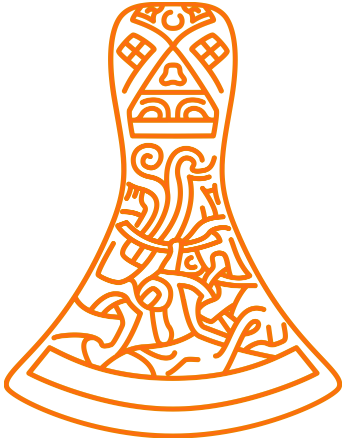
DYALOG

Glasgow 2024

Dyalog for data science

Jesús Galán López

GHENT UNIVERSITY



Dyalog for data science?

What is data science?

What is data science?

Data science is an interdisciplinary academic field that uses statistics, scientific computing, scientific methods, processes, scientific visualization, algorithms and systems to extract or extrapolate knowledge and insights from potentially noisy, structured, or unstructured data.

https://en.wikipedia.org/wiki/Data_science

What is data science?

Data science is an interdisciplinary academic field that uses statistics, scientific computing, scientific methods, processes, scientific visualization, algorithms and systems to extract or extrapolate knowledge and insights from potentially noisy, structured, or unstructured data.

https://en.wikipedia.org/wiki/Data_science

What is data science?

Data science is an interdisciplinary academic field that uses statistics, scientific computing, scientific methods, processes, scientific visualization, algorithms and systems to extract or extrapolate knowledge and insights from potentially noisy, structured, or unstructured data.

Large
data tables
(CSV files)

SCIENCE

Smaller
data tables
and charts

Examples

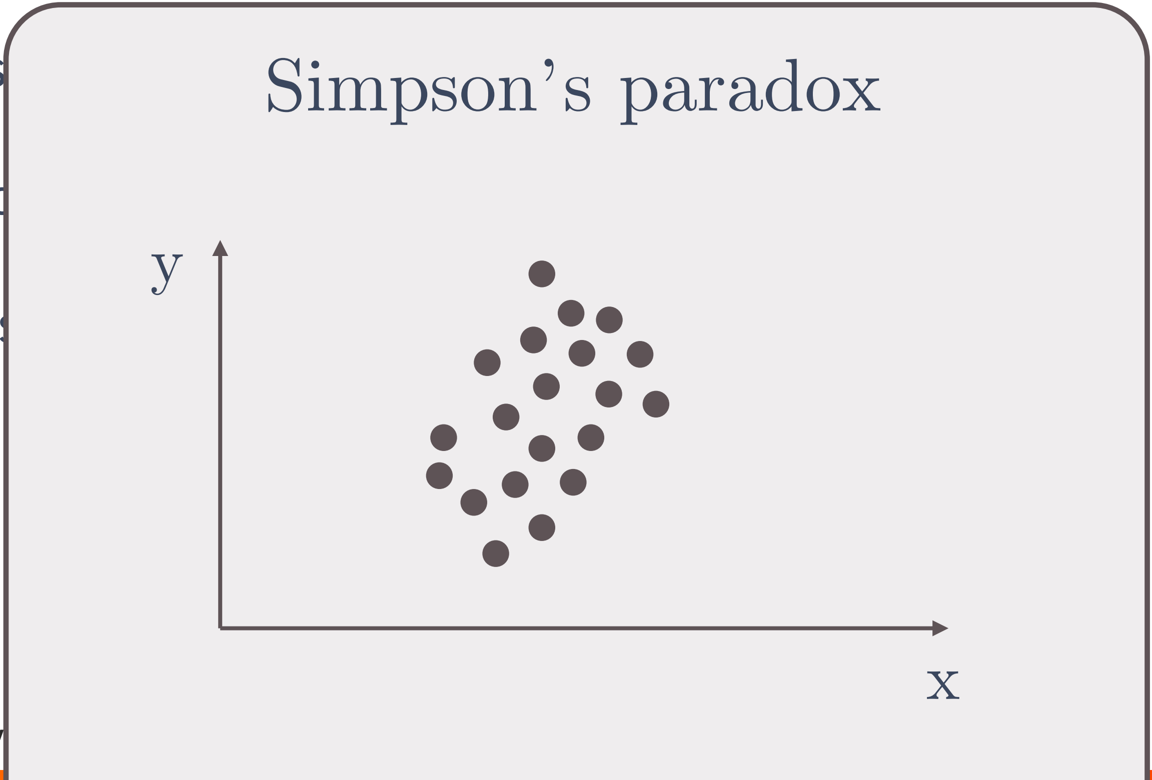
Example 1

Berkeley admissions (1973)

- Admissions at UC Berkeley graduate schools in 1973
- Larger percentage of male applicants admitted
- Gender bias or Simpson's paradox?

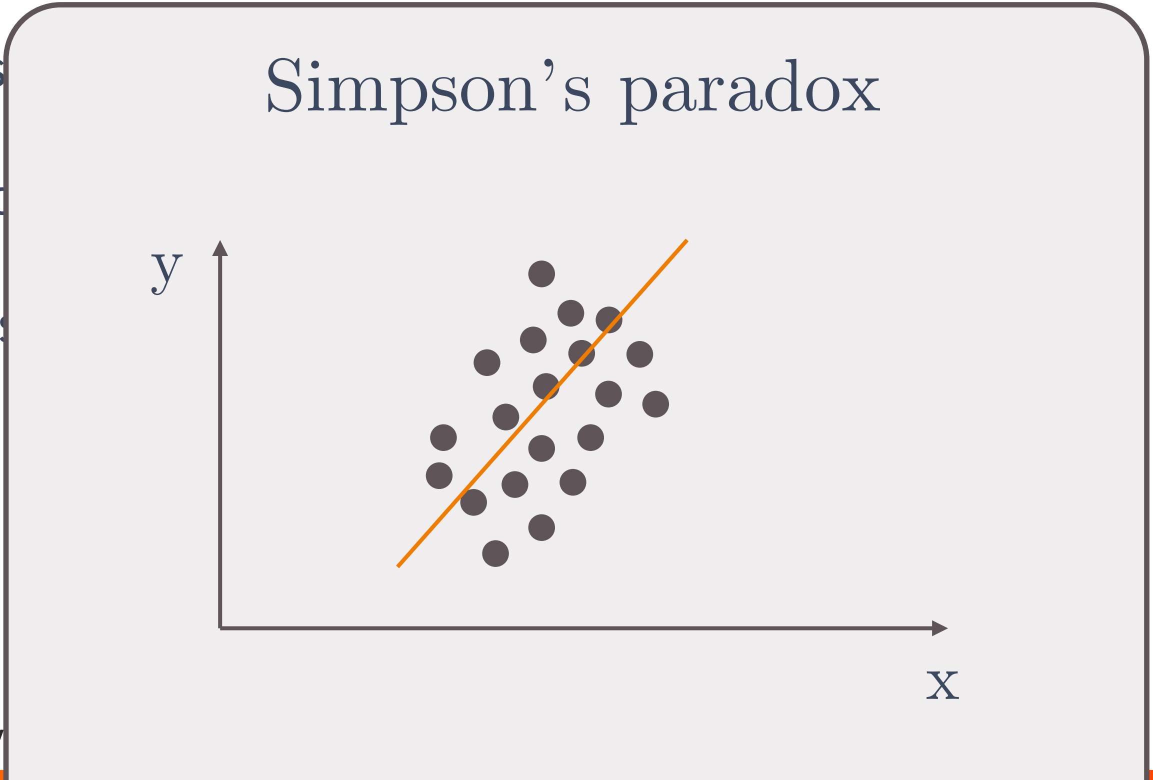
Berkeley admissions (1973)

- Admissions
- Larger perc
- Gender bia



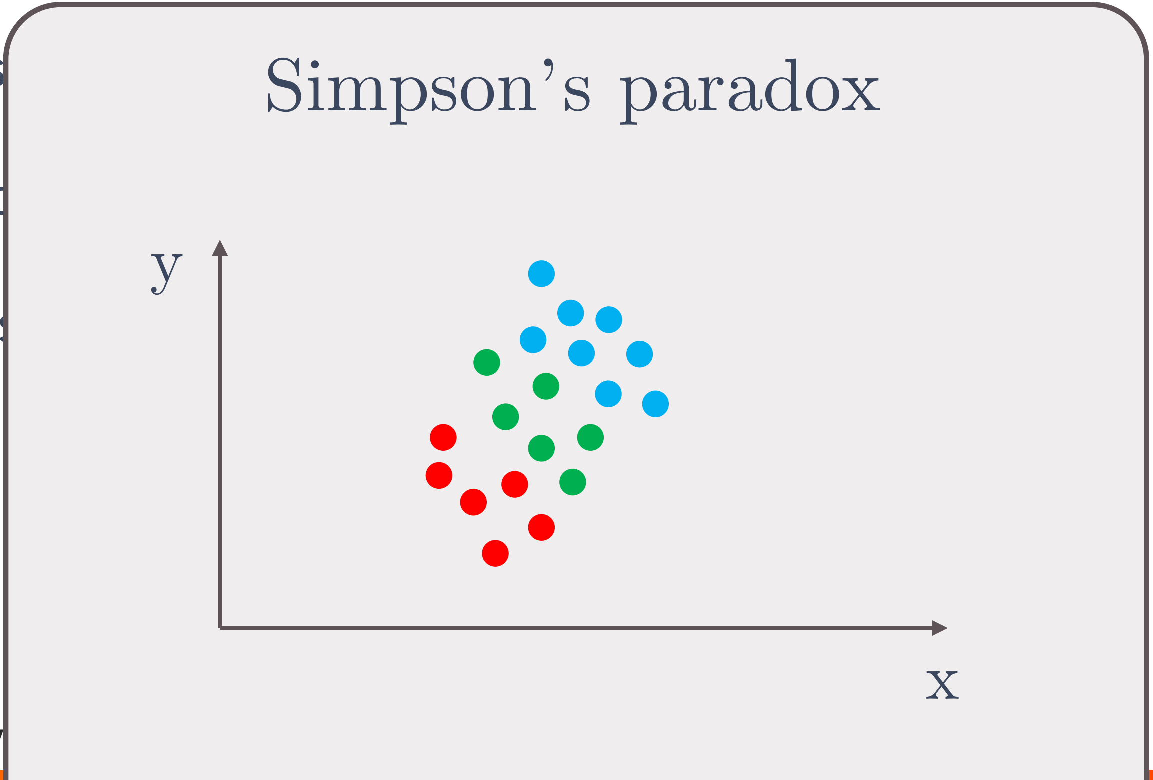
Berkeley admissions (1973)

- Admissions
- Larger perc
- Gender bias



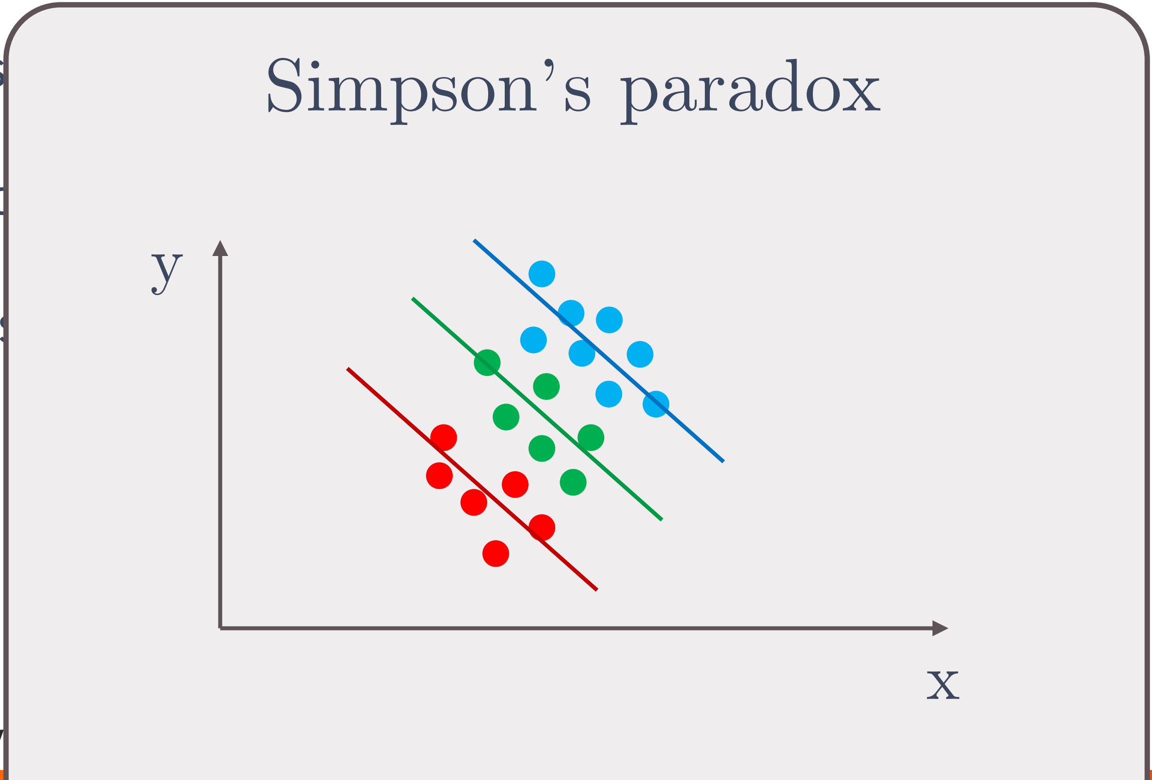
Berkeley admissions (1973)

- Admissions
- Larger perc
- Gender bias



Berkeley admissions (1973)

- Admissions
- Larger perc
- Gender bias



Berkeley admissions (1973)

- Admissions at UC Berkeley graduate schools in 1973
- Larger percentage of male applicants admitted
- Gender bias or Simpson's paradox?

Berkeley admissions (1973)

berkeley.csv

```
Year,Major,Gender,Admission
1973,C,F,Rejected
1973,B,M,Accepted
1973,Other,F,Accepted
1973,Other,M,Accepted
1973,Other,M,Rejected
1973,Other,M,Rejected
1973,F,F,Accepted
1973,Other,M,Accepted
1973,Other,M,Rejected
1973,A,M,Accepted
1973,Other,F,Rejected
1973,B,M,Accepted
1973,C,M,Rejected
1973,A,M,Rejected
1973,Other,M,Rejected
…
```

Berkeley admissions (1973)

berkeley.csv

12763 lines

```
Year,Major,Gender,Admission
1973,C,F,Rejected
1973,B,M,Accepted
1973,Other,F,Accepted
1973,Other,M,Accepted
1973,Other,M,Rejected
1973,Other,M,Rejected
1973,F,F,Accepted
1973,Other,M,Accepted
1973,Other,M,Rejected
1973,A,M,Accepted
1973,Other,F,Rejected
1973,B,M,Accepted
1973,C,M,Rejected
1973,A,M,Rejected
1973,Other,M,Rejected
…
```

Berkeley admissions (1973)

berkeley.csv

12763 lines

```
Year,Major,Gender,Admission
1973,C,F,Rejected
1973,B,M,Accepted
1973,Other,F,Accepted
1973,Other,M,Accepted
1973,Other,M,Rejected
1973,Other,M,Rejected
1973,F,F,Accepted
1973,Other,M,Accepted
1973,Other,M,Rejected
1973,A,M,Accepted
1973,Other,F,Rejected
1973,B,M,Accepted
1973,C,M,Rejected
1973,A,M,Rejected
1973,Other,M,Rejected
…
```

Year	Major	Gender	Admission
1973	C	F	Rejected
1973	B	M	Accepted
1973	Other	F	Accepted
1973	Other	M	Accepted
1973	Other	M	Rejected
1973	Other	M	Rejected
1973	F	F	Accepted
1973	Other	M	Accepted
1973	Other	M	Rejected
1973	A	M	Accepted
1973	Other	F	Rejected

Berkeley admissions (1973)

Year	Major	Gender	Admission
1973	C	F	Rejected
1973	B	M	Accepted
1973	Other	F	Accepted
1973	Other	M	Accepted
1973	Other	M	Rejected
1973	Other	M	Rejected
1973	F	F	Accepted
1973	Other	M	Accepted
1973	Other	M	Rejected
1973	A	M	Accepted
1973	Other	F	Rejected

Major	Gender	Admitted	Applicants
A	F	89	108
A	M	825	1138
B	F	17	25
B	M	353	560
C	F	201	593
C	M	120	325
D	F	131	375
D	M	138	417
E	F	94	393
E	M	53	191
F	F	25	341
F	M	22	373
Other	F	937	2486
Other	M	2227	5438

Berkeley admissions (1973)

Major	Gender	Admitted	Applicants
A	F	89	108
A	M	825	1138
B	F	17	25
B	M	353	560
C	F	201	593
C	M	120	325
D	F	131	375
D	M	138	417
E	F	94	393
E	M	53	191
F	F	25	341
F	M	22	373
Other	F	937	2486
Other	M	2227	5438

