DATA SCIENCE CHEATSHEET 2019

ASHWIN

ROUND (2.04, 2) Round of op 2 decimale

FLOAT (2) maak comma getal

List.APPEND('item4') add item to list

List[:2] selection of list

PRINT 'HENK is %S JAAR OUD' %STR(LEEFTIJD)

IF 'HENK' IN MEMBER: check of Henk in mensen voorkomt

CREATE DATAFRAME

DF = PD.DATAFRAME (DATA, INDEX = [1,2,3,4],
COLUMNS = ['DATE', 'B', 'C', 'D']) maak dataframe van strings

DF = PD.READ_EXCEL ('HTTP://URL.NL.XLS') read file

DF = PD.READ_CSV('./output.csv', SEP=',',
SKIPROW = 1, HEADER=NONE, NAMES = ['NAME', 'F', 'M']

Summerise DATAFRAME

DF.HEAD(10) idem

LEN(DF) # of Rows in DF

LEN(DF.CLMN_NAME.UNIQUE()) no distinct values

DF.CLMN_NAME.VALUE_COUNTS() Count # of Rows for each unique value

DF.APPLY(PD.VALUE_COUNTS) voer functie uit op alle colums van df.

Doing stuff with DATAFRAME

DF.SORT_VALUES ('CLMN_NAME', ASCENDING = FALSE) sorteer rijen

DF.SORT_INDEX() sorteer index van df

DF.RESET_INDEX() reset index, maakt # van rijen van, move old index to clumns

NEW_DF = DF.SET_INDEX ('DATE') set colum as index

DF.COLUMNS = ['NAME', 'F', 'M'] set colum names

DF.GROUPBY (BY = 'CLMN_NAME') groep by values of in column

DF.CLMN_NAME = DF.CLMN_NAME.ASTYPE (INT) change datatype of values in column

DF.TRANSPOSE() draait df kwalitatief. maak van columns rijen

DF.CLMN_NAME.MIN() MAX() min max van column

DF.SUM (AXIS = 1) som van waarde

DF.COUNT()

DELETE DATA

DF = DF.DROP (['LENGTH', 'HEIGHT'], AXIS = 1) delete column

DF.DROP_DUPLICATES() Remove duplicates in rows

missing DATA

DF = DF.DROPNA() drop Rows with any column with NAN

DF = DF.DROPNA (AXIS = 1) drop rows with NAN in specific colum

DF.DROPNA (SUBSET = ['CLMN_NAME'])

DF = DF.FILLNA (VALUE) replace NAN with value

Data Science cheatsheet

df.loc [voorwaarde] , df.column, iloc[1

df."column".value_counts() Fancy indexing: data [[1, 2]]

df."column".sort_values()

df.index

df.columns

df= pd.read_csv(data, index_col=)

pd.Series(dict)

regex for removing tags: <[^>]*>

pd.DataFrame.from_dict(data, orient, columns=)

.[illegible]

.dict.get( )

df["Column name"] = list

df[[Column1, Column2]].plot.bar()

.sort_index()

.set_index()

.groupby()

.idxmax(), .dropna(), .astype()

.stack() creates multiindex

.aggregate() can take list of multiple aggregates

.pivot_table(data, values=, index=, columns=, aggfunc=)

pd.crosstab(titanic.sex, titanic['class']) hoeveel per geslacht per klasse

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

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

df nummers met het woord sex

prince.text.str.startswith('sex').sum() + "('sex')

nummers beginnend met sex

axis=0 → rows
axis=1 = columns

df.groupby(['Animal']).mean()
([Animals', *'year', ...]

count woman & man.
1 df.sex.value_counts()
2 df.groupby('sex')['survived'].count()
3 {s: df[df.sex==s].sex.count() for s in df.sex.unique()}
4 df.pivot_table(index='sex', values='survived', aggfunc=len)
6 pd.crosstab(df.sex, df.survived).sum(axis=1)

gemiddelde & mediaan van overlevende:
df.pivot_table(index='sex', columns='survived',
values='age', aggfunc=[np.mean, np.median])

def teldeletters:
from collections import defaultdict
telling = defaultdict(int)
with open(bestand) as f:
for l in f:
for c in list(l):
telling[c] += 1
tseries = pd.Series(telling).sort_values(asc=False)
return tseries

