row/col van df

applymap → element-wise df

map → element-wise Series

df.dropna() →how="all", axis="columns"

df.fillna(value) == df.replace(np.nan, value)

df.transform(lambda x: x+1)

change col order
df = df[[col1, col2 ...]]

find dtypes → df.dtypes

aantal kolommen selecteren

df.loc[:, 'col1':'col2'].apply(np.argmax, axis=1)

Groupby

df.groupby('col')[waarde].aggregate[min, len ...]

axis = 0 (rows) or 1 (cols)

Filter

df.groupby('col').filter(lambda x: len(x) >= 3)

transform

df.groupby('col').transform(lambda x: (x - x.mean()) / x.std()*10)

---

regex

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

Series.str.extract('([A-Za-z]+)') ← losse woorden

findall(r'^[^AEIOU].*[aeiou]$')

^ = start string
$ = end string
[] = set of chars
. = any char except \n
* = 0, meer
+ = 1, meer

{} = precies zoveel → a{2} = a, L, L

---

mean absolute deviation

absolute value

unbiased variance

standard error mean

Regex

.str.findal(r'...')

\d	-	digit
\w	-	word char
\s	-	whitespace
\b	-	word boundary
^	-	beginning of string
$	-	end of string
.	-	Anything
-	-	Symbol range

(?<! ignore me)

[ ]	-	Matches char in brackets
[^]	-	negates
|	-	either or
( )	-	group

quantifiers:

*	-	0 or more
+	-	1 or more
?	-	0 or 1
{3}	-	exact number
{3,5}	-	range

Pandas

df.Pivot_table(index='', values='', columns='', aggfunc=[len, 'mean', 'median'])

↑ np.mean

Pd.~~cross~~crosstab(df.x, df.y).sum(axis=1)

Pd.column.Value_counts()

Pd.groupby('column1')['column2'].count()

Extra

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

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

$$A = \frac{tp + tn}{tp + tn + fp + fn}$$