Major	Gender	Admitted	Applicants
A	F	89	108
A	M	825	1138
A	T	914	1246
B	F	17	25
B	M	353	560
B	T	370	585
C	F	201	593
C	M	120	325
C	T	321	918
D	F	131	375
D	M	138	417
D	T	269	792
E	F	94	393
E	M	53	191
E	T	147	584
F	F	25	341
F	M	22	373
F	T	47	714
Other	F	937	2486
Other	M	2227	5438
Other	T	3164	7924
Total	F	1494	4321
Total	M	3738	8442
Total	T	5232	12763

Berkeley admissions (1973)

Major	Gender	Admitted	Applicants
A	F	89	108
A	M	825	1138
A	T	914	1246
B	F	17	25
B	M	353	560
B	T	370	585
C	F	201	593
C	M	120	325
C	T	321	918
D	F	131	375
D	M	138	417
D	T	269	792
E	F	94	393
E	M	53	191
E	T	147	584
F	F	25	341
F	M	22	373
F	T	47	714
Other	F	937	2486
Other	M	2227	5438
Other	T	3164	7924
Total	F	1494	4321
Total	M	3738	8442
Total	T	5232	12763

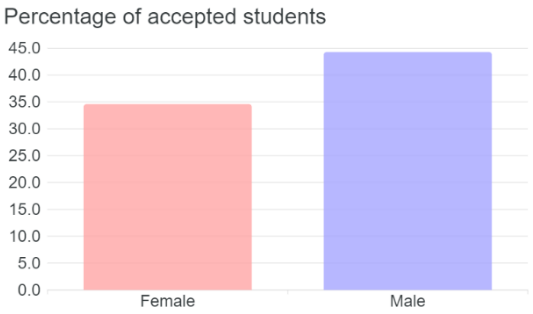
Major	Gender	Admitted	Applicants	%Admitted	%Applicants
A	F	89	108	82.4	2.5
A	M	825	1138	72.5	13.5
A	T	914	1246	73.4	9.76
B	F	17	25	68	0.579
B	M	353	560	63	6.63
B	T	370	585	63.2	4.58
C	F	201	593	33.9	13.7
C	M	120	325	36.9	3.85
C	T	321	918	35	7.19
D	F	131	375	34.9	8.68
D	M	138	417	33.1	4.94
D	T	269	792	34	6.21
E	F	94	393	23.9	9.1
E	M	53	191	27.7	2.26
E	T	147	584	25.2	4.58
F	F	25	341	7.33	7.89
F	M	22	373	5.9	4.42
F	T	47	714	6.58	5.59
Other	F	937	2486	37.7	57.5
Other	M	2227	5438	41	64.4
Other	T	3164	7924	39.9	62.1
Total	F	1494	4321	34.6	100
Total	M	3738	8442	44.3	100
Total	T	5232	12763	41	100

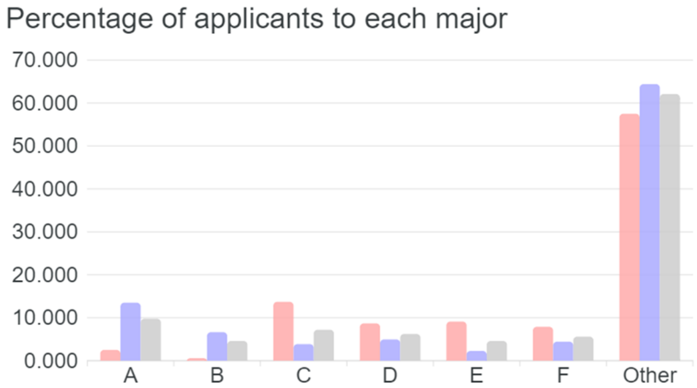
Berkeley admissions (1973)

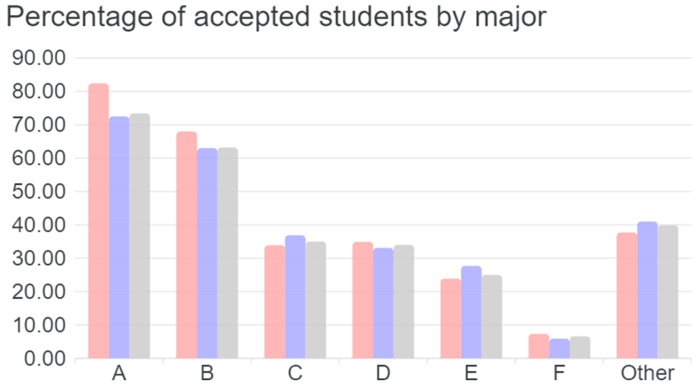
Major	Gender	Admitted	Applicants	%Admitted	%Applicants
A	F	89	108	82.4	2.5
A	M	825	1138	72.5	13.5
A	T	914	1246	73.4	9.76
B	F	17	25	68	0.579
B	M	353	560	63	6.63
B	T	370	585	63.2	4.58
C	F	201	593	33.9	13.7
C	M	120	325	36.9	3.85
C	T	321	918	35	7.19
D	F	131	375	34.9	8.68
D	M	138	417	33.1	4.94
D	T	269	792	34	6.21
E	F	94	393	23.9	9.1
E	M	53	191	27.7	2.26
E	T	147	584	25.2	4.58
F	F	25	341	7.33	7.89
F	M	22	373	5.9	4.42
F	T	47	714	6.58	5.59
Other	F	937	2486	37.7	57.5
Other	M	2227	5438	41	64.4
Other	T	3164	7924	39.9	62.1
Total	F	1494	4321	34.6	100
Total	M	3738	8442	44.3	100
Total	T	5232	12763	41	100

Berkeley admissions (1973)

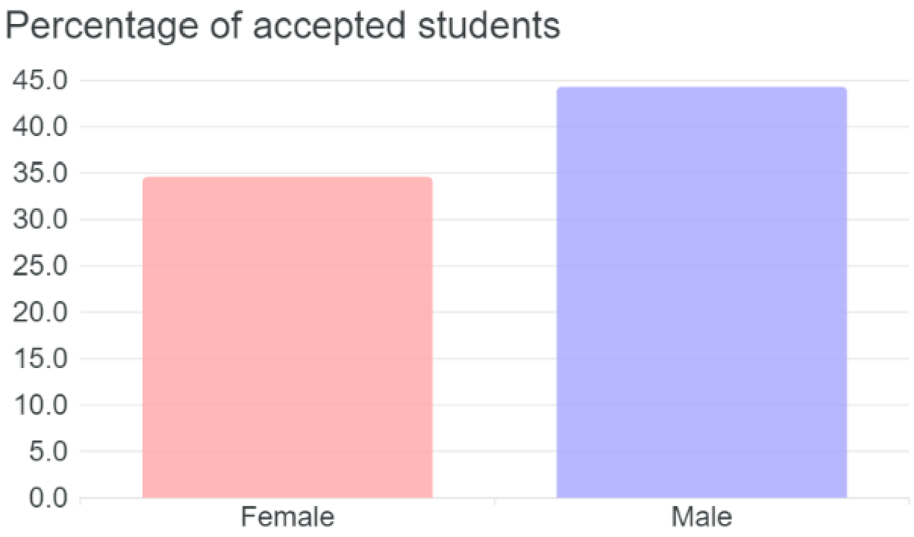
Major	Gender	Admitted	Applicants	%Admitted	%Applicants
A	F	89	108	82.4	2.5
A	M	825	1138	72.5	13.5
A	T	914	1246	73.4	9.76
B	F	17	25	68	0.579
B	M	353	560	63	6.63
B	T	370	585	63.2	4.58
C	F	201	593	33.9	13.7
C	M	120	325	36.9	3.85
C	T	321	918	35	7.19
D	F	131	375	34.9	8.68
D	M	138	417	33.1	4.94
D	T	269	792	34	6.21
E	F	94	393	23.9	9.1
E	M	53	191	27.7	2.26
E	T	147	584	25.2	4.58
F	F	25	341	7.33	7.89
F	M	22	373	5.9	4.42
F	T	47	714	6.58	5.59
Other	F	937	2486	37.7	57.5
Other	M	2227	5438	41	64.4
Other	T	3164	7924	39.9	62.1
Total	F	1494	4321	34.6	100
Total	M	3738	8442	44.3	100
Total	T	5232	12763	41	100

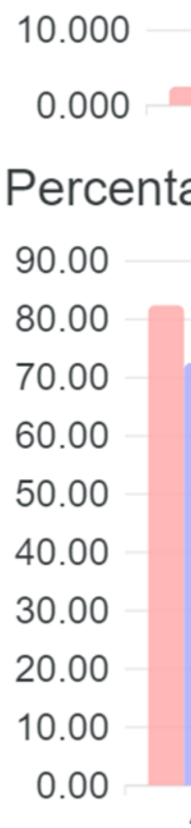


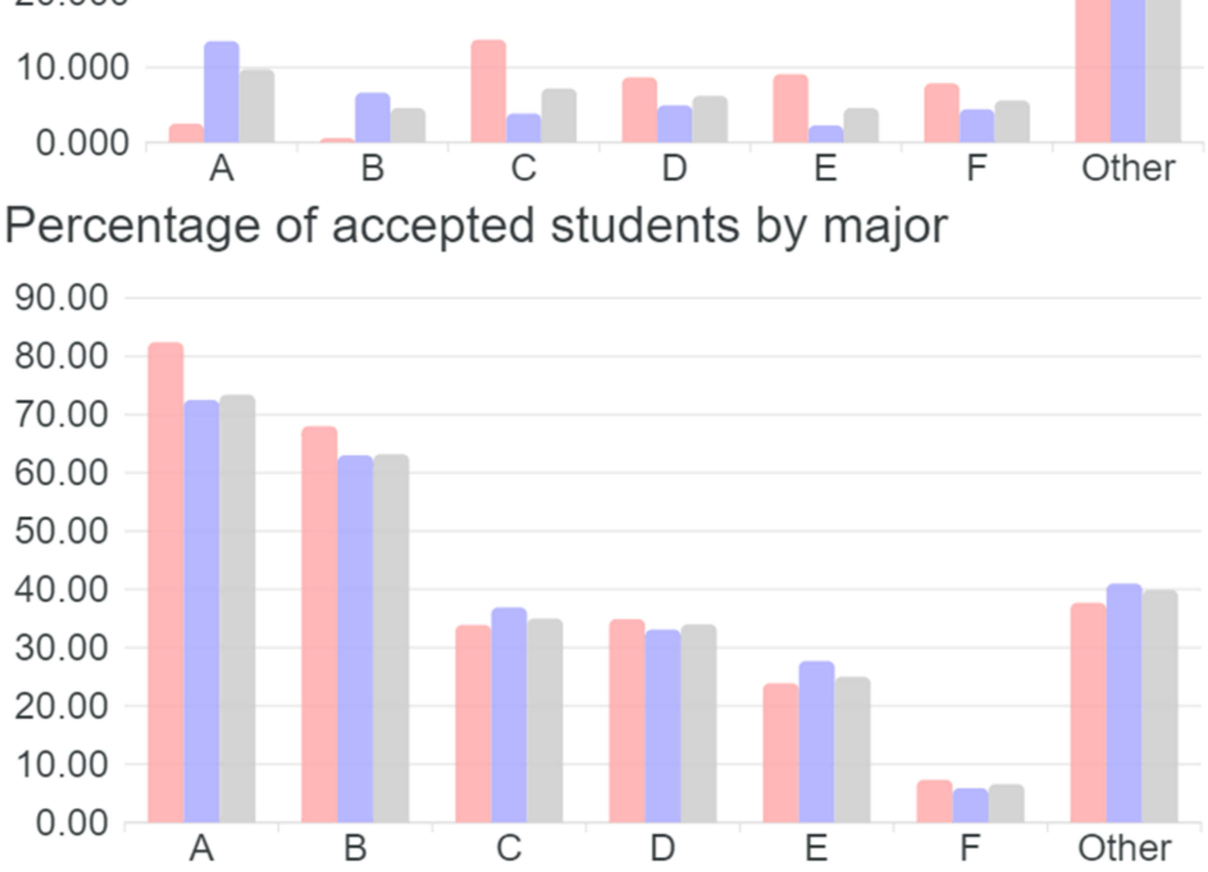




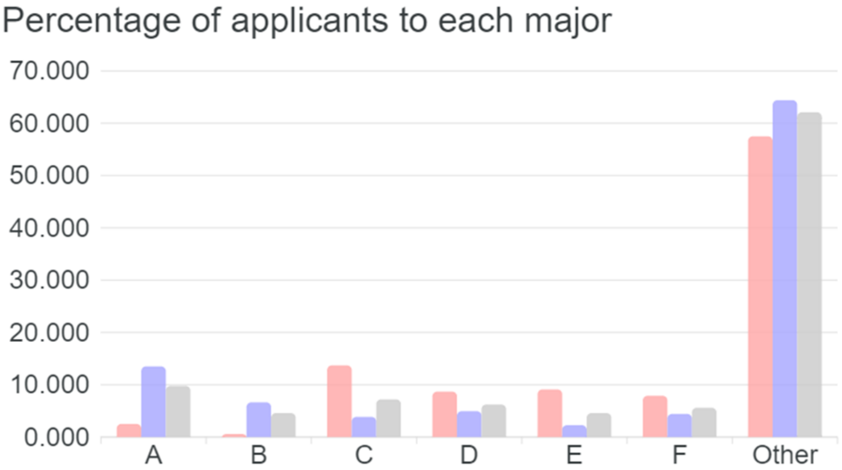
F	T	47	714	6.58	5.59
Other	F	937	2486	37.7	57.5
Other	M	2227	5438	41	64.4
Other	T	3164	7924	39.9	62.1
Total	F	1494	4321	34.6	100
Total	M	3738	8442	44.3	100
Total	T	5232	12763	41	100

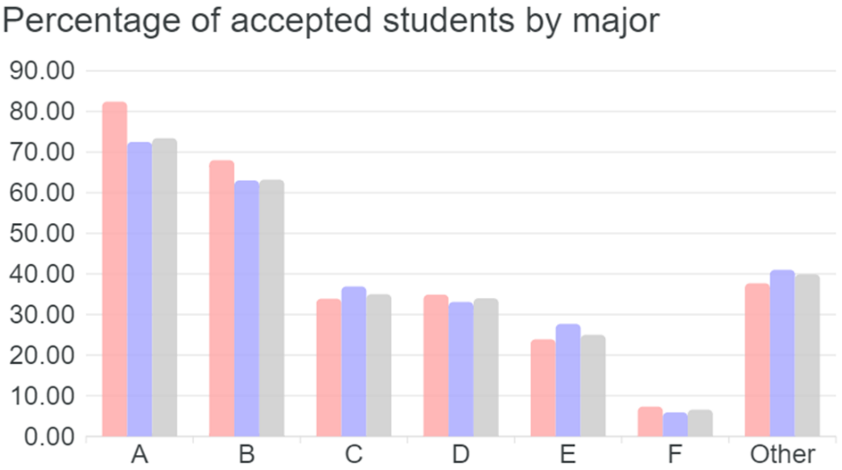





10.000
0.000
A
B
C
D
E
F
Other
Percentage of accepted students by major
90.00
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
A
B
C
D
E
F
Other

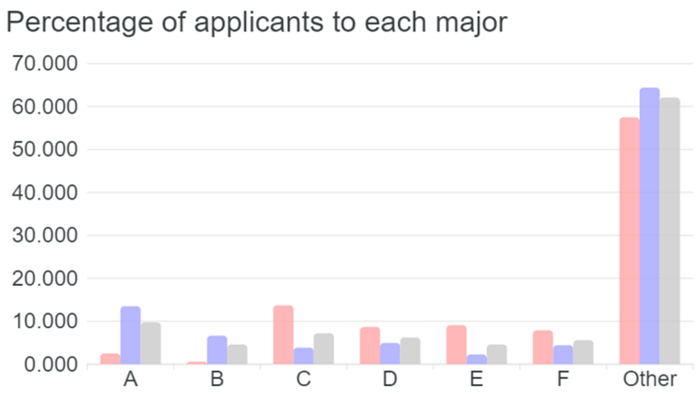
ted	%Applicants
.4	2.5
.5	13.5
.4	9.76
	0.579
	6.63
.2	4.58
.9	13.7
.9	3.85
	7.19
.9	8.68
.1	4.94
	6.21
.9	9.1
.7	2.26
.2	4.58
.33	7.89
.9	4.42
.58	5.59
.7	57.5
	64.4
.9	62.1
.6	100
.3	100
	100