#dict comprehension:
{k:v[:30] for k,v in genlist.items()}
pd.DataFrame.from_dict(data, orient='index')

# selectie = df[df['year']=='2018']
df1.join(df2, rsuffix="_my")
pd.Series(np.diag(df))
prince.text = prince.text.astype(str)
prince.text.contains(r'\b[ss]ex\b').sum()
prince.text.startswith('sex ').sum()
# percentage winners
(100* prince.text.findall(r"[0-9]+%").str.len()/
prince.text.len()).describe()

freq_sect = sectietitels(df).most_common(8)
index
dummydf = df.sections.apply(pd.Series).
stack
dikt = dict()
for b in freq_sect:
dikt[b] = dict()
for a in freq_sect:
if dummydf[b].sum() > 0:
dikt[b][a] = dummydf[dummydf
[b] == True][a].sum() /
dummydf[b].sum()
else:
dikt[b][a] = 0

def sectietitels(df):
df.dropna(inplace=True)
x = pd.DF.from_dict(Counter
(list(it.chain.from_iterable
(df.secties))), orient='index')
y = pd.Series(x[0]).sort_values()
return y[y>5]

%ls -th row column
%lsmagic x[1,2]

x[:2,:3] two rows three columns
x[:3,::2] 3 rows, every other col
x[::1,::-1] reverse
x[:,0] first column
x[0] first row
x[x%3==0] deel baar door 3

df.sort_values(['height','width',ascendin
=[False,True])
df.groupby('year')['length'].max(
df.index.str.lower().str.strip().str
.replace("x","y").sort_values()
df.apply(np.mean) # mean of each col.
df.woonpl = df['woonplaats'].loc[df['woonplaats
'].isin(['Breda','Tilburg'])]
df['ratio man'] = df['m']/(df['m']+df['v'])

counting:

find unique:

```
df.column.value_counts()
df.groupby(column)[column2].count()
{s: df[df.column==s].column.count() for s in df.column.unique()}
df.pivot_table(index='column', values='column', aggfunc=len)
pd.crosstab(titanic.sex, titanic.survived).sum(axis=1)

pd.crosstab()
pd.pivot_table(index, column, values, aggfunc='')

with open(bertand) as f:
    for l in f:
        code

.plot(kind='bar', 'hbar' etc.)
```

Pandas

df.dropna() df.replace("a","t")
df.fillna(0) (inplace = true for column)

selecting:
df.loc[:, df>1].any()
df.loc[:, df>1].all()]
df.loc[:, df.isnull().any()]

indexing
df[df.country.isin(df2.type)]
df.filter(items='a','b'])
df.select(lambda x: x*2)
s.where(s>0)
df.query('second > first')

df.ix[:,'capita']
df.ix[1,'country']

[~(s>1)]
[(s<-1) | (s>2)]
[df['popul']>12000]
['logic'] = np.where(df['AAA']>5, 'high', 'low')

multi index
p = pd.series(populations, index=index)
dex = pd.MultiIndex.from_tuples(index)
p = pop.reindex(index)
p.df = pop.unstack()

merge

pd.merge(
data1, data2, how='left',
on='x1')
- left, right, inner, outer

data1.join(data2, how='right')
data1.append(data2)
pd.concat([s1,s2], axes=1, keys=['1','2'])
pd.concat(data1, data2], axis=1, join='inner')

df.drop(['a','c'])= from rows
df.drop('country', axis=1) = from columns

f = lambda x: x*2
df.apply(f)
df.applymap(f)

```
def jaccard(i, j, df):
    i = list(df.loc[df['title']==i), 'sections'])[0]
    j = list(df.loc[      ==j)]      "
    s1 = set(i)
    s2 = set(j)
    return len(s1.intersection(s2)) / len(s1.union(s2))
```

pd.read_csv(bestand, etc)
pd.read_excel( )
sep = \t, usecols =
test = pd.DataFrame(random.rand(20,5))
pd.DataFrame.from_dict(dict, orient='index

help(pd.series.lo

tito pandas

- tito.index.str.lower().str.strip().str.replace(" ","").str.replace(',',"").sort_values()
- tito['rsme'] = np.sqrt((tito.quasitito - tito.verwachte)**2)
- dubbel = tito.index.value_counts()
  dubbel[dubbel>2]
  len(set(tito.index)) == len(tito.index)

