PD.TAB

```
df = df.set_index('A')
df = df.astype(dtype)
df = df.T
top_left_corner_df = df.iloc[:4, :4]
df.set_index

df_copy = pd.concat
df.nlargest

df = df[df['col2'] >= 0.0]
df = df[df['col'].isin([1,2,5,7,11])]
df = df[df['col'].str.contains('hello')]
value = df.loc['row', 'col']
df = df.loc['row1':'row3', 'col1':'col3']

df[x] = df[x].str.split(';').str[1]
for x in list(df.columns):
    df[x] = df[x].str.split(';').str[1].astype(int)
pwa.index = pd.to_datetime(pwa.index)

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

```
df = df['col3'] >= 1.0 | df['col1'] <
```

```
df.A.value_counts()
df.groupby('A').B.count()
df.pivot_table(index='A', values='B', aggfunc=len)
pd.crosstab(df.A, df.B).sum(axis=1)
```

```
df.pivot_table(index='A', columns='B', values='C')
pd.crosstab(df.A, df['B']) #relatie
```

[x for sublist in L for x in sublist]

```
'[AEIOUYaeiouy]'
datum
'\d{2}-\d{2}-\d{4}'
html
'<[^?]+?>'
'\b[Aa]\b'
woorden
'\w+'
```

```
df.pivot_table(index='A', columns='B', values='C', aggfunc=[np.mean, np.median])
df = pd.read_excel(file, index_col='A')
(df.A.str.findall(r'abc').str.len() / df.A.str.len()).describe()
df.A.str.contains(r'abc').sum()
df.join(df2, lsuffix='_suf')
```

```
sns.scatterplot(x=df.A, y=df.B)
corr = df.A.corr(df.B)
```

```
dict2 = {x: dict[x] for x in dict if dict[x] > 4}
df = pd.read_csv(file, sep='\t', encoding='ISO-8859-1').sort_values(by='A').drop(remove_cols, axis=1)
df = pd.read_csv(file, sep='\t', usecols=use_cols, index_col=0).sort_index()
np.mean, np.median, df.loc[df.A > mean]
df_series = pd.Series(df.A)
df_series[:20].plot(logy=True)
```

```
df.A.mean()
df.A.median()
df.A.std()
```

```
df2 = df.groupby('A').sum().sort_values(by='B', ascending=F)
```

```
intersection = set(a) & set(b)
union = set(a) | set(b)
jaccard = int / un
```