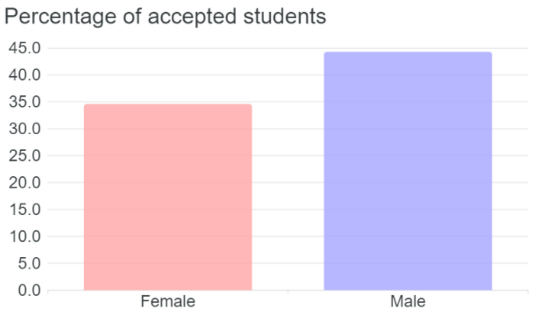


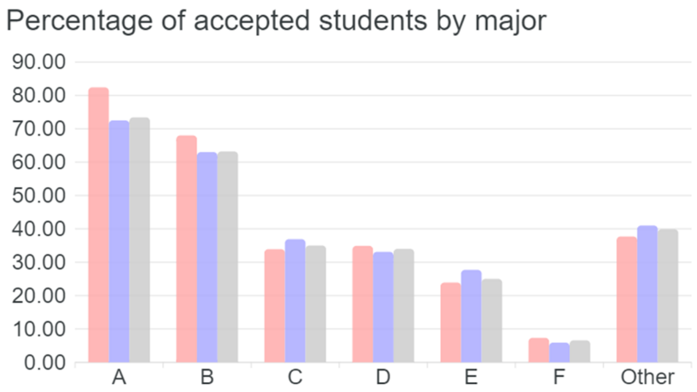


Berkeley admissions (1973)

Major	Gender	Admitted	Applicants	%Admitted	%Applicants
A	F	89	108	82.4	2.5
A	M	825	1138	72.5	13.5
A	T	914	1246	73.4	9.76
B	F	17	25	68	0.579
B	M	353	560	63	6.63
B	T	370	585	63.2	4.58
C	F	201	593	33.9	13.7
C	M	120	325	36.9	3.85
C	T	321	918	35	7.19
D	F	131	375	34.9	8.68
D	M	138	417	33.1	4.94
D	T	269	792	34	6.21
E	F	94	393	23.9	9.1
E	M	53	191	27.7	2.26
E	T	147	584	25.2	4.58
F	F	25	341	7.33	7.89
F	M	22	373	5.9	4.42
F	T	47	714	6.58	5.59
Other	F	937	2486	37.7	57.5
Other	M	2227	5438	41	64.4
Other	T	3164	7924	39.9	62.1
Total	F	1494	4321	34.6	100
Total	M	3738	8442	44.3	100
Total	T	5232	12763	41	100







Berkeley admissions (1973)

The code

Berkeley admissions (1973)

The code

```
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{⍵[⍋⍵;]}d[;2 3]{⍺,(+/('A'=⊃)¨⍵),≢⍵}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{⍺,+⌿⍵}⌸a[;3 4]
m←{⍵[⍋⍵;]}g⍪g[;1]{⍺,'T',+⌿⍵}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

```
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{ω[⍋ω;]}d[;2 3]{α,(+/('A'=⊃)¨ω),≢ω}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{α,+⌿ω}⌸a[;3 4]
m←{ω[⍋ω;]}g⍪g[;1]{α,'T',+⌿ω}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

```apl
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{⍵[⍋⍵;]}d[;2 3]{⍺,(+/('A'=⊃)¨⍵),≢⍵}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{⍺,+⌿⍵}⌸a[;3 4]
m←{⍵[⍋⍵;]}g⍪g[;1]{⍺,'T',+⌿⍵}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

```python
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

Year	Major	Gender	Admission
1973	C	F	Rejected
1973	B	M	Accepted
1973	Other	F	Accepted
1973	Other	M	Accepted
1973	Other	M	Rejected
1973	Other	M	Rejected
1973	F	F	Accepted
1973	Other	M	Accepted
1973	Other	M	Rejected
1973	A	M	Accepted
...	...	...	...

```
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{⍵[⍋⍵;]}d[;2 3]{⍺,(+/('A'=⊃)¨⍵),≢⍵}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{⍺,+⌿⍵}⌸a[;3 4]
m←{⍵[⍋⍵;]}g⍪g[;1]{⍺,'T',+⌿⍵}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

	Year	Major	Gender	Admission
0	1973	C	F	Rejected
1	1973	B	M	Accepted
2	1973	Other	F	Accepted
3	1973	Other	M	Accepted
4	1973	Other	M	Rejected
...	...	...	...	...
12758	1973	Other	M	Accepted
12759	1973	D	M	Accepted
12760	1973	Other	F	Rejected
12761	1973	Other	M	Rejected
12762	1973	Other	M	Accepted

```
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

Major	Gender	Admitted	Applicants
A	F	89	108
A	M	825	1138
B	F	17	25
B	M	353	560
C	F	201	593
C	M	120	325
D	F	131	375
D	M	138	417
E	F	94	393
E	M	53	191
F	F	25	341
F	M	22	373
Other	F	937	2486
Other	M	2227	5438

```
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{ω[⍋ω;]}d[;2 3]{α,(+/('A'=⊃)¨ω),≢ω}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{α,+⌿ω}⌸a[;3 4]
m←{ω[⍋ω;]}g⍪g[;1]{α,'T',+⌿ω}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

Major	Gender	Admitted	Applicants
A	F	89	108
	M	825	1138
B	F	17	25
	M	353	560
C	F	201	593
	M	120	325
D	F	131	375
	M	138	417
E	F	94	393
	M	53	191
F	F	25	341
	M	22	373
Other	F	937	2486
	M	2227	5438

```
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

Major	Gender	Admitted	Applicants
A	F	89	108
A	M	825	1138
A	T	914	1246
B	F	17	25
B	M	353	560
B	T	370	585
C	F	201	593
C	M	120	325
C	T	321	918
D	F	131	375
D	M	138	417
D	T	269	792
E	F	94	393
E	M	53	191
E	T	147	584
F	F	25	341
F	M	22	373
F	T	47	714
Other	F	937	2486
Other	M	2227	5438
Other	T	3164	7924
Total	F	1494	4321
Total	M	3738	8442
Total	T	5232	12763

```
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{⍵[⍋⍵;]}d[;2 3]{⍺,(+/('A'=⊃)¨⍵),≢⍵}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{⍺,+⌿⍵}⌸a[;3 4]
m←{⍵[⍋⍵;]}g⍪g[;1]{⍺,'T',+⌿⍵}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

		Admitted	Applicants
Major	Gender		
A	F	89	108
	M	825	1138
	T	914	1246
B	F	17	25
	M	353	560
	T	370	585
C	F	201	593
	M	120	325
	T	321	918
D	F	131	375
	M	138	417
	T	269	792
E	F	94	393
	M	53	191
	T	147	584
F	F	25	341
	M	22	373
	T	47	714
Other	F	937	2486
	M	2227	5438
	T	3164	7924
Total	F	1494	4321
	M	3738	8442
	T	5232	12763

```
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

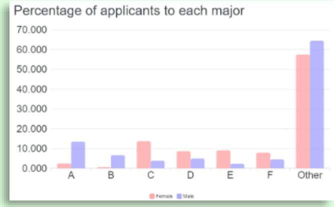
Major	Gender	Admitted	Applicants	%Admitted	%Applicants
A	F	89	108	82.4	2.5
A	M	825	1138	72.5	13.5
A	T	914	1246	73.4	9.76
B	F	17	25	68	0.579
B	M	353	560	63	6.63
B	T	370	585	63.2	4.58
C	F	201	593	33.9	13.7
C	M	120	325	36.9	3.85
C	T	321	918	35	7.19
D	F	131	375	34.9	8.68
D	M	138	417	33.1	4.94
D	T	269	792	34	6.21
E	F	94	393	23.9	9.1
E	M	53	191	27.7	2.26
E	T	147	584	25.2	4.58
F	F	25	341	7.33	7.89
F	M	22	373	5.9	4.42
F	T	47	714	6.58	5.59
Other	F	937	2486	37.7	57.5
Other	M	2227	5438	41	64.4
Other	T	3164	7924	39.9	62.1
Total	F	1494	4321	34.6	100
Total	M	3738	8442	44.3	100
Total	T	5232	12763	41	100

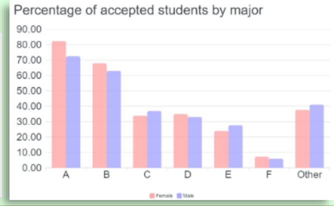
APL

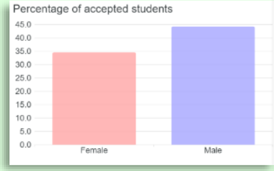
```
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{⍵[⍋⍵;]}d[;2 3]{⍺,(+/('A'=⊃)¨⍵),≢⍵}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{⍺,+⌿⍵}⌸a[;3 4]
m←{⍵[⍋⍵;]}g⍪g[;1]{⍺,'T',+⌿⍵}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

Major	Gender	Admitted	Applicants	PctAdmitted	PctApplicants
A	F	89	108	82.407407	2.499421
	M	825	1138	72.495606	13.480218
	T	914	1246	73.354735	9.762595
B	F	17	25	68.000000	0.578570
	M	353	560	63.035714	6.633499
	T	370	585	63.247863	4.583562
C	F	201	593	33.895447	13.723675
	M	120	325	36.923077	3.849799
	T	321	918	34.967320	7.192666
D	F	131	375	34.933333	8.678547
	M	138	417	33.093525	4.939588
	T	269	792	33.964646	6.205438
E	F	94	393	23.918575	9.095117
	M	53	191	27.748691	2.262497
	T	147	584	25.171233	4.575727
F	F	25	341	7.331378	7.891692
	M	22	373	5.898123	4.418384
	T	47	714	6.582633	5.594296
Other	F	937	2486	37.691070	57.532978
	M	2227	5438	40.952556	64.416015
	T	3164	7924	39.929329	62.085717
Total	F	1494	4321	34.575330	100.000000
	M	3738	8442	44.278607	100.000000
	T	5232	12763	40.993497	100.000000

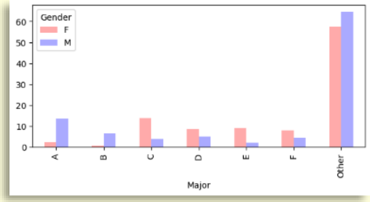
```python
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

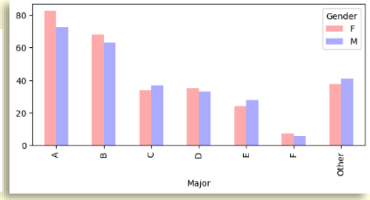


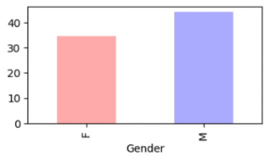




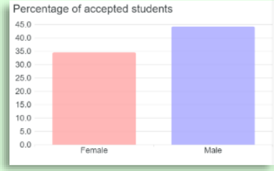

```apl
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{ω[⍋ω;]}d[;2 3]{α,(+/('A'=⊃)¨ω),≢ω}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{α,+⌿ω}⌸a[;3 4]
m←{ω[⍋ω;]}g⍪g[;1]{α,'T',+⌿ω}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

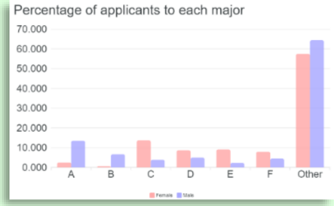


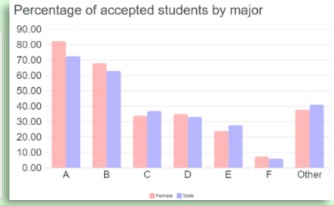




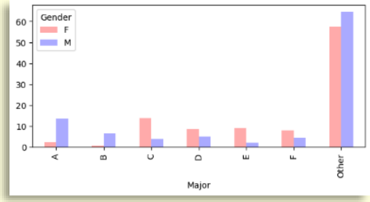

```python
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

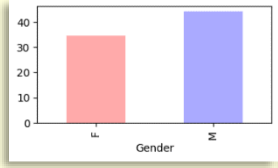


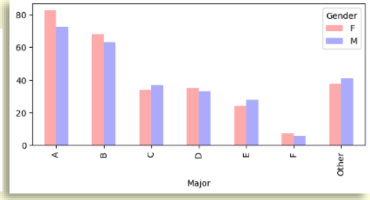





```apl
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{⍵[⍋⍵;]}d[;2 3]{⍺,(+/('A'=⊃)¨⍵),≢⍵}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{⍺,+⌿⍵}⌸a[;3 4]
m←{⍵[⍋⍵;]}g⍪g[;1]{⍺,'T',+⌿⍵}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```








```python
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

Berkeley admissions (1973)

The code

```apl
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{⍵[⍋⍵;]}d[;2 3]{⍺,(+/('A'=⊃)¨⍵),≢⍵}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{⍺,+⌿⍵}⌸a[;3 4]
m←{⍵[⍋⍵;]}g⍪g[;1]{⍺,'T',+⌿⍵}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

```python
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

Berkeley admissions (1973)

The code

```apl
⍝ read data
d h←⎕CSV'berkeley.csv' ''1 1
⍝ group by gender and by major
a←{⍵[⍋⍵;]}d[;2 3]{⍺,(+/('A'=⊃)¨⍵),≢⍵}⌸d[;4]
⍝ totals by gender and by major
g←a⍪(⊂'Total'),a[;2]{⍺,+⌿⍵}⌸a[;3 4]
m←{⍵[⍋⍵;]}g⍪g[;1]{⍺,'T',+⌿⍵}⌸g[;3 4]
⍝ admission and applicants ratios
ar←m,100×m[;3]÷m[;4]
mr←ar,100×ar[;4]÷(≢ar)⍴¯3↑ar[;4]
```

```python
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
mr = ar.assign(PctApplicants = 100*ar.Applicants/ar.loc['Total']['Applicants'])
```

https://github.com/yiyus/data-science-in-APL/

Example 2

Fisher's Iris dataset (1936)

- Measures of four different flower features
- Three classes of iris flower
- The goal is to classify flowers in basis to these features

Fisher's Iris dataset (1936)

- Measures o
- Three class
- The goal is

Iris

Danielle Langlois. 2005. CC-BY-SA

Fisher's Iris dataset (1936)

- Measures o
- Three class
- The goal is

Iris

Fisher's Iris dataset (1936)

- Measures o
- Three class
- The goal is

Iris

Fisher's Iris dataset (1936)

- Measures o
- Three class
- The goal is

Iris

Iris-setosa, Iris-versicolor, Iris-virginica

Fisher's Iris dataset (1936)

- Measures of four different flower features
- Three classes of iris flower
- The goal is to classify flowers in basis to these features

Fisher's Iris dataset (1936)

The code

```apl
⍝ statistics
AVG←
STD←
PCT←
PCC←
```

```apl
⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(ω[;1],∘αα⌸1↓[2]ω)⍪(⊂'Total'),αα1↓[2]ω}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉ω}_A d
c←⊂⍤2{∘.PCC⍨↓⍉ω}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, names=cols+['class'])
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

```
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(ω[;1],∘αα⌸1↓[2]ω)⍪(⊂'Total'),αα1↓[2]ω}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉ω}_A d
c←⊂⍤2{∘.PCC⍨↓⍉ω}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, names=cols+['class'])
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

```
⍝ statistics
AVG←
STD←
PCT←
PCC←
⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(ω[;1],∘αα⌸1↓[2]ω)⍪(⊂'Total'),αα1↓[2]ω}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉ω}_A d
c←⊂⍤2{∘.PCC⍨↓⍉ω}_A d
c,←⊂(⊂'Class'),(⊂ι⍨d[;1])PCC¨↓⍉1↓[2]d
```

```
⍝ statistics
AVG←…         ⍝ average
STD←…         ⍝ standard deviation
PCT←…         ⍝ percentiles
PCC←…         ⍝ Pearson’s correlation coefficient
⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂∪⍨d[;1])PCC¨↓⍉1↓[2]d
```

```
⍝ statistics
AVG←+⌿÷≢
STD←(2*∘÷⍨+⌿÷¯1+≢)2*⍨⊢-⍤1+⌿÷≢
PCT←{((2÷⍨+/)⊢⌷⍨∘⊂⍋⌷⍨∘⊂∘⌈100÷⍨⍺×0 1+≢)⍵}
PCC←+.×⍥((⊢÷2*∘÷⍨+.×⍨)⊢-+⌿÷≢)
⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total')
⍝ summary, percentiles and correla
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

```
⍝ statistics
AVG←stats.Average
STD←stats.StdDev
PCT←stats.Percentile
PCC←stats.Correlation
⍝ read data
d←¯
⍝ a
_A←
⍝ s
s←(
p←{
c←⊂
c,←
```