TITANIC 5 different ways

titanic.sex.value_counts()
titanic.groupby('sex')['survived'].count()
titanic[titanic.sex==s].sex.count() for s in titanic.sex.unique()}
titanic.pivot_table(index='sex', values='survived', aggfunc=len)
titanic.crosstab(tit.sex, tit.survived).sum(axis=1)
wat is gem. en median leeftijd per geslacht van mensen welk niet overleefd?
titanic.pivot_table(index='sex', columns='survived', values='age', aggfunc=[np.mean, np.median])
verschil relatie class en pclass
pd.crosstab(titanic.pclass, titanic['class'])
wanneer is adult male waar?

```
t = titanic.dropna(subset='age')
t.pivot_table(index='age', columns='sex', values='adult_male', aggfunc=sum)['male']
```

churn

%ls /// churn*
!head /// churn.all
!cat /churn.names

prince

```
def teldeletters(bestand):
    telling = defaultdict(int)
    with open(bestand) as f:
        for l in f:
            for c in list(l):
                tellingtest = 1
    tseries = pd.series(telling).sort_values(asc=False)
    return tseries
```

meeste vertalingen vanuit NL

Df met totaalcode als index en aantal vert. als waarde

lezen:

```
prince = pd.read_csv(bestand)
prince.text = prince.text.astype(str)
prince.text.str.contains(r'\b[sS]ex\b').sum()
prince.text.str.contains(r'[sS]ex\b').sum(),
    prince.text.startswith('Sex').sum(), "".join('sex').sum(), df2 = df[n_words < df.n_words.quantile(.gg)]
ef = p.t.str.findall(r'[sS]ex\w+')
```

corr2 = df2.n_sections.corr(df2.n_words)

```
means = (df.n_words/df.n_sections)
df['mean_words'] = means
df = df.dropna(subset=['mean_words'])
mean_words.describe()
```

logs = np.log(df.mean_words)
normal = df.mean_words
sns.distplot(logs)
sns.distplot(normal)

other examples:

```
nl_trans = raw.loc[raw['orig_lang'] == 'nld']
most_nl = nl_trans.sort_values(by=['num_trans'], asc=False).head()
most_trans = set(most_nl[['targ_lang', 'num_trans']].apply(tuple, ax
total_trans = pd.pivot_table(raw, idx='org', col='target', val='num_trans',
    aggfunc=sum, marg=true, fill_value=0).reset_ind().rename_axis(none,
    axis=1).sort_values(by=['all'], asc=False).iloc[1:]
total_trans = totaltrans.filter(['org', all'], axis=1)
total_trans = total_trans.set_index('org')
total_trans.head(20).plot(kind=bar, logy=true)
```

- to series: total_trans.squeeze()
- corr = df.n_sections.corr(df.n_words)

```
col.value_counts()
groupby(col)[col].count()
pivot_table(index=col, value=col, aggfunc=)
crosstab(df.col, df.col).sum(axis=1)
df[df.col == x].col.count() for x in df.col.unique()]
```

pandas ob vragen

① 5 manieren hoeveel m/v t.sex = male/female
1) titanic.sex.value_counts()
2) titanic.groupby('sex')['survived'].count()
3) ES: titanic['titanic.sex == s].sex.count()
for s in titanic.sex.unique()}
4) titanic.pivot_table(index='sex', values='survived', aggfunc=len)
5) pd.crosstab(titanic.sex, titanic.survived).sum(axis=1)

② Hoeveel per geslacht in elke klasse[a], welk deel survived[b]?
b) titanic.pivot_table(index='sex', columns='class', values='survived') .(aggfunc=sum)/.aggfunc=len)
a) pd.crosstab(titanic.sex, titanic['class'])