```
sns.distplot(df.A)
plt.subplots()
sns.distplot(df.A.apply(np.log))
```

- pivot_table (welke data, index = x, columns = y)
- sns. load dataset ('planets')
- agg recht = multiple functions
- filter groups with .filter(filterfunc)
  filterfunc = x[data].std() > y
- transform by group data (lambda x: x - x.mean())
- apply applies arbritary function
- list in gp to define which group
- {A: vowel, B: constant, C: constant} picks index
- str. lower picks index
- (10 // x[year]) * 10
- pivot table me gp = .gp([x1, x2])

make lt with groupby:

open bestand:
with open (bestand) as f
  for l in f:
    for l in list(l)

make date with datetime()
pd. datetimeindex() make a date index
subtract index from eachother to get [0days, 1days]
- make dti with pd.date_range

Regex =
- één van: [ABC]
- niet één van: [^ABC]
- any = .
- white sp = \s
- digit = \d
- één of meer: +
- 0 of 1 = ?
- ^ = start
- $ = end
- WB: \B
- 3 van A: A{3}
- letters: [a-z A-Z]

- normalize min max = (df - df.min()) / (df.max() - df.min())
- normalize z-score = (df - df.mean()) / df.std()
- sns. pairplot is used to visually find correlations
- zipfile - zipfile() use to read but not open zip

- errorbar(x, y; yerr = 0.8, fmt = '.k')
- fill_between(xfit, yfit - dyfit, yfit + dyfit, color)
- plt.histo for histogram
- distplot combines kde & histo
- sns. facetgrid (tips df, row = s, column = t, margin title)
- grid. map (plt. hist, tip_pct, bins = np. linspace(0, 40, 15))
- sns. factorplot ("day", "total", "sex", data = tips, kind = 'box')
  ↳ map total against day for random

- `df['status'].astype('cat')`
  `df['status'].cat.set_categories(['won','lose'], inplace=True)`

test acc = 95%	TP = .95 * 0.001
SCA: 1/1000	FP = (1-.95) * 0.999
TP/(TP+FP)	1-(TP/(TP+FP))
	of: FP/(TP+FP)

- `df.pivot_table(index=['M','S'], columns=['P'], values=['Q','P'], aggfunc={"Quan":len, "price":np.sum}, fill_value=0, margins=True`

- `table.query('Manager == ["Mik"]')`
- `" ('status == ["pending","won"]')`

`df.country.str.replace('.','')`
`"".".".".".(r'U.*','usa')`
- `dfnorm = (df - df.min()) / df.max() - df.min())`

Z-score: `(df - df.mean()/df.std()).describe()`

- `titanic.sex.value_counts()`
- `titanic.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, t.survived).sum(axis=1)`

—

- `tit.pivot_table(index='sex', columns='class', values='survived')`
- `"" "" ("" "", aggfunc=len) -> pd.crosstab(t.sex, tit['class'])`
- `pd.crosstab(titanic.pclass, titanic['class'])`

—

- `t = tit.dropna(subset=['age'])`
  `t.pivot_table(index='age', columns='sex', v='adult_male', aggfn=sum)['male']`
  `pd.testing.assert_frame_equal(t.query("sex == 'male' and age >= 16"), t[t.adult_male])`

—

- `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);`

—

- `prince.text.str.contains(r'\b[Ss]ex\b').sum()`
- `prince."".".".(r'^[Ss]ex\b').sum()`
- `"".".".".".startswith('Sex').sum()`
- `eg = "".".".str.findall(r'[Ss]ex\w+')`
  `eg[eg.str.len()>0].head()`
  `.count()`
- `(Counter([s for l in eg.values for s in l]))`

- `prince.text.str.findall(r'\b[A-Za-zÜğ]*[0-9]+[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')`

- `(100* prince.text.str.findall(r'[AEIOUYaeiouy]').str.len()`

- `planets.groupby('method')['distance'].max()`
  `planets."(".apply(sort_values('distance')`
  `df.index = df.index.droplevel(0)`
  `"."("")['dist'].idxmax().dropna.astype(int).values`

`Planets.loc[]`

```
for l in tqdm_notebook(f):
    rawdata = l.split('\t')
    data = []
    totaledits = 0
    for n in range(1, len(rawdata)):
        Currentedits =
            int(rawdata[n].split(',')
            [1].strip('\n'))
    if Currentedits > 1:
        totaledits += Currentedits
```

```
if totaledits > 4 and len(data) <= 5:
    for lang in data:
        edits[lang[0]] += lang[1]
        editors["[0]] += 1;
if len(data) > 1:
    Edits_by_mul["[0]] += lang[1]
    multilnr[lang[0]] += 1;
if len(data) >= 2:
    data = sorted(data)
    combi = list(combinations(np.arange(0, len(data)), 2))
    for i in combi:
        lang1 = data[i[0]][0]
        lang2 = "  [i[1]][0]
        paren[lang1, lang2] += 1
```