```
:Namespace stats
    Average←{(+⌿⍵)÷≢⍵}
    StdDev←{(÷2)*⍨(+.×⍨⍵-Average⍵)÷(≢⍵)-1}
    StdScore←{⍵÷(÷2)*⍨(+.×⍨⍵-Average⍵)}
    Correlation←{⍺+.×⍥StdScore⍵} ⍝ Pearson's coeff
    Percentile←{
        i←⌈(⍺÷100)×0 1+≢⍵ ⋄ v←(⊂(⊂i)⌷⍋⍵)⌷⍵ ⋄ (+/v)÷2
    }
:EndNamespace
```

```
⍝ statistics
AVG←stats.Average
STD←stats.StdDev
PCT←stats.Percentile
PCC←stats.Correlation
⍝ read data
d←¯
⍝ a
_A←
⍝ s
s←(
p←{
c←⊂
c,←
```

```
:Namespace stats
    Average←{(+⌿⍵)÷≢⍵} ⍝ DANGER: not valid if 0=≢⍵
    StdDev←{(÷2)*⍨(+.×⍨⍵-Average⍵)÷(≢⍵)-1}
    StdScore←{⍵÷(÷2)*⍨(+.×⍨⍵-Average⍵)}
    Correlation←{⍺+.×⍥StdScore⍵} ⍝ Pearson's coeff
    Percentile←{
        i←⌈(⍺÷100)×0 1+≢⍵ ⋄ v←(⊂(⊂i)⌷⍋⍵)⌷⍵ ⋄ (+/v)÷2
    }
:EndNamespace
```

TamStat

TAMING STATISTICS

https://tamstat.dyalog.com

```
⍝ statistics
AVG←
STD←
PCT←
PCC←
⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

```
⍝ statistics
AVG←
STD←
PCT←
PCC←
⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

TamStat
TAMING STATISTICS
https://tamstat.dyalog.com

mean
0∘sdev
percentile
corr

```apl
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, names=cols+['class'])
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

```apl
⍝ statistics
AVG←
STD←
PCT←
PCC←
⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

Iris-setosa	5.1	3.5	1.4	0.2
Iris-setosa	4.9	3	1.4	0.2
Iris-setosa	4.7	3.2	1.3	0.2
Iris-setosa	4.6	3.1	1.5	0.2
Iris-setosa	5	3.6	1.4	0.2
Iris-setosa	5.4	3.9	1.7	0.4
Iris-setosa	4.6	3.4	1.4	0.3
Iris-setosa	5	3.4	1.5	0.2
Iris-setosa	4.4	2.9	1.4	0.2
Iris-setosa	4.9	3.1	1.5	0.1
Iris-setosa	5.4	3.7	1.5	0.2
Iris-setosa	4.8	3.4	1.6	0.2
Iris-setosa	4.8	3	1.4	0.1
Iris-setosa	4.3	3	1.1	0.1

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, names=cols+['class'])
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

```apl
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, names=cols+['class'])
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

```apl
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(ω[;1],∘αα⌸1↓[2]ω)⍪(⊂'Total'),αα1↓[2]ω}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉ω}_A d
c←⊂⍤2{∘.PCC⍨↓⍉ω}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

class	sl25	sl50	sl75	sw25	sw50	sw75	pl25	pl50	pl75	pw25	pw50	pw75
Iris-setosa	4.8	3.1	1.4	0.2	5	3.4	1.5	0.2	5.2	3.7	1.6	0.3
Iris-versicolor	5.6	2.5	4	1.2	5.9	2.8	4.35	1.3	6.3	3	4.6	1.5
Iris-virginica	6.2	2.8	5.1	1.8	6.5	3	5.55	2	7	3.2	5.9	2.3
Total	5.1	2.8	1.6	0.3	5.8	3	4.35	1.3	6.4	3.3	5.1	1.8

class	⌊sl	⌊sw	⌊pl	⌊pw	Asl	Asw	Apl	Apw	Ssl	Ssw	Spl	Spw	⌈sl	⌈sw	⌈pl	⌈pw
Iris-setosa	4.3	2.3	1	0.1	5.01	3.42	1.46	0.244	0.352	0.381	0.174	0.107	5.8	4.4	1.9	0.6
Iris-versicolor	4.9	2	3	1	5.94	2.77	4.26	1.33	0.516	0.314	0.47	0.198	7	3.4	5.1	1.8
Iris-virginica	4.9	2.2	4.5	1.4	6.59	2.97	5.55	2.03	0.636	0.322	0.552	0.275	7.9	3.8	6.9	2.5
Total	4.3	2	1	0.1	5.84	3.05	3.76	1.2	0.828	0.434	1.76	0.763	7.9	4.4	6.9	2.5

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, names=cols+['class'])
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

```
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(ω[;1],∘αα⌸1↓[2]ω),(⊂'Total'),αα1↓[2]ω}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉ω}_A d
c←⊂⍤2{∘.PCC⍨↓⍉ω}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

class	sl25	sl50	sl75	sw25	sw50	sw75	pl25	pl50	pl75	pw25	pw50	pw75
Iris-setosa	4.8	3.1	1.4	0.2	5	3.4	1.5	0.2	5.2	3.7	1.6	0.3
Iris-versicolor	5.6	2.5	4	1.2	5.9	2.8	4.35	1.3	6.3	3	4.6	1.5
Iris-virginica	6.2	2.8	5.1	1.8	6.5	3	5.55	2	7	3.2	5.9	2.3
Total	5.1	2.8	1.6	0.3	5.8	3	4.35	1.3	6.4	3.3	5.1	1.8

class	⌊sl	⌊sw	⌊pl	⌊pw	Asl	Asw	Apl	Apw	Ssl	Ssw	Spl	Spw	⌈sl	⌈sw	⌈pl	⌈pw
Iris-setosa	4.3	2.3	1	0.1	5.01	3.42	1.46	0.244	0.352	0.381	0.174	0.107	5.8	4.4	1.9	0.6
Iris-versicolor	4.9	2	3	1	5.94	2.77	4.26	1.33	0.516	0.314	0.47	0.198	7	3.4	5.1	1.8
Iris-virginica	4.9	2.2	4.5	1.4	6.59	2.97	5.55	2.03	0.636	0.322	0.552	0.275	7.9	3.8	6.9	2.5
Total	4.3	2	1	0.1	5.84	3.05	3.76	1.2	0.828	0.434	1.76	0.763	7.9	4.4	6.9	2.5

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)],
    return pd.concat([df.groupby('cl
# summary, percentiles and correlati
s = [A(df, x) for x in ['min', 'max'
p = [A(df, lambda d: d.quantile(q=x)
ct = pd.concat([df.corr()], keys=['T
c = pd.concat([df.groupby('class').c
tt = df.corrwith(pd.Series(pd.factor
cc = pd.concat([tt], keys=[('Total',
c = pd.concat([c, cc])
```

```
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

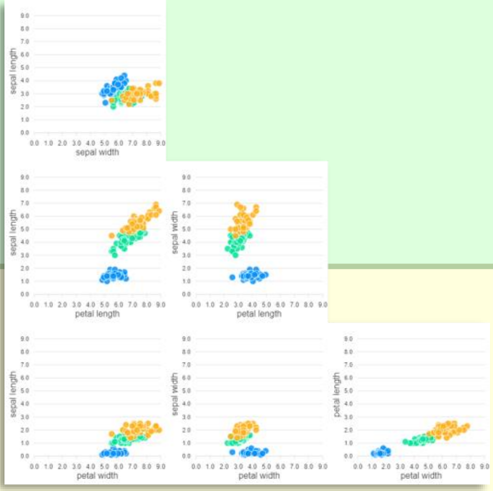
class	sl	sw	pl	pw
Iris-setosa	1	0.747	0.264	0.279
Iris-setosa	0.747	1	0.177	0.28
Iris-setosa	0.264	0.177	1	0.306
Iris-setosa	0.279	0.28	0.306	1
Iris-versicolor	1	0.526	0.754	0.546
Iris-versicolor	0.526	1	0.561	0.664
Iris-versicolor	0.754	0.561	1	0.787
Iris-versicolor	0.546	0.664	0.787	1
Iris-virginica	1	0.457	0.864	0.281
Iris-virginica	0.457	1	0.401	0.538
Iris-virginica	0.864	0.401	1	0.322
Iris-virginica	0.281	0.538	0.322	1
Total	1	¯0.109	0.872	0.818
Total	¯0.109	1	¯0.421	¯0.357
Total	0.872	¯0.421	1	0.963
Total	0.818	¯0.357	0.963	1
Class	0.783	¯0.419	0.949	0.956

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, na
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

```apl
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

class	sl	sw	pl	pw
Iris-setosa	1	0.747	0.264	0.279
Iris-setosa	0.747	1	0.177	0.28
Iris-setosa	0.264	0.177	1	0.306
Iris-setosa	0.279	0.28	0.306	1
Iris-versicolor	1	0.526	0.754	0.546
Iris-versicolor	0.526	1	0.561	0.664
Iris-versicolor	0.754	0.561	1	0.787
Iris-versicolor	0.546	0.664	0.787	1
Iris-virginica	1	0.457	0.864	0.281
Iris-virginica	0.457	1	0.401	0.538
Iris-virginica	0.864	0.401	1	0.322
Iris-virginica	0.281	0.538	0.322	1
Total	1	¯0.109	0.872	0.818
Total	¯0.109	1	¯0.421	¯0.357
Total	0.872	¯0.421	1	0.963
Total	0.818	¯0.357	0.963	1
Class	0.783	¯0.419	0.949	0.956

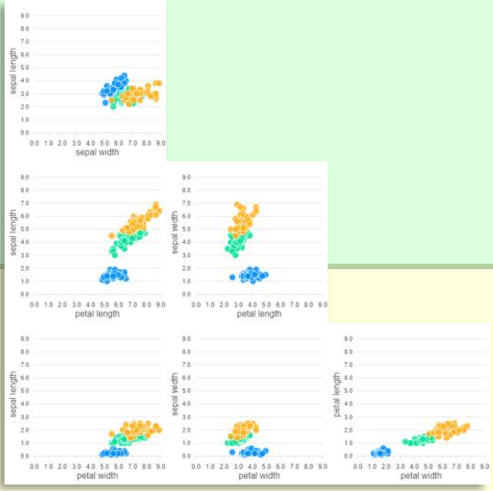



```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, na
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

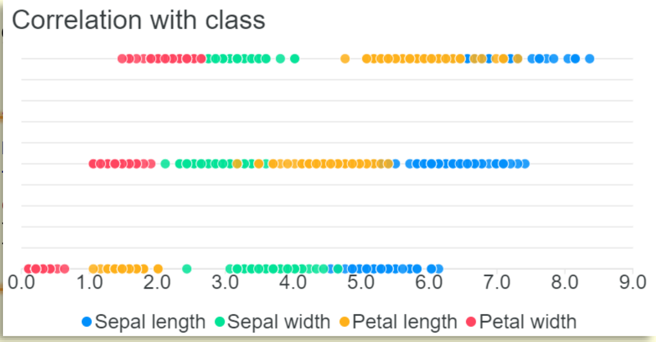
```
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

class	sl	sw	pl	pw
Iris-setosa	1	0.747	0.264	0.279
Iris-setosa	0.747	1	0.177	0.28
Iris-setosa	0.264	0.177	1	0.306
Iris-setosa	0.279	0.28	0.306	1
Iris-versicolor	1	0.526	0.754	0.546
Iris-versicolor	0.526	1	0.561	0.664
Iris-versicolor	0.754	0.561	1	0.787
Iris-versicolor	0.546	0.664	0.787	1
Iris-virginica	1	0.457	0.864	0.281
Iris-virginica	0.457	1	0.401	0.538
Iris-virginica	0.864	0.401	1	0.322
Iris-virginica	0.281	0.538	0.322	1
Total	1	¯0.109	0.872	0.818
Total	¯0.109	1	¯0.421	¯0.357
Total	0.872	¯0.421	1	0.963
Total	0.818	¯0.357	0.963	1
Class	0.783	¯0.419	0.949	0.956



```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, na
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').ag
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean'
p = [A(df, lambda d: d.quantile(q=x)) for x
ct = pd.concat([df.corr()], keys=['Total'],
c = pd.concat([df.groupby('class').corr(), c
tt = df.corrwith(pd.Series(pd.factorize(df['
cc = pd.concat([tt], keys=[('Total', 'class'
c = pd.concat([c, cc])
```

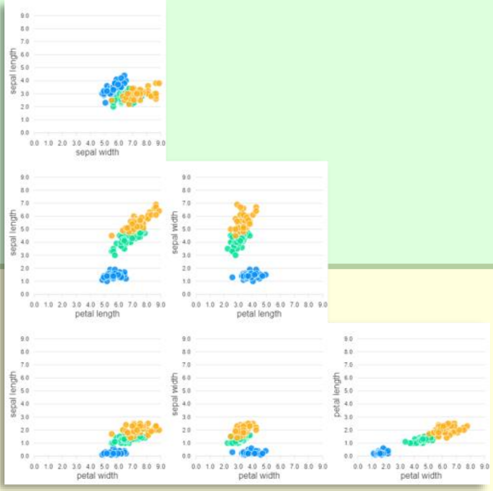


eWC

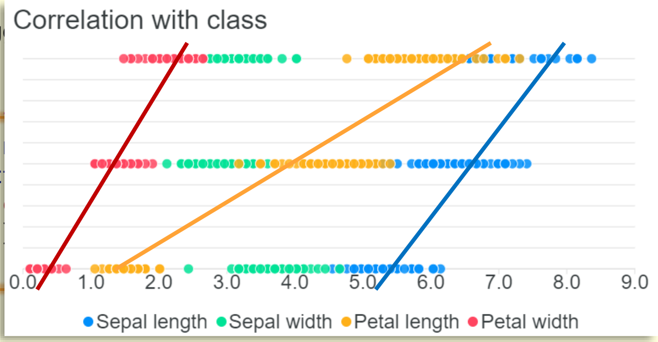
```
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT↓⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

class	sl	sw	pl	pw
Iris-setosa	1	0.747	0.264	0.279
Iris-setosa	0.747	1	0.177	0.28
Iris-setosa	0.264	0.177	1	0.306
Iris-setosa	0.279	0.28	0.306	1
Iris-versicolor	1	0.526	0.754	0.546
Iris-versicolor	0.526	1	0.561	0.664
Iris-versicolor	0.754	0.561	1	0.787
Iris-versicolor	0.546	0.664	0.787	1
Iris-virginica	1	0.457	0.864	0.281
Iris-virginica	0.457	1	0.401	0.538
Iris-virginica	0.864	0.401	1	0.322
Iris-virginica	0.281	0.538	0.322	1
Total	1	¯0.109	0.872	0.818
Total	¯0.109	1	¯0.421	¯0.357
Total	0.872	¯0.421	1	0.963
Total	0.818	¯0.357	0.963	1
Class	0.783	¯0.419	0.949	0.956




```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, na
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').ag
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean'
p = [A(df, lambda d: d.quantile(q=x)) for x
ct = pd.concat([df.corr()], keys=['Total'],
c = pd.concat([df.groupby('class').corr(), c
tt = df.corrwith(pd.Series(pd.factorize(df['
cc = pd.concat([tt], keys=[('Total', 'class'
c = pd.concat([c, cc])
```



eWC
APEXCHARTS

Fisher’s Iris dataset (1936)

The code

```apl
⍝ statistics
AVG←
STD←
PCT←
PCC←

⍝ read data
d←¯1⌽⎕CSV'iris.csv' ''4
⍝ aggregate with total
_A←{(⍵[;1],∘⍺⍺⌸1↓[2]⍵)⍪(⊂'Total'),⍺⍺1↓[2]⍵}
⍝ summary, percentiles and correlation
s←(⌊⌿,AVG,STD,⌈⌿)_A d
p←{,25 50 75∘.PCT⍉⍵}_A d
c←⊂⍤2{∘.PCC⍨↓⍉⍵}_A d
c,←⊂(⊂'Class'),(⊂⍳⍨d[;1])PCC¨↓⍉1↓[2]d
```

```python
# read data
cols = ['sl', 'sw', 'pl', 'pw']
df = pd.read_csv("iris.csv", header=None, names=cols+['class'])
# aggregate with total
def A(df, a):
    t = pd.concat([df[cols].agg(a)], keys=['Total'], names=['class']).unstack()
    return pd.concat([df.groupby('class').agg(a), t])
# summary, percentiles and correlation
s = [A(df, x) for x in ['min', 'max', 'mean', 'std']]
p = [A(df, lambda d: d.quantile(q=x)) for x in [0.25, 0.50, 0.75]]
ct = pd.concat([df.corr()], keys=['Total'], names=['class'])
c = pd.concat([df.groupby('class').corr(), ct])
tt = df.corrwith(pd.Series(pd.factorize(df['class'])[0]))
cc = pd.concat([tt], keys=[('Total', 'class')], names=['class', '']).unstack()
c = pd.concat([c, cc])
```

Example 3

Google trends (last 5 years)