③ gem & med. leeftijd per geslacht per overleefd/niet overl.
titanic.pivot_table(index='sex', columns='survived' values='age', aggfunc=[np.mean, np.median])

④ leeftijd relatie 2 var.
pd.crosstab(t.pclass, t['class']) + met elkaar te vertalen

⑤ wanneer adult_male waar?
t = titanic.dropna(subset=['age'])
t.pivot_table(index='age', columns='sex', values='adult_male' aggfunc=sum)['male']

⑥ pd series hoe vaak elke letter voorkomt
```
def teldeletters(bestand):
    from c import defaultdict
    count = defaultdict(int)
    with open(bestand) as f:        encoding='utf-8'
        for L in f:
            for c in list(L):
                count[c] += 1
    countseries = pd.Series(count).sort_values(asc=False)
    return countseries
```

⑦ pmnce.text = pmnce.text.astype(str) p.t.s. = pmnce.text.str
a) bevat 'sex' of 'Sex'
pts.contains(r'\b[Ss]ex\b').sum()
b) begint met 'sex of 'Sex'
pts.contains(r'^[Ss]ex\b').sum()
of pts.startswith('sex').sum + pts.startswith('Sex').sum()
c) hoeveel/bevat sex maar != 'sex', welke woorden
ef = pts.findall(r'[Ss]ex\w+')
ef[ef.str.len()>0].count()
.head(10)
d) alle woorden printen
Counter([s for L in ef.values for s in L]) [a-zA-Z]
f) % klinkers
&pts.findall(r'[AEIOUaeiou]').str.len()/pts.str.len().sum() x100

- index_col='naam' of .set_index('naam')
- dubbele namen index + hoevaak:
cito.index.value_counts()[cito.i.v.c() values != 1]
- verwijder spaties begin/eind, quotes
cito.index = cito.index.str.lower().str.replace(r'\"|\'', '').str.strip('')
- geen "school" in naam
cito[~cito.index.str.contains('school')]
- cito['RMSE'] = np.sqrt((cito.quasi - cito.verw)**2)
- normaliseren door v. elke waarde gem v. col af te halen
mean = cito.mean()
cito[mean.index] - mean

, skiprows=1)
- sns.distplot(df)
- df.isnotnull().mean()

max sep_length for each species
ins.groupby('species')['sepal_length'].max()
- CS = pd series of word : occurrence no
len word x occurrence == 24
CS[CS * CS.index.str.len() == 24].index
- % of unique words in txt occurs 1x
CS[CS == 1].sum() / CS.count()
- % of all words in txt occurs 1x
CS[CS == 1].sum() / CS.sum()

Theorie

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

accuracy 95% zieke 1/1000
TP = 0,95 x 0,001 TN = 0,95 x 0,999
FP = 0,05 x 0,999 FN = 0,05 x 0,001
precision – de test zegt het, heb ik het ook
recall/pakkans – ik heb het, zegt de test het ook
P(test positief, heb het niet) = 1 - precisie

voorwaardelijke kans

$P(A|B) = \frac{P(A \cap B)}{P(B)}$ aantal x dat A&B voorkomt / aantal x dat B voorkom

Min max norm

$\frac{X - min(X)}{max(X) - min(X)}$ 0-1, 0.5 ≤ gem

Z-score

$\frac{X - mean(X)}{SD(X)}$ -4 t/m +4, 0 = gem → .std() = 1

IQR

50% tussen 70 & 80 IQR = 80 - 70 = 10
outlier als 80 + 1,5 x 10 = 95 of 70 - 1,5 x 10 = 55

stromend algoritme / top-k

bgv. 5 langste lln:
heap = {3} #lln toev. +/m len 5
tot_nu_toe_langste # langste in heap
nieuw daarmee vgl.

with gzip.open(...)

* meeste bondsdag leden v. welk land
w = bd.groupby(0)[1].count()
w.idxmax(), w.max()
* welke party grootste aantal leden uit 1 deelst
bd.groupby(0)[2].count().argmax()