```
data.append
((rawdata[n]
.split(','),
Currentedits))]
```

```
with open('recipe.tens
'r') as f:
data=(line.strip()
    for line in f
```

```
dat_js
"
[x0]
.format
(',',join(
data
```

variance: `(x1 - x1.mean(` `** 2`

Std: `np.sqrt(((x1 - x1.`

a* , a+ , a? → 0≥ 1≥ 0/1

a{1,3}

a{5}, a{2,}

abicd

. any char exc newline

\w \d \s → word /digit /whitespace

\W\D\S → NOT

[abc] → any → [^abc] → not

^ABC$ → start / end

[a-g] → between

\b \B → word bound / not word bound

Hoeveel mensen zaten er per geslacht in elke klasse?

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

Hoeveel kans had de rampzalige daarvan overleefd?

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

Wat is de gemiddelde en median leeftijd per geslacht v.d mensen die het hebben overleefd en niet.

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

Beschrijf de relatie tussen de variabele class en pclass.

pd.crosstab(titanic.pclass, titanic['class'])

Wanneer is de variabele 'adult_male' waar?

t = titanic.dropna(subset=['age'])
t.pivot_table(index='age', columns='sex', values='adult_male', aggfunc=sum)['male']
pd.testing.assert_frame_equal(t.query(t.query("sex=='male' and age>=16"), t[t.adult_male])

1. Hoeveel namen bevatten het woord _sex_.
2. Hoeveel beginnen met dat woord
3. Hoeveel namen bevatten een woord sex, waar is niet gelijk aan _sex_
4. print al deze woorden uit deze _sex_ bevatten
5. vind alle woorden in zijn songs die bestaan uit gemeenschappelijke letters in andere teksten als (42, 42)

preprocess.

prince = pd.read(bestand)
prince.text = prince.text.astype(str)
prince.head()

#1&2. print(prince.text.str.contains(r'\b[Ss]ex\b').sum())
print(..... r'^[Ss]ex\b').sum),
prince.text.str.startswith('Sex').sum() + prince.... startswith('sex').sum())

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

#4. from collections import Counter
print(Counter([s for l in ef.values for s in l]))

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

Prob A given B
`df.groupby('A')['B'].value_counts() / df.groupby('A')['B'].count`

```
titanic.pivot_table(ind=sex, col=class, val=survived, aggfunc=sum) / ... agg=len of mean
Pd.crosstab(titanic.sex, titanic['class'])
titanic.pivot table(ser, survided, age, agafunc=[np.mean, np.median])
pd.crosstab(titanic.pclass, titanic[class])
t = titanic.dropna(subset=[age])
t.pivot_table(ind=age, col=sex, val=adult_male, aggfunc=sum)['male']
pd.testing.assert_frame_equal(t.query("sex=='male' and age>=16"), t[t.adult_male])
```

```
from collections import Counter
let_count = defaultdict(int)
with open(best, encoding='utf8') as f:
    for line in f:
        for letter in list(line):
            let_count += 1
    pd.Series(let_count).sort_values(ascending=False)
```

. = any char, + = a lot
\d = digit
\w = word char
\s = white space
\ = escape ~~sep~~ special char
^ = except
- = ... tot ...
{3} = 3 matches
[^abc] = ~~ab~~ char except abc
^ = start
$ = end
^...$ = exact match
roar = any string with roar in it

```
Prince.text.str.contains(r'\b[Ss]ex\b').sum()
Prince.text.str.startswith('Sex.').sum()
Prince.text.str.findall(r'[Ss]ex\w+') :ef
ef[ef.str.len()>0].count()
Counter([s for l in el.values for s in l])
100 * prince.text.str.findall(r'[AEIOUYaeiouy]').str.len()/prince.text.str.len().
      count.
```

1 op 1000 soa
Tp = 0,95 × 0.001
FP = 1-0.95 × 0.999
TP/TP+FP = ~~Recall~~ precisio
FN = recall

```
carsdf.hp.fillna(carsdf.hp.mean())
carsdf.hp.replace(0, carsdf.hp.mode(), inplace=True)
... = carsdf.hp.apply(lambda x: np.random.choice(carsdf.hp.dropna()))
2 score = (df.col - df.col.mean())/df.col.std()
min-max normal = df - df.min()/df.max - df.min()
normalize = p/(p.sum(axis=0))
```

pd.cut = data in bins

```
filter = titan[titan['class']=='first'].groupby('sex')['survived'].sum()
pd.items(), set(), df.loc[df['n.soc'].idxmax()].title, df.at = value
```

re index

```
voor waardelijke kans = A&B / B
df.apply(pd.value_counts, axis=1).fillna(0).astype(int) / apply(pd.Series).stack().val_counts
```