- Example of time-series analysis
- Searches of “scotch” during last 5 years
- Interest each week, relative to maximum (100)

  https://trends.google.com/trends/explore?date=today%205-y&q=scotch

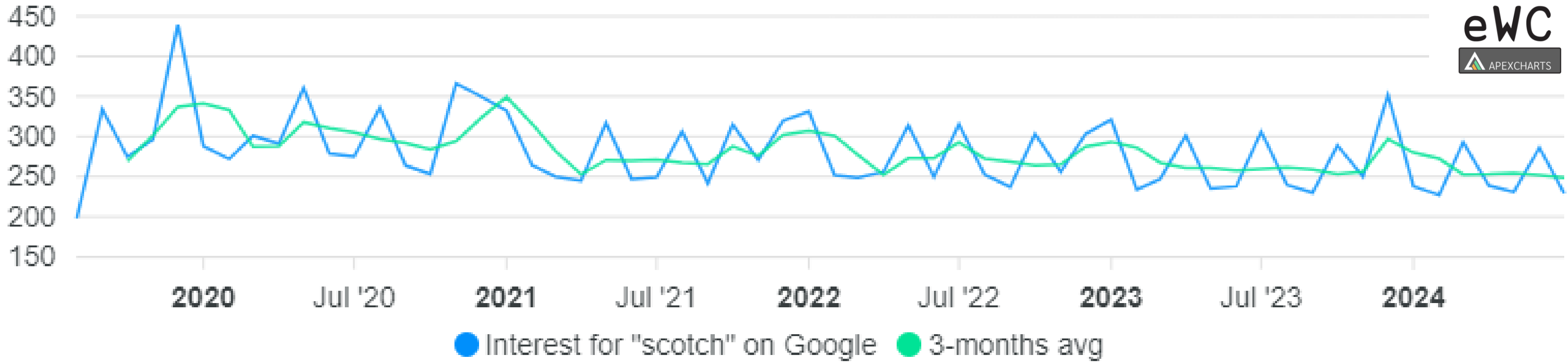
Google trends (last 5 years)

Total	2019	2020	2021	2022	2023	2024
1	·	288	332	331	321	238
2	·	272	264	252	234	227
3	·	301	250	249	247	293
4	·	291	245	256	301	239
5	·	361	317	314	235	231
6	·	279	247	250	238	286
7	·	275	249	315	306	229
8	198	336	306	253	240	111
9	334	264	242	237	230	·
10	275	253	315	303	289	·
11	296	366	271	256	250	·
12	439	350	320	303	352	·

Change	2019	2020	2021	2022	2023	2024
1	·	¯151	¯18	11	18	¯114
2	·	¯16	¯68	¯79	¯87	¯11
3	·	29	¯14	¯3	13	66
4	·	¯10	¯5	7	54	¯54
5	·	70	72	58	¯66	¯8
6	·	¯82	¯70	¯64	3	55
7	·	¯4	2	65	68	¯57
8	·	61	57	¯62	¯66	¯118
9	136	¯72	¯64	¯16	¯10	·
10	¯59	¯11	73	66	59	·
11	21	113	¯44	¯47	¯39	·
12	143	¯16	49	47	102	·

month	min	avg	max	total
1	238	302	332	1510
2	227	250	272	1249
3	247	268	301	1340
4	239	266	301	1332
5	231	292	361	1458
6	238	260	286	1300
7	229	275	315	1374
8	111	241	336	1444
9	230	261	334	1307
10	253	287	315	1435
11	250	288	366	1439
12	303	353	439	1764

year	min	avg	max	total	change
2019	198	308	439	1542	·
2020	253	303	366	3636	2094
2021	242	280	332	3358	¯278
2022	237	277	331	3319	¯39
2023	230	270	352	3243	¯76
2024	111	232	293	1854	¯1389



Dyalog for data science https://github.com/yiyus/data-science-in-APL/

DYALOG

Google trends (last 5 years)

The code

```apl
⍝ read data
(ds n)←⎕CSV⍠'Invert'2⊢(3↓⊃⎕NGET'google-scotch.csv'1)'N'4
⍝ dates
d←{⍲/(∧/∊∘(⎕D,'-'))¨⍵:⎕SIGNAL 11 ⋄ ↑'-'(⍎¨≠⊆⊢)¨⍵}ds

⍝ group by month and year, summary and relative change
t←d[;1 2],∘(+/)⌸n
s←{⍵[⍋⍵;]}¨{t[;⍵],∘(⌊⌿,(+⌿÷≢),⌈⌿,+⌿)⌸t[;3]}¨⍳2
c←{¯2-/t[;⍵]+/⍤⊢⌸t[;3]}¨⍳¨⍳2
```

```python
# read data
df = pd.read_csv("google-scotch.csv", header=1)
# dates
d = df.assign(Week = pd.to_datetime(df['Week'])).set_index('Week')

# group by month and year, summary and relative change
a = d.groupby([d.index.year, d.index.month]).agg(Total=(d.columns[0], 'sum'))
a.index.names = ['Year', 'Month']
def A(df, by):
    (m, a, M, t) = [('Total',x) for x in ['min', 'mean', 'max', 'sum']]
    return df.reset_index().groupby(by).agg(min=m, mean=a, max=M, total=t)
m = A(a, 'Month')
y = A(a, 'Year')
y = y.assign(change=y['total'].diff())
tm = a.unstack(level=0)
cm = a.diff().unstack(level=0)
```

```
⍝ read data
(ds n)←⎕CSV⍠'Invert'2⊢(3↓⊃⎕NGET'google-scotch.csv'1)'N'4
⍝ dates
d←{⍨̃/(∧/∊∘(⎕D,'-'))¨ω:⎕SIGNAL 11 ⋄ ↑'-'(⍎¨≠⊆⊢)¨ω}ds

⍝ group by month and year, summary and relative change
t←d[;1 2],∘(+/)⌸n
s←{ω[⍋ω;]}¨{t[;ω],∘(⌊⌿,(+⌿÷≢),⌈⌿,+⌿)⌸t[;3]}¨⍳2
c←{¯2-/t[;ω]+/⍤⊢⌸t[;3]}¨⍳¨⍳2
```

```
# rea
df =
# dat
d = d

# gro
a = d
a.ind
def A
    (
    r
m = A
y = A
y = y
tm =
cm =
```

Google trends (last 5 years)

The code

```
⍝ read data
(ds n)←⎕CSV⍠'Invert'2⊢(3↓⊃⎕NGET'google-scotch.csv'1)'N'4
⍝ dates
d←{⍲̃/(∧/∊∘(⎕D,'-'))¨⍵:⎕SIGNAL 11 ⋄ ↑'-'(⍎¨≠⊆⊢)¨⍵}ds

⍝ group by month and year, summary and relative change
t←d[;1 2],∘(+/)⌸n
s←{⍵[⍋⍵;]}¨{t[;⍵],∘(⌊⌿,(+⌿÷≢),⌈⌿,+⌿)⌸t[;3]}¨⍳2
c←{¯2-/t[;⍵]+/⍤⊢⌸t[;3]}¨⍳¨⍳2
```

```python
# read data
df = pd.read_csv("google-scotch.csv", header=1)
# dates
d = df.assign(Week = pd.to_datetime(df['Week'])).set_index('Week')

# group by month and year, summary and relative change
a = d.groupby([d.index.year, d.index.month]).agg(Total=(d.columns[0], 'sum'))
a.index.names = ['Year', 'Month']
def A(df, by):
    (m, a, M, t) = [('Total',x) for x in ['min', 'mean', 'max', 'sum']]
    return df.reset_index().groupby(by).agg(min=m, mean=a, max=M, total=t)
m = A(a, 'Month')
y = A(a, 'Year')
y = y.assign(change=y['total'].diff())
tm = a.unstack(level=0)
cm = a.diff().unstack(level=0)
```

The answer

What about data science in APL?

Can you do data science in APL?

How do you do data science in APL?

data science in APL?

data science in APL?

```
science⌸data
```

data science in APL?

`science⌸data`

and more ...

data science in APL?

```
science⌸data
```

and more ...

User defined functions

data science in APL?

```
science⌸data
```

and more ...

Namespaces

data science in APL?

```
science⌸data
```

and more ...

Classes

data science in APL?

`science⌸data`

and more ...

Packages

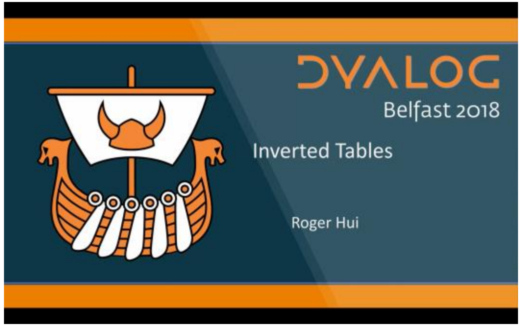
Beyond ⌸

Inverted tables by Roger Hui

```apl
⎕ct       ← 0
invert    ← {↑¨ ⊂⍤¯1 ⍉ ⍵}
assert    ← {⍺←'assertion failure' ⋄ 0∊⍵:⍺ ⎕signal 8 ⋄ shy←0}

tassert  ← {
  assert (1≤≢⍵)∧1=⍴⍴⍵ :   ⍝ non-empty vector
  assert (⊃=⊢)≢¨⍵ :       ⍝ equal tally in each item
  assert 2=≡⍵ :           ⍝ nested array with simple items
  1
}

tindex    ← {(⊂⊂⍺)⌷¨⍵}
tix       ← 8⌶
teps      ← {(≢⊃⍵) > ⍵ tix ⍺}
twithout  ← {⍺ ⌿¨⍨ ⊂~⍺ teps ⍵}
tunique   ← {⍵ ⌿¨⍨ ⊂(⍳≢⊃⍵)=tix⍨ ⍵}
tkey      ← {(⊂tix⍨⍺) ⍺⍺⌸¨ ⍵}
tgr       ← {⊃ {⍵[⍋(⊂⍵)⌷⍺]}/ ⍵,⊂⍳≢⊃⍵}
torder    ← {0=⎕nc '⍺':tgr ⍵ ⋄ (tgr ⍵)tindex ⍺}
```



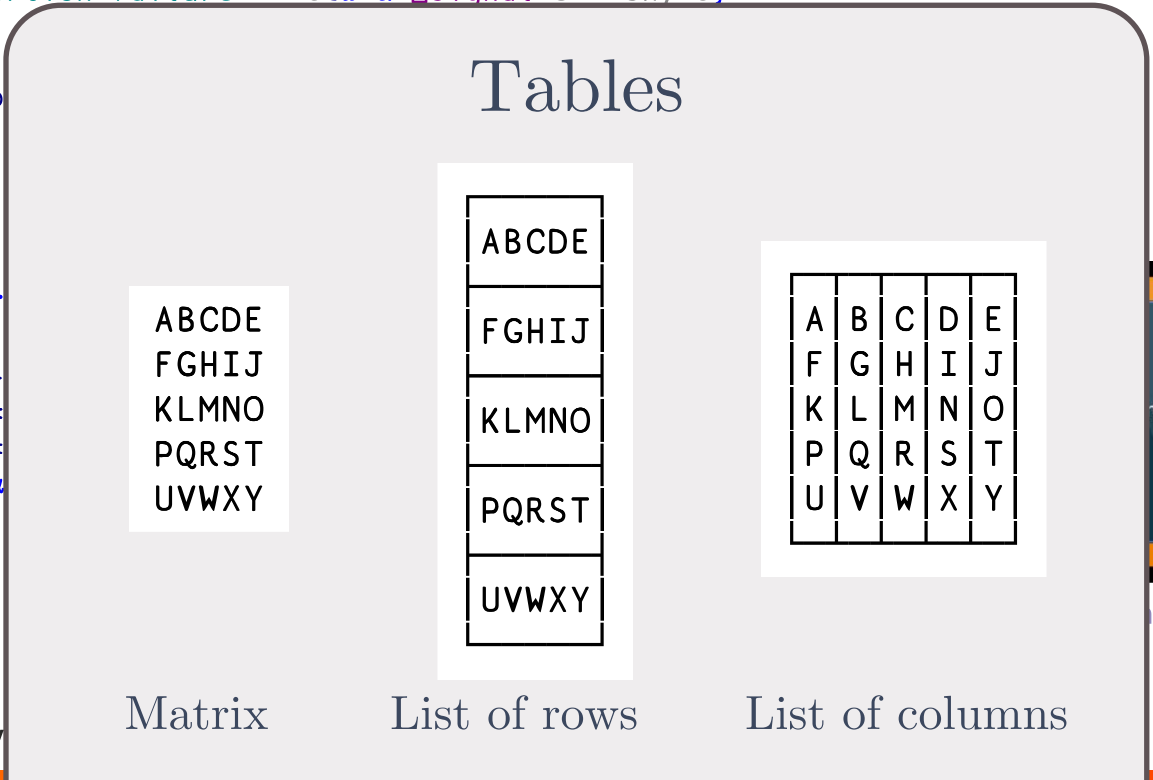
https://www.youtube.com/watch?v=IOWDkqKbMwk

Inverted tables by Roger Hui

```
⎕ct      ← 0
invert   ← {↑¨ ⊂⍤¯1 ⍉ ⍵}
assert   ← {⍺←'assertion failure' ⋄ 0∊⍵:⍺ ⎕signal 8 ⋄ shy←0}

tassert  ← {
   assert (1≤≢⍵)∧1=⍴
   assert (⊃=⊢)≢¨⍵ :
   assert 2=≡⍵ :
   1
}

tindex   ← {(⊂⊂⍺)⌷¨
tix      ← 8⌶
teps     ← {(≢⊃⍵) >
twithout ← {⍺ /¨⍨ ⊂
tunique  ← {⍵ /¨⍨ ⊂
tkey     ← {(⊂tix⍨⍺
tgr      ← {⊃ {⍵[⍋(
torder   ← {0=⎕nc '
```



?v=IOWDkqKbMwk

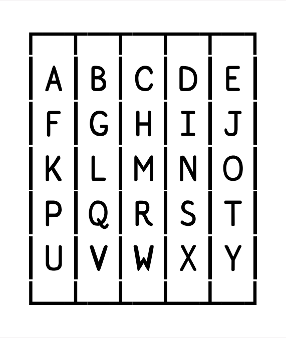
Inverted tables by Roger Hui

```
⎕ct       ← 0
invert    ← {↑¨ ⊂ö¯1 ⌽ ω}
assert    ← {α←'assertion failure' ⋄ 0∊ω:α ⎕signal 8 ⋄ shy←0}

tassert   ← {
  assert (1≤≢ω)∧1=ρ
  assert (⊃=⊢)≢¨ω :
  assert 2=≡ω :
  1
}

tindex    ← {(⊂⊂α)⌷¨
tix       ← 8⌶
teps      ← {(≢⊃ω) >
twithout  ← {α ⍱¨⍨ ⊂
tunique   ← {ω ⍱¨⍨ ⊂
tkey      ← {(⊂tix⍨α
tgr       ← {⊃ {ω[⍋(
torder    ← {0=⎕nc '
```

Tables



List of columns

?v=IOWDkqKbMwk

Inverted tables by Roger Hui

```
⎕ct       ← 0
invert    ← {↑¨ ⊂⍤¯1 ⍉ ⍵}
assert    ← {⍺←'assertion failure' ⋄ 0∊⍵:⍺ ⎕signal 8 ⋄ shy←0}

tassert   ← {
  assert (1≤≢⍵)∧1=⍴
  assert (⊃=⊢)≢¨⍵ :
  assert 2=≡⍵ :
  1
}

tindex    ← {(⊂⊂⍺)⌷¨
tix       ← 8⌶
teps      ← {(≢⊃⍵) >
twithout  ← {⍺ ≠⍤¨ ⊂
tunique   ← {⍵ ≠⍤¨ ⊂
tkey      ← {(⊂tix⍨⍺
tgr       ← {⊃ {⍵[⍋(
torder    ← {0=⎕nc '
```

Inverted table

AFKPU	BGLQV	CHMRW	DINSX	EJOTY

List of columns

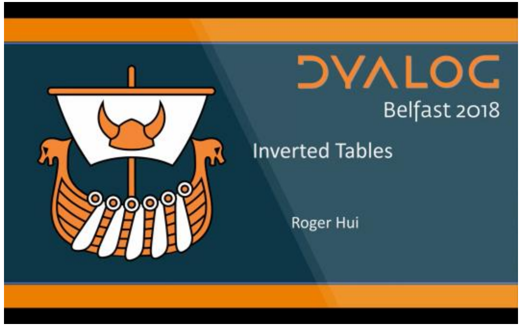
?v=IOWDkqKbMwk

Inverted tables by Roger Hui

```
⎕ct      ← 0
invert   ← {↑¨ ⊂⍤¯1 ⍉ ⍵}
assert   ← {⍺←'assertion failure' ⋄ 0∊⍵:⍺ ⎕signal 8 ⋄ shy←0}

tassert  ← {
  assert (1≤≢⍵)∧1=⍴⍴⍵ :  ⍝ non-empty vector
  assert (⊃=⊢)≢¨⍵ :      ⍝ equal tally in each item
  assert 2=≡⍵ :          ⍝ nested array with simple items
  1
}

tindex   ← {(⊂⊂⍺)⌷¨⍵}
tix      ← 8⌶
teps     ← {(≢⊃⍵) > ⍵ tix ⍺}
twithout ← {⍺ ⌿¨⍨ ⊂~⍺ teps ⍵}
tunique  ← {⍵ ⌿¨⍨ ⊂(⍳≢⊃⍵)=tix⍨ ⍵}
tkey     ← {(⊂tix⍨⍺) ⍺⍺⌸¨ ⍵}
tgr      ← {⊃ {⍵[⍋(⊂⍵)⌷⍺]}/ ⍵,⊂⍳≢⊃⍵}
torder   ← {0=⎕nc '⍺':tgr ⍵ ⋄ (tgr ⍵)tindex ⍺}
```



https://www.youtube.com/watch?v=IOWDkqKbMwk

(Dynamic) Namespaces

New array notation and system functions to safely set variables will allow to easily create ad-hoc namespaces

```
⍝ labels and columns arrays
l←'one' 'two' 'three' ⋄ c←(⍳5)(2×⍳5)(10×⍳5)

⍝ dataframe namespace
df←(labels:l)⎕VSET(↑l)c
df.labels ⋄ df.one ⋄ df.(two+three)
```

(Dynamic) Namespaces

New array notation and system functions to safely set variables will allow to easily create ad-hoc namespaces

```
⍝ labels and columns arrays
l←'one' 'two' 'three' ⋄ c←(⍳5)(2×⍳5)(10×⍳5)

⍝ dataframe namespace
df←(labels:l)⎕VSET(↑l)c
df.labels ⋄ df.one ⋄ df.(two+three)
```

(Dynamic) Namespaces

New array notation and system functions to safely set variables will allow to easily create ad-hoc namespaces

```
⍝ labels and columns arrays
l←'one' 'two' 'three' ⋄ c←(⍳5)(2×⍳5)(10×⍳5)

⍝ dataframe namespace
df←(labels:l)⎕VSET(↑l)c
df.labels ⋄ df.one ⋄ df.(two+three)
```

Object oriented programming:
data namespace

PROOF OF CONCEPT

Dyalog for data science https://github.com/yiyus/data-science-in-APL/

PROOF
OF
CONCEPT

```
⍝ load data file
f←data.frame'berkeley.csv'
⍝ group
a←data.('Applicants' 'Accepted'{(≢⍵),('A'+.=⊃¨)⍵}by'Major' 'Gender'⊢)f[]~f[⊂'Year’]
⍝ totals by gender and major
g←a data.(⍋sort⊣,join⊣(⊂'Gender')(+⌿,'T'⍨)by(⊂'Major')⊢)a[]~a[⊂'Gender']
m←g data.(⍋sort⊣,join⊣(⊂'Major')(+⌿,(⊂'Total')⍨)by(⊂'Gender')⊢)g[]~g[⊂'Major’]
⍝ accepted and applicants ratios
r←m data.(frame⊣,'%Accepted'series⊢)100×÷/m[;'Accepted' 'Applicants']
r data.(frame⊣,'%Applicants'series⊢)(100×⊢÷≢⍴('T'=⊃¨r[;⊂'Major'])⌿⊢)⊢r[;⊂'Applicants']
```

PROOF OF CONCEPT

```
⍝ load data file
f←data.frame'berkeley.csv'
⍝ group
a←data.('Applicants' 'Accepted'{(≢⍵),('A'+.=⊃¨)⍵}by'Major' 'Gender'⊢)f[]~f[⊂'Year']
⍝ totals by gender and major
g←a data.(⍋sort⊣,join⊣(⊂'Gender')(+⌿,'T'⍨)by(⊂'Major')⊢)a[]~a[⊂'Gender']
m←g data.(⍋sort⊣,join⊣(⊂'Major')(+⌿,(⊂'Total')⍨)by(⊂'Gender')⊢)g[]~g[⊂'Major']
⍝ accepted and applicants ratios
r←m data.(frame⊣,'%Accepted'series⊢)100×÷/m[;'Accepted' 'Applicants']
r data.(frame⊣,'%Applicants'series⊢)(100×⊢÷≢⍴('T'=⊃¨r[;⊂'Major'])⌿⊢)⊢r[;⊂'Applicants']
```

Year	Major	Gender	Admission
1973	C	F	Rejected
	B	M	Accepted
	Other	F	
		M	
			Rejected
	F	F	Accepted
	Other	M	
			Rejected
	A		Accepted
	Other	F	Rejected
	B	M	Accepted
	C		Rejected
	A		
	Other		
		F	
	A	M	Accepted
	Other	F	
		M	Rejected
	F		
	Other		
	C		Accepted
…12763			

Major	Gender	Applicants	Accepted
A	F	108	89
	M	1138	825
B	F	25	17
	M	560	353
C	F	593	201
	M	325	120
D	F	375	131
	M	417	138
E	F	393	94
	M	191	53
F	F	341	25
	M	373	22
Other	F	2486	937
	M	5438	2227

Major	Gender	Applicants	Accepted
A	F	108	89
	M	1138	825
	T	1246	914
B	F	25	17
	M	560	353
	T	585	370
C	F	593	201
	M	325	120
	T	918	321
D	F	375	131
	M	417	138
	T	792	269
E	F	393	94
	M	191	53
	T	584	147
F	F	341	25
	M	373	22
	T	714	47
Other	F	2486	937
	M	5438	2227
	T	7924	3164
Total	F	4321	1494
	M	8442	3738
	T	12763	5232

Major	Gender	Applicants	Accepted	%Accepted	%Applicants
A	F	108	89	82.4	2.5
	M	1138	825	72.5	13.5
	T	1246	914	73.4	9.76
B	F	25	17	68	0.579
	M	560	353	63	6.63
	T	585	370	63.2	4.58
C	F	593	201	33.9	13.7
	M	325	120	36.9	3.85
	T	918	321	35	7.19
D	F	375	131	34.9	8.68
	M	417	138	33.1	4.94
	T	792	269	34	6.21
E	F	393	94	23.9	9.1
	M	191	53	27.7	2.26
	T	584	147	25.2	4.58
F	F	341	25	7.33	7.89
	M	373	22	5.9	4.42
	T	714	47	6.58	5.59
Other	F	2486	937	37.7	57.5
	M	5438	2227	41	64.4
	T	7924	3164	39.9	62.1
Total	F	4321	1494	34.6	100
	M	8442	3738	44.3	
	T	12763	5232	41	

PROOF OF CONCEPT

```
⍝ load data file
f←data.frame'berkeley.csv'
⍝ group
a←data.('Applicants' 'Accepted'{(≢⍵),('A'+.=⊃¨)⍵}by'Major' 'Gender'⊢)f[]~f[⊂'Year’]
⍝ totals by gender and major
g←a data.(⍋sort⊣,join⊣(⊂'Gender')(+⌿,'T'⍨)by(⊂'Major')⊢)a[]~a[⊂'Gender']
m←g data.(⍋sort⊣,join⊣(⊂'Major')(+⌿,(⊂'Total')⍨)by(⊂'Gender')⊢)g[]~g[⊂'Major’]
⍝ accepted and applicants ratios
r←m data.(frame⊣,'%Accepted'series⊢)100×÷/m[;'Accepted' 'Applicants']
r data.(frame⊣,'%Applicants'series⊢)(100×⊢÷≢⍴('T'=⊃¨r[;⊂'Major'])⌿⊢)⊢r[;⊂'Applicants']
```

```
# read data
df = pd.read_csv("berkeley.csv")
# group by gender and by major
adm = ('Admission', lambda c:(c=='Accepted').sum())
app = ('Admission', 'count')
a = df.groupby(['Major','Gender']).agg(Admitted=adm, Applicants=app)
# totals by gender and by major
gg = a.reset_index().groupby('Gender').sum()
gt = pd.concat([gg], keys=['Total'], names=['Major'])
g = pd.concat([a, gt])
mg = g.reset_index().groupby('Major').sum()
mt = pd.concat([mg], keys=['T'], names=['Gender']).reorder_levels([1, 0])
m = pd.concat([g, mt]).sort_index()
# admission and applicants ratios
ar = m.assign(PctAdmitted=100*m.Admitted/m.Applicants)
```

Object oriented programming:
data namespace

PROOF OF CONCEPT

```
⍝ load data file
f←data.frame'berkeley.csv'
⍝ group
a←data.('Applicants' 'Accepted'{(≢⍵),('A'+.=⊃¨)⍵}by'Major' 'Gender'⊢)f[]~f[⊂'Year']
⍝ totals by gender and major
g←a data.(⍋sort⊣,join⊣(⊂'Gender')(+⌿,'T'⍨)by(⊂'Major')⊢)a[]~a[⊂'Gender']
m←g data.(⍋sort⊣,join⊣(⊂'Major')(+⌿,(⊂'Total')⍨)by(⊂'Gender')⊢)g[]~g[⊂'Major']
⍝ accepted and applicants ratios
r←m data.(frame⊣,'%Accepted'series⊢)100×÷/m[;'Accepted' 'Applicants']
r data.(frame⊣,'%Applicants'series⊢)(100×⊢÷≢⍴('T'=⊃¨r[;⊂'Major'])⌿⊢)⊢r[;⊂'Applicants']
```

Berkeley

Object oriented programming:
data namespace

PROOF OF CONCEPT

data.Series class

An instance of the data.Series class contains a labelled array. Bracket indexing of the series gives access to the values of the array. The **values** property is equivalent to []. The label, which can take any value, can be accessed through the **label** property.

data.Frame class

An instance of the data.Frame class contains a list of data.Series instances. All the series must contain values arrays of the same length.
The series list can be accessed by bracket indexing of rank 1 using the labels of the series as indices. Bracket indexing of rank 2 gives access to the values in the series. The properties **series**, **labels** and **values** are equivalent to [], [].label and [;].
Frames are displayed with shades at row intervals of the size specified by the **SHADE** property and up to a maximum number of lines specified by the **MAXLINES** property.

data.sort operator

This operator sorts data according to the left function.
•α (αα data.sort) ω returns ω (a frame, list of series, or array) sorted according to the result of αα α (where αα typically is one of ⍋⍒).
•(αα data.sort) ω is equivalent to (αα data.sort)⍨ω.

data.by operator

This operator groups data by the right operand and applies the left function.
•α (αα data.by ωω) ω returns the data in ω (a frame or list of series) grouped according to ωω (also a frame or list of series) and apply αα to each group.
If αα and ωω are both series or labels, the values in αα are grouped for each value in ωω and distributed in series. Labels (either all of them, the ones not in ω, or the ones not in ωω) are given in α, which can be a list of values, a list of series, or a frame.
•(αα data.by ωω) ω is equivalent to θ (αα data.by ωω) ω. If αα and ωω are both series or labels, the values in αα are grouped for each value in ωω and distributed in series.

data.join operator

This operator merges two frames (or lists of series).
•α (αα data.join ωω) ω returns frame with series labelled α.labels ωω ω.labels.
If two series at left and right have the same label, its values are combined as α.values αα ω.values.
•(αα data.join ωω) ω returns a series with label ωω ω.labels and values αα ω.values.

data.series function

This function returns an instance or a list of instances of the data.Series class.
•α data.series ω creates an instance of data.Series with label α and values ω.
If α is a series, the label is taken from it.
•data.series ω creates an instance of data.Series for each of the series in ω and each of the series contained in each frame in ω. If ω is a rank 2 array, it must contain series with the same label in each column, and their values will be concatenated.

data.frame function

This function returns an instance of the data.Frame class.
•α data.frame ω creates an instance of data.Frame with labels α (or the labels of the series list or frame α) and values ω. If ω is a string, it writes the frame α to the CSV file ω or reads the CSV file ω without header and returns frame with labels α.
•data.frame ω creates an instance of data.Frame with each of the series returned by data.series ω. If ω is a string, it reads the file ω as CSV with header and returns frame.

```
⍝ load data file
f←data.frame'berkeley.csv'
⍝ group
a←data.('Applicants' 'Accepted'{(≢⍵),('A'+.=⊃¨)⍵}by'Major' 'Gender'⊢)f[]~f[⊂'Year']
⍝ totals by gender and major
g←a data.(⍋sort⊣,join⊣(⊂'Gender')(+⌿,'T'⍨)by(⊂'Major')⊢)a[]~a[⊂'Gender']
m←g data.(⍋sort⊣,join⊣(⊂'Major')(+⌿,(⊂'Total')⍨)by(⊂'Gender')⊢)g[]~g[⊂'Major']
⍝ accepted and applicants ratios
r←m data.(frame⊣,'%Accepted'series⊢)100×÷/m[;'Accepted' 'Applicants']
r data.(frame⊣,'%Applicants'series⊢)(100×⊢÷≢⍴('T'=⊃¨r[;⊂'Major'])⌿⊢)⊢r[;⊂'Applicants']
```

Berkeley

```
AVG←+⌿÷≢ ⋄ STD←(2*∘÷⍨+⌿÷¯1+≢)2*⍨⊢-⍤1+⌿÷≢      ⍝ average and standard deviation
PCT←{((2÷⍨+/)⊢⌷⍨∘⊂⍋⌷⍨∘⊂∘⌈100÷⍨⍺×0 1+≢)⍵}     ⍝ percentile-⍺
PCC←+.×⍤((⊢÷2*∘÷⍨+.×⍨)⊢-+⌿÷≢)                ⍝ Pearson correlation coefficient
f←'sl' 'sw' 'pl' 'pw' 'class'data.frame'iris.csv'
aggs←{l f←⍺⍵ ⋄ ⍪,¨⍺⍺{(⍺⍺ ⍵)⍵⍵data.by(f[l])⊢f[⊂⍵]}⍵⍵¨f.labels~l}
s←(⊂'class'){'⌊AS⌈',¨⊂⍵}aggs(⌊⌿,AVG,STD,⌈⌿)f
p←(⊂'class')('25' '50' '75',⍨¨⊂)aggs(,25 50 75∘.PCT↓∘⍉)f
s←s{data.frame⍺.series,1↓⍵.series}¨¨p ⋄ s.SHADE←0
pcc←{∘.PCC⍨↓⍉⍵}data.by(⊂'class')⊢f[]
pcc.values⍪←(⊂'Class'),({(∪⍵)⍳⍵}f[;⊂'class'])∘PCC¨f[f.labels~⊂'class'].values
⍕(⍪⊃(⊣,(⊂''),⊢)⌿s)pcc
```

Iris

```
g←'date' 'n'data.frame⎕CSV⍠'Invert'2⊢(3↓⊃⎕NGET'google-scotch.csv'1)'N'4
d←{⍲/(∧/∊∘(⎕D,'-'))¨⍵:⎕SIGNAL 11 ⋄ ↑'-'(⍎¨≠⊆⊢)¨⍵}g[;⊂'date']
tt←'n'data.by'month'⊢t←'year' 'month' 'n'+⌿data.by(d[;1 2])⊢g[⊂'n']
tv←(tt[;,⊂'year']⍪⊂'total'),(⊢⍪(+⌿+/¨))(⊢,(+/+/¨))tt[;1↓tt.labels]
tt←(tt.labels,⊂'total')data.frame tv
c←'n'data.by'month'⊢t data.frame t[;t.labels~'n'],(⊂θ),¯2-/t[;'n']
tt←tt data.frame tt[;]⍪(⊂'')⍪c[;],(⊂θ)⍪¯2-⌿¯1↓tt[;,⊂'total']
tt ⋄ θ ⋄ {'min' 'avg' 'max' 'total'(⌊⌿,(+⌿÷≢),⌈⌿,+⌿)data.by(t[⊂⍵])⊢t[⊂'n']}¨'year' 'month'
```

Google

Dyalog for data science https://github.com/yiyus/data-science-in-APL/

DYALOG

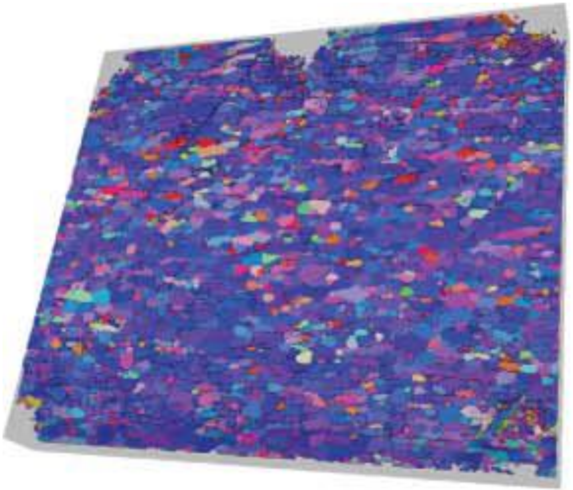
Application

Microstructural analysis of metals
Orientation distribution functions

Galan Lopez & Kestens (2021). J. Appl. Cryst. 54, 148-162
https://doi.org/10.1107/S1600576720014909