```
pd.Series([0.25, 0.5, 0.75, 1], index=[2, 5, 3, 7])
pd.Series(dict)
A.add(B, fill_value=0)
data.dropna() axis=0, how='all', thresh=3
data.fillna(0)
index = pd.MultiIndex.from_tuples(index)
pop.reindex(index)
pop[:, 2010]
pop.unstack() level=0, 1
pop.index.names = ['state', 'year']
pd.MultiIndex.from_product([[2013, 2014], row[:2]])
np.concatenate([x, y, z])
pd.concat([ser1, ser2]) ignore_index=True, join='inner'
df1.append(df2)
pd.merge(df1, df2) on='employee'
df1.join(df2)
merged[merged['population'].isnull()]
df.query("year == 2010 & age == 'total'")
df.groupby('key').sum() .aggregate(['min', max])
pd.cut(titanic['age'], [0, 18, 80])
pd.qcut(titanic['Fare'], 2)
margins = True → all
.astype(int)
ser.str.capitalize() .extract('([A-Za-z]+)', expand=False)
.findall(r'^[^AEIOU].*[^aeiou]$')
.split().str.get(-1)
full_monte['info'].str.get_dummies('|')
with open(file, 'r') as f:
    data = (line.strip() for line in f)
recipes.name[np.argmax(recipes.ingred.str.len())]
↳ longest list
recipes.desc.str.contains('[Bb]reakfast').sum()
Recipes.name[selection.index]
```

Oefen

```
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='survival', aggfunc=len)
pd.crosstab(titanic.sex, titanic.survived).sum(axis=1)
titanic.pivot_table(index='sex', columns='class', values='survived')
aggfunc = sum / aggfunc = len
titanic.pivot_table(index='sex', columns='survived', values='age', aggfunc=np.mean)
pd.crosstab(titanic.pclass, titanic['class'])
t = titanic.dropna(subset=['age'])
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])
```

```
def letterletters(bestand):
    from collections import defaultdict   - counter
    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(asc=False)
    return tellingseries
zinnen = pd.read_csv(bestand)  - names=[ ] skiprows=1
zinnen_text = zinnen.text.astype(str)
zinnen_text.str.contains(r'\b[Ss]ex\b').sum() - mid
zinnen_text.str.contains(r'^[Ss]ex\b').sum() - begin
zinnen_text.str.startswith('Sex ').sum() + - (1 - ('Sex ').sum())
? = zinnen_text.str.findall(r'[Ss]ex\w+')
[f.str.len()>0].count()
counter([s for l in f.values for s in l])
100 * zinnen_text.str.findall(r'[AEOUaeiou]').str.len() /
    zinnen_text.str.len()).describe()
```

```
with open(file, encoding='utf-8') as f:
    editors = defaultdict(int)
    edits = {}
    combilst = []
    D = {}
    for l in tqdm_notebook(f):
        ll = l.split("\t")
        for k,v in [(i.split(":")[0], int(i.split(":")[1].rstrip("\n"))) for i in ll[1:]]:
            editors[k] += 1
            edits[k] += v
        D[ll[0]] = {i.split(":")[0]: int(i.split(":")[1].rstrip("\n")) for i in ll[1:]}
    for k,v in D.items():
        if len(v) > 1:
            combilst.append(list(combinations(sorted(v.keys()), 2)))
            for i,j in v.items():
                multilingual[i] += 1
                Edits_by_multi[i] += j
    for i in combilst:
        for j in i:
            pairs[j] += 1
pd.DataFrame.from_dict(editors, orient='index', columns=['Editors'])

dataset_stats_wikipedia = pd.read_csv(local_stats(), sep="\t", index_col='Code', usecols=['Code','Editors',
.sort_index()
    coursdf.country.str.replace(r'Us*','USA')
    coursdf.plot.scatter()
    sns.lmplot()
```

min-max

```
(df-df.min())/(df.max()-df.min())
```

z-score

```
t602 = (t60 - t60.mean())/t60.std()
sns.distplot(t602)
sns.pairplot(df[[C]])
sns.heatmap(churndf.corr())
churndf[20] = (churndf[20].str.strip() == 'True.')
pd.crosstab(churndf[19], churndf[20]).plot.bar(stacked=True)
sns.load_dataset(':s')
df.sort_values(["  "], ascending=[False, True])
df[~df.str.contains("_")]
```

TP/(TP+FP) — Pos and test → besmet

1-(TP/(TP+FP)) == FP/(TP+FP) — pos and test → niet besmet

TP = 0.95 * 0.001 ≈ 0.00095

accuracy · 1 op 1000

FP = (1-0.95) * 0.9999

kans op geen soa

```
df[" "].corr(df[" "])
```

```
jaccard =
    A = set(isec)
    B = set(jsec)
    return len(A & B)/len(A/B)
        intersection  union
df[sections].apply(pd.Series).stack().value_counts()
plt.xticks([])

chance_dict = dict()
for b in frequent_sections:
    chance_dict[b] = dict
    for a in frequent_sections:
        if dummy_df[b].sum() > 0:
            chance_dict[b][a] = dummy_df[dummy_df[b] == True][a].sum()/dummy_df[b].sum
        else:
            chance_dict[b][a] = 0
```

```
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',aggfun=len)
pd.crosstab(titanic.sex,titanic.survived).sum(axis=1)

titanic.pivot_table(index='sex',columns='class',values='survived')
titanic.groupby(['sex','class'])['survived'].mean().unstack()

t=titanic.dropna(subset=['age']) 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])
```

```
bestand='prince_lyrics.csv'
from collections import defaultdict
def teldeletters(bestand):
    aantal=defaultdict(int)
    with open(bestand) as f:
        for L in f:
            for c in list(L):
                aantal[c]+=1
    aantal_series=pd.Series(aantal).sort_values()
    return aantal_series
teldeletters(bestand).plot(kind='bar',figsize=(15,7),lagy=True
```

```
prince=pd.read_csv('prince_lyrics.csv')
prince.text=prince.text.astype(str)
prince.str.contains(r'\b[Ss]ex\b').sum())
prince.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.count())
print(Counter([s for L in ef.values for s in L]))
# Alleen woorden die zowel een letter als een cijfer bevatten en of alleen daaruit bestaa
ef=prince.text.str.findall(r'\b[A-Za-z0-9]*[0-9]+[A-Za-z0-9]*[A-Za-z][A-Z
a-z0-9]*\b|\b[A-Za-z0-9]*[A-Za-z][A-Za-z0-9]*[0-9][A-Za-z0-9]*\b')
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()
```

```
with open(load_userlang(),encoding='utf-8') as f:
    c=0
    from collections import defaultdict
    editors=defaultdict(int)
    edits=defaultdict(int)
    multilingual=defaultdict(int)
    Edits_by_multilinguals=defaultdict(int)
    pairs=defaultdict(int)
    for f in tqdm_notebook(f):
        Line=L.strip()
        list_1=L.split('\t')[1:]
        list_2=[word.split(',') for word in list_1]
        dict_1={x[0]:int(x[1]) for x in list_2}
        data=dict_1.keys()
        data=sorted(data)
        if len(data)>1:
            pair_list=[]
            for p in data:
                pair_list.append(p)
            combi=combinations(pair_list,2)
            for sp in comb:
                if sp not in pairs:
                    pairs[sp]=1
                else:
                    pairs[sp]+=1
        if sum(dict_1.values())>=5 and len(dict_1)<6:
            for x in dict_1.keys():
                if dict_1[x]>1:
                    editors[x]+=1
                    edits[x]+=int(dict_1[x])
                    if len(dict_1)>1:
                        multilingual[x]+=1
                        Edits_by_multilinguals[x]+=int(dict_1[x])
        c+=1
```

```
*Hierarchical indexing
index=[('New York',2000),('New York',2010),
       ('Texas',2000),('Texas',2010)]
population=[1,2,3,4]
index=pd.MultiIndex.from_tuples(index)
pop=pd.Series(population,index=index)
pop⇒New York 2000 1
          2010 2
     Texas 2000 3
          2010 4
pop.unstack()⇒      2000 2010
New York          1    2
Texas             3    4
*un/stack(level=0/1)

df.index.droplevel(0
```

```
*pd.concat([df5,df6],join_axes=[df5.columns])
*pd.concat([df5,df6])==df5.append(df6)

df3=pd.merge(df1,df2)
df1.join(df2.set_index('employee'),on='employee')
```

```
dfnom=(df-df.min())/df.max()-df.min())
z-score:(df-df.mean())/df.std()).describe()
```