3D EBSD measurement:

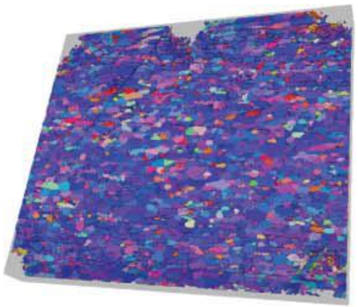
- Low-carbon steel sample
- 9047108 data points
- 750×580×85 um
- Crystallographic orientation of each point (indicated by color)



Microstructural analysis of metals

Orientation distribution functions

Galan Lopez & Kestens (2021). J. Appl. Cryst. 54, 148-162
https://doi.org/10.1107/S1600576720014909



Orientation distribution function:

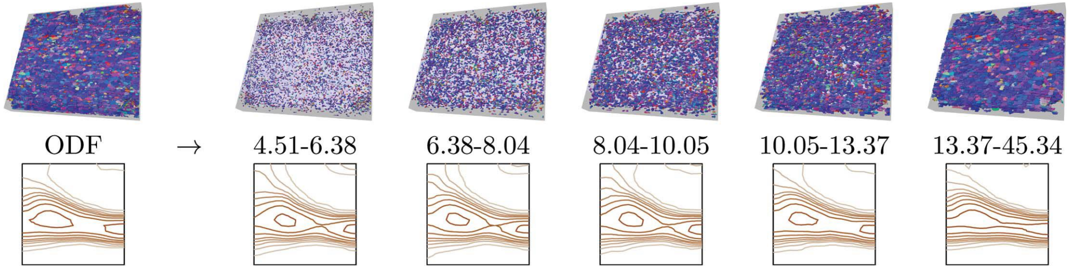
- Represents preferential orientations in the sample
- Continuous function in orientation (Euler) space
- Generalised spherical harmonics

Microstructural analysis of metals
Orientation distribution functions

Galan Lopez & Kestens (2021). J. Appl. Cryst. 54, 148-162
https://doi.org/10.1107/S1600576720014909

Grain size dependency:

- Group grains by size
- Calculate ODF for each size group



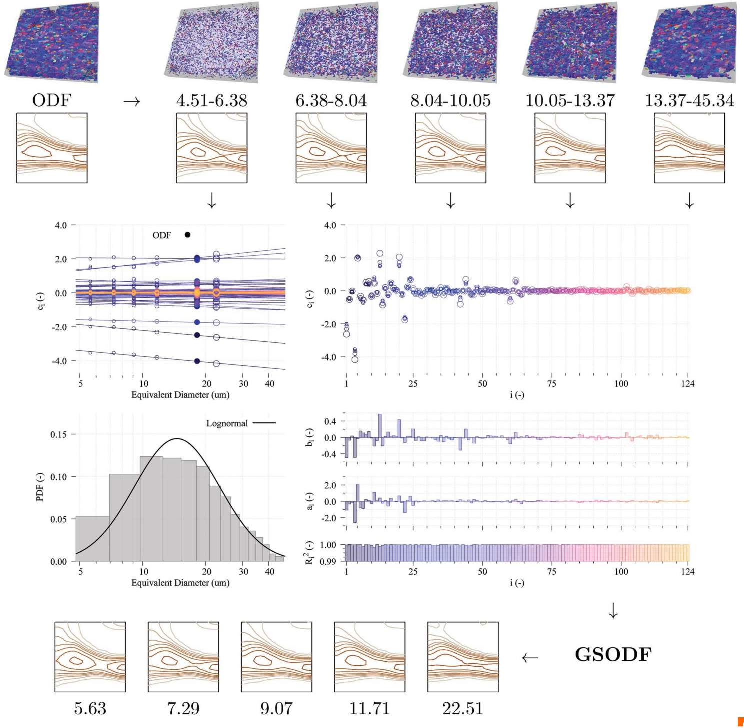
* Grain: region of material with the same crystallographic orientation (same color)

Microstructural analysis of metals
Orientation distribution functions

Galan Lopez & Kestens (2021). J. Appl. Cryst. 54, 148-162
https://doi.org/10.1107/S1600576720014909

Grain size dependent orientation distribution function (**GSODF**):

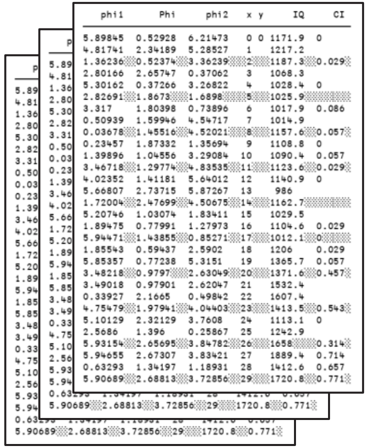
- Continuous function which combines grain size and orientation distributions



Microstructural analysis of metals
Orientation distribution functions

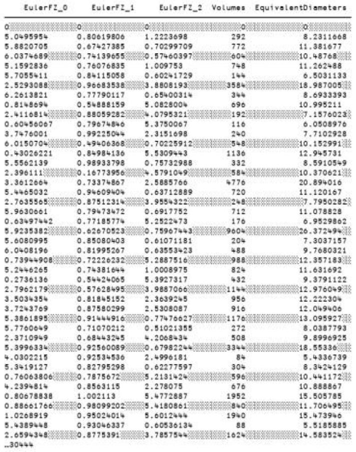
Galan Lopez & Kestens (2021). J. Appl. Cryst. 54, 148-162
https://doi.org/10.1107/S1600576720014909

EBSD files



phi1	Phi	phi2	x	y	IQ	CI
5.89845	0.52928	6.21473	0	0	1171.9	0
4.81741	2.34189	5.28527	1		1217.2	
1.36236	0.52374	3.36239	2		1187.3	0.029
2.80166	2.65747	0.37062	3		1068.3	
5.30162	0.37266	3.26822	4		1028.4	0
2.82691	1.8673	1.6898	5		1025.9	
3.317	1.80398	0.73896	6		1017.9	0.086
0.50939	1.59946	4.54717	7		1014.9	
0.03678	1.45516	4.52021	8		1157.6	0.057
0.23457	1.87332	1.35694	9		1108.8	0
1.39896	1.04556	3.29084	10		1090.4	0.057
3.46718	1.29774	4.83535	11		1123.6	0.029
4.02352	1.41181	5.64012	12		1140.9	0
5.66807	2.73715	5.87267	13		986	
1.72004	2.47699	4.50675	14		1162.7	
5.20746	1.03074	1.83411	15		1029.5	
1.89475	0.77991	1.27973	16		1104.6	0.029
5.94471	1.43855	0.85271	17		1012.1	0
1.85543	0.59437	2.5902	18		1206	0.029
5.85357	0.77238	5.3151	19		1365.7	0.057
3.48218	0.9797	2.63049	20		1371.6	0.457
3.49018	0.97901	2.62047	21		1532.4	
0.33927	2.1665	0.49842	22		1607.4	
4.75479	1.97941	4.04403	23		1413.5	0.543
5.10129	2.32129	3.7608	24		1113.1	0
2.5686	1.396	0.25867	25		1242.9	
5.93154	2.65695	3.84782	26		1658	0.314
5.94655	2.67307	3.83421	27		1889.4	0.714
0.63293	1.34197	1.18931	28		1412.6	0.657
5.90689	2.68813	3.72856	29		1720.8	0.771

List of grains



EulerFZ_0	EulerFZ_1	EulerFZ_2	Volumes	EquivalentDiameters
0	0	0	0	0
5.0495954	0.80619806	1.2223698	292	8.2311668
5.8820705	0.67427385	0.70299709	772	11.381677
6.0374689	0.74139655	0.57460397	604	10.48768
5.1592836	0.76076835	1.009753	748	11.262488
5.7055411	0.84115058	0.60241729	144	6.5031133
2.5293088	0.96683538	3.8808193	3584	18.987005
6.2613821	0.77790117	0.65400314	344	8.6933393
0.8148694	0.54888159	5.0828004	696	10.995211
2.4116814	0.88059282	4.0795321	192	7.1576023
0.60456067	0.79674846	5.3750067	116	6.0508976
3.7476001	0.99225044	2.3151698	240	7.7102928
6.0150704	0.49406368	0.70225912	548	10.152991
0.43026221	0.84984136	5.5309443	1136	12.945731
5.5562139	0.98933798	0.75732988	332	8.5910549
2.396111	0.16773956	4.5791049	584	10.370621
3.3612664	0.73374867	2.5885766	4776	20.894016
5.4465032	0.94609404	0.63712889	720	11.120167
2.7635565	0.87512314	3.9554322	248	7.7950282
5.9630661	0.79473472	0.6917752	712	11.078828
0.63497442	0.77185774	5.2522473	176	6.9529862
5.9235382	0.62670523	0.75967443	9604	26.372494
5.6080995	0.85080403	0.61071181	204	7.3037157
6.0408196	0.81995267	0.63553423	488	9.7680321
0.73944908	0.72226232	5.2887516	988	12.357183
5.2446265	0.74381644	1.0008975	824	11.631692
0.2736136	0.54424065	5.3927317	432	9.3791122
2.7962179	0.57628495	3.9887066	1144	12.976049
3.5034354	0.81845152	2.3639245	956	12.222304
3.7243769	0.87580299	2.5308087	916	12.049406
5.3861895	0.91444916	0.77476627	1176	13.095927
5.7760649	0.71070212	0.51021355	272	8.0387793
2.3710949	0.68443245	4.2068434	508	9.8996925
5.3996334	0.92560089	0.67982244	3344	18.55336
4.0302215	0.92534536	2.4996181	84	5.4336739
5.3419127	0.82795298	0.62277597	304	8.3424129
0.76063806	0.7875672	5.2131424	596	10.441172
4.2394814	0.8563115	2.278075	676	10.888867
0.80678838	1.002113	5.4772887	1952	15.505785
0.88661766	0.98099202	5.4180861	840	11.706495
1.0268919	0.95024014	5.6012444	1940	15.473946
5.4389448	0.93046337	0.6053613	88	5.5185885
2.6594348	0.8775391	3.7857544	1624	14.583524

…30444

ODF calculation

GSODF

```
⍝ Hofsommer and Potters. Series A: Mathematical Sciences 63.5 (1960): 460-480
I←2∘|+2×∘⍳1+⌊⍤÷∘2
dn←{
    Nnm←{(n m)←⍺⍵ ⋄ n<m:0 ⋄ (-÷2)*⍨(n+÷2)×n(-÷⍤!+)m}  ⍝ (2.11)
    pj←{(4*-⍵)×⍵!2×⍵}  ⍝ (2.8)
    d0nk←{(n k)←⌊⍺ ⍵÷2 ⋄ (1+⍵>0)×(n(-×⍤pj(2|⍵)++)k)×(⍺+÷2)*÷2}  ⍝ (3.4)
    gnm←{(n m)←⍺⍵ ⋄ (m≤n)×(¯1*m)×((n+m)*÷2)×(1+n-m)*÷2}  ⍝ (3.1)
    EQ←{(m(n k))←⍺⍵ ⋄ (0⍴⍨m-1),(n-m-2)↑(n gnm m)(2×k)(n gnm m+1)}  ⍝ (3.3)
    dnk←{(n k)←⍺⍵ ⋄ ((-n+1)↑n d0nk k)⌹(1↑⍨n+1)⍪⍨↑(1+⍳n)EQ¨⊂n k}  ⍝ solve (3.3)
    n←1+⍳⍵ ⋄ ((⍵+1)↑0 d0nk 0)⍪↑,¨/↑n∘.dnk¨I¨n
}
_dnmk←{(nm k)←⍺⍵ ⋄ (⌊k÷2)⌷∊nm⌷⍺⍺}

⍝ Bunge. Kristall und Technik 9.8 (1974): 939-963
_almns←{(l(m n s))←⍺⍵ ⋄ m s×⍤(l∘⍺⍺)n s}  ⍝ (6)
Nm←{2÷⍨(⊢×1+⊢)⍵} ⋄ Nl←{+/Nm 1+⍳1+⍵} ⋄ lmn←{(l m n)←⍺,(⌈/,⌊/)⍵ ⋄ n+(Nm m)+Nl l-1}
_Q←{
    _alms←{(l(m s))←⍺⍵ ⋄ (¯1*m)×l m⍺⍺s}
    _Qlms←{(l(m s))←⍺⍵ ⋄ o←2|l+s ⋄ o∧2|l+m:l(⍺⍺_Qlms1)m s ⋄ l(⍺⍺_Qlms0)o⌽m s}  ⍝ (7-8)
    _Pl0s←{(l s)←⍺⍵ ⋄ +/⍺⍺{(l⍺⍺s⍵)×¯1*⌊⍵÷2}¨I l}  ⍝ (19-20)
    _Qlms0←{(l(m s))←⍺⍵ ⋄ (÷1+s>0)×(¯1*(⌊m÷2)-l×s>0)×(l⍺⍺m s)÷l(⍺⍺_Pl0s)s}  ⍝ (21-23)
    Mlm←{(l m)←⍺⍵ ⋄ -(0j1*m)×m×(÷2)*⍨8÷l×(1+l)×1+2×l}  ⍝ (31)
    Ams←{(m s)←⍺ ⋄ ⍵÷(0j1*2|m)×1+s>0}  ⍝ (14-16)
    _Pl1s←{(l s)←⍺⍵ ⋄ +/⍺⍺{(1+⍵>0)×(0j1*2|l)×(l⍺⍺1 s⍵)×¯1*⌊⍵÷2}¨I l}  ⍝ (25-26)
    ilmn←{i/⍨{(⍵[0]∘≥∧⍵[2]∘≤)⍵[1]}¨i←,⍳3⍴1+⍵}
    O←{2|+/(↑⍵)[;0 ⍺]} ⋄ A←⍺⍺_alms ⋄ (om os)←1 2 O¨⊂q←ilmn⍵ ⋄ Aq←{q[⍺lmn⍵]}_almns
    Qlms0←A _Qlms0 ⋄ Qlms1←{(l(m s))←⍺⍵ ⋄ (a⊃⍨(l-1),⌊m s÷2)÷(0j1*1+s)×l(Aq _Pl1s)s}  ⍝ (21-24)
    a←{l←⍵ ⋄ {m←⍵ ⋄ ⌽+\⌽{(l Mlm m)×(⊢Ams l A⊢)m,1+⍵}¨I l-1}¨I l-1}¨1+⍳⍵  ⍝ (32-34)
    _←{(l m s)←⍵ ⋄ q[l lmn m s]←l Qlms0(2|l+s)⌽m s}¨(os⍲om)/q
    _←{(l m s)←⍵ ⋄ q[l lmn m s]←l Qlms1 m s}¨(os∧om)/q
    q
}
_Qlmn←{(l(m n))←⍺⍵ ⋄ m<n:l∇n m ⋄ n<0:(¯1*l+m)×l∇m(|n) ⋄ ⍺⍺[⍺lmn⍵]}
_Plmn_←{(((l m n)p)←⍵⍵ ⍵ ⋄ +/⍺⍺{(l⍺⍺m n⍵)×*0j1×⍵×p}¨(-l)+⍳1+2×l}  ⍝ (2)
_Tlmn_←{(((l m n)(p1 p p2))←⍵⍵ ⍵ ⋄ (⍺⍺_Plmn_ l m n⊢p)×m n×⍤*.×0j1×p2 p1}  ⍝ (1)

⍝ Esling. Thesis Metz. 1981
Ml←{⌊0.5+24÷⍨w+(6×1÷.○w 1×○÷4)+(8×1÷.○(w←1+2×⍵)1×○÷3)+9×¯1*⍵}  ⍝ (3.45)
_B←{
    _Pl4m4n←{(l(m n))←⍺⍵ ⋄ 3÷⍨((m=n)×(0=2|l)∨m>1)+(2*(÷2)×(0<m)∧0=n)×(2+2×n>0)×l⍺⍺m n}  ⍝ 3.37
    _Plm←{(l m)←⍺⍵ ⋄ 9○ROUND(l⍺⍺_Pl4m4n((4×⌊l÷4)-m),4×⊢)¨⍳⌊1+l÷4}
    S←{⍺≥≢⍵:⍵ ⋄ (f r)←1(,⍤↑,⍤⊂↓)⍵ ⋄ ⍬≡v←⍸1e¯4<f:⍺∇r ⋄ (f÷f[i]*÷2)⍪(⍺-1)∇r-r[;i]∘.×f÷f[i←⊢/v]}  ⍝ schmidt (3.47)
    _Bl←{l←⍵ ⋄ 0=n←Ml l:⍬ ⋄ (÷(○1+(1+⌊l÷4)↑1)*÷2)×[1]n S↑(l⍺⍺_Plm ×∘4)¨⍳1+⌊l÷4}  ⍝ (3.50)
    l←⍵ ⋄ ⍺⍺_Bl¨⍳1+⍵
}
_Blmn←{(l(m n))←⍺(⌊⍵÷4) ⋄ (l⊃⍺⍺)[m;n]}

⍝ Bunge. Mathematical methods
⍝_TClmn_←{(l m n)←⍺ ⋄ +/⍵∘(⍺⍺{(⍵ ⍵⍵m n)×⍺⍺ _Tlmn_ ⍵ m n⊢⍺}⍵⍵)¨1-⍨⍳1+2×l}  ⍝ (14.122)
⍝ eg l m n(Blmn _TClmn_ Almns)p1 p p2

_Slmn_←{(((l m n)(p1 p p2))←⍵⍵ ⍵ ⋄ +/(1 2∘.○m n×p2 p1)∘(⍺⍺{(¯1*2|⍵)×(l⍺⍺m n⍵)×(×/⍺[~2|⍵;])×2○⍵×p})¨⍳1+l} ⍝ (14.257)
_T21lmn_←{(((l m n)(p1 p p2))←⍺⍵ ⋄ (l⍵⍵m n)×(⍺⍺_almns _Slmn_⍺⊢⍵)×(2*÷2)×.*m n≠0}  ⍝ (14.259-258)

CH←{ear←○⍵÷180 ⋄ {0=ml←Ml(l←⍵):⍬ ⋄ {m←⍵ ⋄ {n←⍵ ⋄ l m n∘((q _Qlmn)_T21lmn_(b _Blmn))¨ear}¨4×⍳1+⌊l÷4}¨⍳ml}¨1+⍳lmax}

⍝ Van Houtte
Conv←{
    ea←(180 0 180+1 ¯1 1×⊢)¨@(0>1⊃¨⊢)(180 0 0-⍨360 0 0|180 0 0+⊢)¨⍵
    ea←360 0 90∘|¨(180 180 0+1 ¯1 ¯1×⊢)¨@(90<1⊃¨⊢)ea
    ⍺←0 ⋄ 0=⍺:ea ⋄ (q r)←↓⍉↑(0 90⊤⊃)¨ea ⋄ a←⍺-1
    (90×a)0 0∘+¨(90 90 0-1 1 ¯1×0 90 0|⊢)¨@((q≠⍤(2∘|)a)⍨)⊢r(⊣,1↓⊢)¨ea
}
Grid←{⍺←0 ⋄ 1+⌈(360-270×⍺)90 90÷⍵}
_Grid←{
    (ea t)←⍵ ⋄ ea←↓⍉↑ea ⋄ sh←⍺⍺Grid⍺ ⋄ q←1+p←(sh-2)⌊⌊⍺÷⍨ea ⋄ dq←(⍺|ea)÷⍺ ⋄ dp←1-dq
    p←⊃,/⊃(,∘.(,¨))/↓⍉↑p q ⋄ t←⊃,/t∘×¨⊃(,∘.×)/↓⍉↑dp dq ⋄ (⍺×,⍳sh)(,t+@p⊢sh⍴0)
}
```