```
with open('....json','r') as f:
    data=(line.strip() for line in f)
    data_json="[{0}]".format(','.join(data))
```

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

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

actual \ predicted	−	+
−	TN	FP
+	FN	TP

Stel 95% accuracy & 1/1000 mensen heeft SOA
TP = (test ja, SOA) = 0,95 * 0,001
FP = (test ja, geen SOA = (1-0,95)*0,999
Wat is de kans geen SOA maar test positief
1-(TP/(TP+FP)) of FP/(TP+FP)

```
*standaarddeviatie
• bereken het gemiddelde
• neem van elk getal de afstand tot gemiddelde
• neem het kwadraat van die afstanden
• bereken het gemiddelde van die kwadraten
• neem de wortel van de uitkomst

*variance:
(x1-x1.mean())**2
*standaard deviatie:
np.sqrt(((x1-x1.mean())**2).mean())==x1.std(
```

Series

pd.Series([...], index=[...]) or dict
series.values / index / keys() / items()
↓ tuples

DataFrame

df.index / columns / items() / iterrows() / itertuples() rows
pd.Index → &, |, ^
pd.DataFrame(data, index=, columns=)

Slicing, masking & fancy

series[0:2] / ['a':'c'] excl/incl
series[(series > 0.3) & (series < 0.8)]
series[['a', 'e']]
series.loc[1] explicit index
series.iloc[1] python-style
df.T swap rows & cols
df.values[0] row df[0] col
df.iloc[:3, :2] python
df.loc[:'Illinois', :'pop'] incl
df.ix[:3, :'pop'] hybrid (excl & incl)
df.loc[df.density > 100, ['pop', 'density']]
df.iloc[3,2] = 80 assign
df[1:3] → rows
df[df.density > 100]

Ufuncs

np.exp(series) / np.sin(df * np.pi/4)
A + B or A.add(B, fill_value = num)
7.stack.mean() all values
df.substract(df['R'], axis=0) column-wise

Missing values

np.nansum(vals2)
.isnull() boolean
.notnull() opposite
dropna(axis=1) for cols
how='any/all'
thresh = 3 inplace=True
fillna(method='ffill')
0
axis=1

time

pd.to_datetime()
df.resample('BA').mean()

Multi index

pd.MultiIndex.from_tuples() / from_arrays() / from_product()
(levels=[['a','b'],[1,2]], labels=[[0,1,1,0],[1,1,0,0]])
df.reindex(), level=0/1/...
df.unstack() series to df
series
df.stack() df to series
df.index.names = ['ind1', 'ind2']
df.sort_index()
pd.cut(df['age'], [0,18,80])
df.reset_index(name=) cut

Concat, merge

pd.concat([obj1,obj2], join='outer', ...)
df1.append(df2)
pd.merge(df1, df2)
df1.join(df15)
df.col.value_counts()

Query

df.query("col==2010 & other-col=='total'")
@var pd.eval('df2.T[0] + df3.iloc[3]')

Aggregation

df.mean(axis='columns')
df.describe(), count(), first/last(), mad(), prod()

Groupby

[...,...] .aggregate(['min', np.median])
df.groupby('col')['col2'].sum()
.agg(['min','max'])
.describe()
def filter_func(x):
return x['data'].std() > 4
df.groupby('col').filter(filter_func)
.transform(lambda x: x - x.mean())
.apply(norm_by_data)

Pivot tables

1 or list ↓ 1 or list ↓
df.pivot_table('value_col', index=, columns=)
df.pivot_table(data, values=, index=, columns=,
aggfunc= , fill_value= , ...)
↑ or dict

Strings

col.str.match() boolean
.extract() matched groups as str
.findall each elem
.replace()
.contains() boolean
.count() occ. of pattern
.(r)split() excepts regex
.get_dummies()
.get

Regex

^ start, $ end, \b word boundary, \B non-word bound
\d one digit, \D non-digit, \w, \W, \s whitespace, \S
[abc] one char [^abc] not
* 0 or more + 1 or more ? 0 or 1 {2}
{2,5} {2,} {,5} up to 5
\t \n
\A start of string
re.compile match, search, split, sub
findall

String operaties

```
str.split().str[0]
str.contains(r'\b[Ss]ex\b').sum()
str.contains(r'^[Ss]ex\b').sum()
R = Str.findall(r'[Ss]ex\w+')
R[R.str.len()>0]
from collections import Counter
Counter([s for L in eR.values for s in L])
(100 * df.kolom.str.findall(r'[AEIOUYaei...]').str.len() / df.kolom.str.len())
```

decade = 10 * (year // 10)
decade = decade.astype(str) + 's'
decade.name = 'decade'
groupby decade

Capitalize a list with strings and a None → make a series of the list and do Series.str.capitalize()

str.replace('.', '').str.lower().str.strip.str.replace(" ", "").str.replace("\n", "")

Series

```
s.unstack() → multiindex → kolommen → index
s.unstack(0) →           → kolomm 0 index
pd.Series(np.diag(df))
df[df != np.diag(df)]
s.to_frame().reset_index() → df met index series als kolom
ser1[~ser1.isin(ser2)]
```

Dataframe operaties

```
df.mean(), mean(1)
df.count() → aantal regels per kolom
df.dropna()
df.loc[index].kolom → waarde uit kolom bij bepaalde index
df.loc[df.kolom.isin([lijst])] → rijen waar kolom waarde in lijst
(df[df.mean().index] - df)
len(set(df.index)) == len(df.index) → False → duplicate indexes
dubbel_index = df.index.value_counts()
dubbel_index[dubbel.index > 1]
pd.cut(df.kolom, [0, 10, 30, 60, 80])
skiprows = 1 → 1e rij / skiprows = [1] → index 1 → 2e rij
[1,2] → index 1 & 2 → 2e en 3e
df.sort_index(axis=1) → kolommen sorteren
df.pivot(df.mean())
```

spaties verwijderen

```
from collections import defaultdict
telling = defaultdict(int)
with open(bestand, encoding="utf8") as F:
    for L in F
        for c in L:
            telling[c] += 1
```

- alles maar in & \R
- [^ABC] alles Behalve A, B of C

^ = begin string
$ = einde string
[] → of → [Aa]
? = 0 of 1 x / * 0,1 meer
+ 1 of meer

Groupby & pivot etc

- * aggregate([min, max, count]) verschillende berekeningen op groupby
- * aggregate({kolom: funct3}) funct alleen op bepaalde kolom
- * df.Groupby(kolom).transform(lambda x: x - x.mean()) Haal van elke cell de mean van de groep waarbij de cell hoort af
- * df.Groupby(lijst) of [dict, nieuwe namen aan indexes geven en daar op groepen
- * df.Groupby(str.lower).mean() lowercased indexnaam en mean groep
- * df.groupby([str.lower, lijst]) nieuwe namen indexes en ook nam lower multi index
- * pivot_table(df, values = waardes die je wilt tellen, index = index die je wilt, columns = aggfunc = funct over values) OF aggfunct = {kolom: funct} values hoef je dan niet aan te geven
- * df.Groupby(kolom)[[kolom]] → dataframe
  [kolom] → serie
  .idxmax() → vindt alle → max. indexes
- * df.iloc[[df.Groupby...].idxmax()] vindt de rijen
- eut. dropna().astype(int)
- .apply(norm_by_data2)

```
def norm_by_data2(x)
    x['data1'] /= x['data2'].sum()
```

Theorie & formules

acht

	predicted neg	predicted pos
neg	TN	FP
pos	FN	TP

Conditionele Kans = Kans (A als A ook B Bevat) (A ∩ B) / B

intersection (∩) = die beide lijsten hebben
union (|) = alle unieken v/d 2 lijsten
sym. dif. (^) = in de ene maar niet in de ander

$$z\text{score} = (df - df.mean()) / df.std()$$

$$\text{precision} = \frac{TP}{TP+FP} \qquad \text{Accuracy} = \frac{TP+TN}{\text{totaal}}$$

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

TN = true neg → is neg, predic neg
TP = true pos → is pos, predic pos
FN = false neg → is pos, predic neg
FP = false pos → is neg, predic pos

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

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

.aggregate

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

df.value_counts()
Counter(df.sex)
pd.crosstab(df.sex, df.gender, margins=True)
df.pivot_table(c='class', i='sex', v='survi', aggfuc='count')

pd.cut(df['age'], [0, 10, 30])
of qcut

aggfunc={'survived': sum, 'fare': 'mean'}