WORK IN PROGRESS

```
⍝ Hofsommer and Potters. Series A: Mathematical Sciences 63.5 (1960): 460-480
I←2∘|+2×∘⍳1+⌊⍤÷∘2
dn←{
    Nnm←{(n m)←⍺⍵ ⋄ n<m:0 ⋄ (-÷2)*⍨(n+÷2)×n(-÷⍥!+)m}  ⍝ (2.11)
    pj←{(4*-⍵)×⍵!2×⍵}  ⍝ (2.8)
    d0nk←{(n k)←⌊⍺ ⍵÷2 ⋄ (1+⍵>0)×(n(-×⍥pj(2|⍵)++)k)×(⍺+÷2)*÷2}  ⍝ (3.4)
    gnm←{(n m)←⍺⍵ ⋄ (m≤n)×(¯1*m)×((n+m)*÷2)×(1+n-m)*÷2}  ⍝ (3.1)
    EQ←{(m(n k))←⍺⍵ ⋄ (0⍴⍨m-1),(n-m-2)↑(n gnm m)(2×k)(n gnm m+1)}  ⍝ (3.3)
    dnk←{(n k)←⍺⍵ ⋄ ((-n+1)↑n d0nk k)⌹(1↑⍨n+1)⍪⍨↑(1+⍳n)EQ¨⊂n k}  ⍝ solve (3.3)
    n←1+⍳⍵ ⋄ ((⍵+1)↑0 d0nk 0)⍪↑,¨/↑n∘.dnk¨I¨n
}
_dnmk←{(nm k)←⍺⍵ ⋄ (⌊k÷2)⌷⊃nm⌷⍺⍺}

⍝ Bunge. Kristall und Technik 9.8 (1974): 939-963
_almns←{(l(m n s))←⍺⍵ ⋄ m s×⍥(l∘⍺⍺)n s}  ⍝ (6)
Nm←{2÷⍨(⊢×1+⊢)⍵} ⋄ Nl←{+/Nm 1+⍳1+⍵} ⋄ lmn←{(l m n)←⍺,(⌈/,⌊/)⍵ ⋄ n+(Nm m)+Nl l-1}
_Q←{
    _alms←{(l(m s))←⍺⍵ ⋄ (¯1*m)×l m⍺⍺s}
    _Qlms←{(l(m s))←⍺⍵ ⋄ o←2|l+s ⋄ o∧2|l+m:l(⍺⍺_Qlms1)m s ⋄ l(⍺⍺_Qlms0)o⌽m s}  ⍝ (7-8)
    _Pl0s←{(l s)←⍺⍵ ⋄ +/⍺⍺{(l⍺⍺s⍵)×¯1*⌊⍵÷2}¨I l}  ⍝ (19-20)
    _Qlms0←{(l(m s))←⍺⍵ ⋄ (÷1+s>0)×(¯1*(⌊m÷2)-l×s>0)×(l⍺⍺m s)÷l(⍺⍺_Pl0s)s}  ⍝ (21-23)
    Mlm←{(l m)←⍺⍵ ⋄ -(0j1*m)×m×(÷2)*⍨8÷l×(1+l)×1+2×l}  ⍝ (31)
    Ams←{(m s)←⍺ ⋄ ⍵÷(0j1*2|m)×1+s>0}  ⍝ (14-16)
    _Pl1s←{(l s)←⍺⍵ ⋄ +/⍺⍺{(1+⍵>0)×(0j1*2|l)×(l⍺⍺1 s⍵)×¯1*⌊⍵÷2}¨I l}  ⍝ (25-26)
    ilmn←{i/⍨{(⍵[0]∘≥∧⍵[2]∘≤)⍵[1]}¨i←,⍳3⍴1+⍵}
    O←{2|+/(↑⍵)[;0 ⍺]} ⋄ A←⍺⍺_alms ⋄ (om os)←1 2 O¨⊂q←ilmn⍵ ⋄ Aq←{q[⍺lmn⍵]}_almns
    Qlms0←A _Qlms0 ⋄ Qlms1←{(l(m s))←⍺⍵ ⋄ (a⊃⍨(l-1),⌊m s÷2)÷(0j1*1+s)×l(Aq _Pl1s)s}  ⍝ (21-24)
    a←{l←⍵ ⋄ {m←⍵ ⋄ ⌽+\⌽{(l Mlm m)×(⊢Ams l A⊢)m,1+⍵}¨I l-1}¨I l-1}¨1+⍳⍵  ⍝ (32-34)
    _←{(l m s)←⍵ ⋄ q[l lmn m s]←l Qlms0(2|l+s)⌽m s}¨(os⍲om)/q
    _←{(l m s)←⍵ ⋄ q[l lmn m s]←l Qlms1 m s}¨(os∧om)/q
    q
}
_Qlmn←{(l(m n))←⍺⍵ ⋄ m<n:l∇n m ⋄ n<0:(¯1*l+m)×l∇m(|n) ⋄ ⍺⍺[⍺lmn⍵]}
_Plmn_←{((l m n)p)←⍵⍵ ⍵ ⋄ +/⍺⍺{(l⍺⍺m n⍵)×*0j1×⍵×p}¨(-l)+⍳1+2×l}  ⍝ (2)
_Tlmn_←{((l m n)(p1 p p2))←⍵⍵ ⍵ ⋄ (⍺⍺_Plmn_ l m n⊢p)×m n×⍥*.×0j1×p2 p1}  ⍝ (1)

⍝ Esling. Thesis Metz. 1981
Ml←{⌊0.5+24÷⍨w+(6×1÷.○w 1×○÷4)+(8×1÷.○(w←1+2×⍵)1×○÷3)+9×¯1*⍵}  ⍝ (3.45)
_B←{
    _Pl4m4n←{(l(m n))←⍺⍵ ⋄ 3÷⍨((m=n)×(0=2|l)∨m>1)+(2*(÷2)×(0<m)∧0=n)×(2+2×n>0)×l⍺⍺m n}  ⍝ 3.37
    _Plm←{(l m)←⍺⍵ ⋄ 9○ROUND(l⍺⍺_Pl4m4n((4×⌊l÷4)-m),4×⊢)¨⍳⌊1+l÷4}
    S←{⍺≥≢⍵:⍵ ⋄ (f r)←1(,⍤↑,⍥⊂↓)⍵ ⋄ θ≡v←⍸1e¯4<f:⍺∇r ⋄ (f÷f[i]*÷2)⍪(⍺-1)∇r-r[;i]∘.×f÷f[i←⊢/v]}  ⍝ schmidt (3.47)
    _Bl←{l←⍵ ⋄ 0=n←Ml l:θ ⋄ (÷(○1+(1+⌊l÷4)↑1)*÷2)×[1]n S↑(l⍺⍺_Plm ×∘4)¨⍳1+⌊l÷4}  ⍝ (3.50)
    l←⍵ ⋄ ⍺⍺_Bl¨⍳1+⍵
}
_Blmn←{(l(m n))←⍺(⌊⍵÷4) ⋄ (l⊃⍺⍺)[m;n]}

⍝ Bunge. Mathematical methods
⍝_TClmn_←{(l m n)←⍺ ⋄ +/⍵∘(⍺⍺{(⍵ ⍵⍵m n)×⍺⍺ _Tlmn_ ⍵ m n⊢⍺}⍵⍵)¨1-⍨⍳1+2×l}  ⍝ (14.122)
⍝ eg l m n(Blmn _TClmn_ Almns)p1 p p2

_Slmn_←{((l m n)(p1 p p2))←⍵⍵ ⍵ ⋄ +/(1 2∘.○m n×p2 p1)∘(⍺⍺{(¯1*2|⍵)×(l⍺⍺m n⍵)×(×/⍺[~2|⍵;])×2○⍵×p})¨⍳1+l} ⍝ (14.257)
_T21lmn_←{((l m n)(p1 p p2))←⍺⍵ ⋄ (l⍵⍵m n)×(⍺⍺_almns _Slmn_⍺⊢⍵)×(2*÷2)×.*m n≠0}  ⍝ (14.259-258)

CH←{ear←○⍵÷180 ⋄ {0=ml←Ml(l←⍵):θ ⋄ {m←⍵ ⋄ {n←⍵ ⋄ l m n∘((q _Qlmn)_T21lmn_(b _Blmn))¨ear}¨4×⍳1+⌊l÷4}¨⍳ml}¨1+⍳lmax}

⍝ Van Houtte
Conv←{
    ea←(180 0 180+1 ¯1 1×⊢)¨@(0>1⊃¨⊢)(180 0 0-⍨360 0 0|180 0 0+⊢)¨⍵
    ea←360 0 90∘|¨(180 180 0+1 ¯1 ¯1×⊢)¨@(90<1⊃¨⊢)ea
    ⍺←0 ⋄ 0=⍺:ea ⋄ (q r)←↓⍉↑(0 90⊤⊃)¨ea ⋄ a←⍺-1
    (90×a)0 0∘+¨(90 90 0-1 1 ¯1×0 90 0|⊢)¨@((q≠⍥(2∘|)a)⍨)⊢r(⊣,1↓⊢)¨ea
}
Grid←{⍺←0 ⋄ 1+⌈(360-270×⍺)90 90÷⍵}
_Grid←{
    (ea t)←⍵ ⋄ ea←↓⍉↑ea ⋄ sh←⍺⍺Grid⍺ ⋄ q←1+p←(sh-2)⌊⌊⍺÷⍨ea ⋄ dq←(⍺|ea)÷⍺ ⋄ dp←1-dq
    p←⊃,/⊃(,∘.(,¨))/↓⍉↑p q ⋄ t←⊃,/t∘×¨⊃(,∘.×)/↓⍉↑dp dq ⋄ (⍺×,⍳sh)(,t+@p⊢shρ0)
}
```

Conclusion

Good news, everyone!

If you are an APLer…

If you are an APLer...

Congratulations! You are a data scientist!

If you are an APLer...

Congratulations! You are a data scientist!

If you are a data scientist...

If you are an APLer…

Congratulations! You are a data scientist!

If you are a data scientist…

TRy APL https://tryapl.org/

DYALOG

GHENT UNIVERSITY

science⌸data

Berkeley

Iris

Google

data.Series class

An instance of the data.Series class contains a labelled array. Bracket indexing of the series gives access to the values of the array. The **values** property is equivalent to []. The label, which can take any value, can be accessed through the **label** property.

data.Frame class

An instance of the data.Frame class contains a list of data.Series instances. All the series must contain values arrays of the same length.
The series list can be accessed by bracket indexing of rank 1 using the labels of the series as indices. Bracket indexing of rank 2 gives access to the values in the series. The properties **series**, **labels** and **values** are equivalent to [], [].label and [;].
Frames are displayed with shades at row intervals of the size specified by the **SHADE** property and up to a maximum number of lines specified by the **MAXLINES** property.

data.series function

This function returns an instance or a list of instances of the data.Series class.
•⍺ data.series ⍵ creates an instance of data.Series with label ⍺ and values ⍵. If ⍺ is a series, the label is taken from it.
•data.series ⍵ creates an instance of data.Series for each of the series in ⍵ and each of the series contained in each frame in ⍵. If ⍵ is a rank 2 array, it must contain series with the same label in each column, and their values will be concatenated.

data.frame function

This function returns an instance of the data.Frame class.
•⍺ data.frame ⍵ creates an instance of data.Frame with labels ⍺ (or the labels of the series list or frame ⍺) and values ⍵. If ⍵ is a string, it writes the frame ⍺ to the CSV file ⍵ or reads the CSV file ⍵ without header and returns frame with labels ⍺.
•data.frame ⍵ creates an instance of data.Frame with each of the series returned by data.series ⍵. If ⍵ is a string, it reads the file ⍵ as CSV with header and returns frame.

data.sort operator

This operator sorts data according to the left function.
•⍺ (⍺⍺ data.sort) ⍵ returns ⍵ (a frame, list of series, or array) sorted according to the result of ⍺⍺ ⍺ (where ⍺⍺ typically is one of ⍒⍋).
•(⍺⍺ data.sort) ⍵ is equivalent to (⍺⍺ data.sort)⍨⍵.

data.by operator

This operator groups data by the right operand and applies the left function.
•⍺ (⍺⍺ data.by ⍵⍵) ⍵ returns the data in ⍵ (a frame or list of series) grouped according to ⍵⍵ (also a frame or list of series) and apply ⍺⍺ to each group. If ⍺⍺ and ⍵⍵ are both series or labels, the values in ⍺⍺ are grouped for each value in ⍵⍵ and distributed in series. Labels (either all of them, the ones not in ⍵, or the ones not in ⍵⍵) are given in ⍺, which can be a list of values, a list of series, or a frame.
•(⍺⍺ data.by ⍵⍵) ⍵ is equivalent to θ (⍺⍺ data.by ⍵⍵) ⍵.
If ⍺⍺ and ⍵⍵ are both series or labels, the values in ⍺⍺ are grouped for each value in ⍵⍵ and distributed in series.

data.join operator

This operator merges two frames (or lists of series).
•⍺ (⍺⍺ data.join ⍵⍵) ⍵ returns frame with series labelled ⍺.labels ⍵⍵ ⍵.labels. If two series at left and right have the same label, its values are combined as ⍺.values ⍺⍺ ⍵.values.
•(⍺⍺ data.join ⍵⍵) ⍵ returns a series with label ⍵⍵ ⍵.labels and values ⍺⍺ ⍵.values.

https://github.com/yiyus/data-science-in-APL/

Thanks to: Morten Kromberg, Richard Park, Martina Crippa, Stine Kromberg, Roger Hui, Stefano Lanzavecchia, ...