University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Uniwersytet Warszawski

Data analysis and visualization (DAV)

Lecture 02

Łukasz P. Kozłowski

Warsaw, 2025

l.kozlowski@mimuw.edu.pl

University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Uniwersytet Warszawski

Data analysis and visualization (DAV)

Lecture 02

Good practices

Łukasz P. Kozłowski

Warsaw, 2025

l.kozlowski@mimuw.edu.pl

When you present the data use **both**:

Plots

Tables

When you present the data use **both**:

Plots

the best solution, very natural and easy to interpretation

Tables

When you present the data use **both**:

Plots

the best solution, very natural and easy to interpretation
(but also prone for miss-interpretation)

Tables

When you present the data use **both**:

Plots

the best solution, very natural and easy to interpretation
(but also prone for miss-interpretation)

Tables

harder to interpret in short time, but higher information content

When you present the data use **both**:

Plots

the best solution, very natural and easy to interpretation
(but also prone for miss-interpretation)

Tables

harder to interpret in short time, but higher information content

* Raw data

for the sake of completeness if you can add them

* The scripts

for the sake of reproducibility if you can add them

When you present the data use **both**:

Plots

the best solution, very natural and easy to interpretation
(but also prone for miss-interpretation)

Tables

harder to interpret in short time, but higher information content

* Raw data

for the sake of completeness if you can add them

* The scripts

for the sake of reproducibility if you can add them

*** The text**

Some proofs:

Some proofs:

93% of human communication is non-verbal

People remember:

80% of what they **see**
and
20% what they **read**

Albert Mahrabian (1971) „Silent Messages”

Some proofs:

93% of human communication is non-verbal

People remember:

80% of what they **see**
and
20% what they **read**

Albert Mahrabian (1971) „Silent Messages”

using visuals will make a presentation 43% more persuasive

Vogel, D. R., Dickson, G. W., & Lehman, J. A. (1986). Persuasion and the role of visual presentation support: The UM/3M study.

Tables

1) Use clean template

2) Make it interactive if possible (html)

Tables (for print)

State	Date	Item	Price	Qty	Amount
CA	28-May	Tent	199	2	398
WA	16-May	Headlamp	39.99	2	79.98
WA	19-May	Sleeping Bag	58.5	1	58.5
WA	13-May	Headlamp	39.99	1	39.99
CA	6-May	Tent	199	3	597
OR	21-May	Backpack	98.77	1	98.77
OR	5-May	Backpack	98.77	1	98.77
CA	1-May	Bike rack	415.75	2	831.5
CA	5-May	Backpack	180.5	1	180.5
CA	4-May	Bike rack	415.75	1	415.75
CA	12-May	Backpack	220.3	1	220.3
CA	4-May	Headlamp	39.99	4	159.96

Method	Protein dataset				Method	Peptide dataset		
	RMSD	%	Outliers			RMSD	%	Outliers
Avg_pI	0.874	0.96	53		Avg_pI	0.454	59.6	1571
Bjellqvist	0.934	0.944	47		Bjellqvist	0.669	161.5	1583
Dawson	0.944	0.945	56		Dawson	0.435	52.9	1432
DTASelect	0.945	1.032	58		DTASelect	0.55	99.1	1714
EMBOSS	0.955	1.056	69		EMBOSS	0.325	18.5	372
Grimsley	0.963	0.968	60		Grimsley	0.616	131.4	1550
IPC_protein	0.966	0.874	46		IPC_petptide	0.251	0	232
Lehninger	0.968	0.97	59		Lehninger	0.262	2.5	236
Nozaki	0.97	1.024	56		Nozaki	0.602	124.3	1368
Patrickios	0.97	2.392	227		Patrickios	1.998	5479.1	2739
pIPredict	1.013	1.048	56		pIPredict	1.024	493.6	2720
pIR	1.024	1.013	58		pIR	1.881	4159.7	3358
ProMoST	1.03	0.966	52		ProMoST	1.239	873.4	2649
Rodwell	1.032	0.963	58		Rodwell	0.502	78.4	1359
Sillero	1.048	1.059	63		Sillero	0.428	50.3	1223
Solomon	1.056	0.97	58		Solomon	0.255	0.9	235
Thurlkill	1.059	1.032	61		Thurlkill	0.481	69.7	1361
Toseland	2.392	0.934	52		Toseland	0.425	49.1	990
Wikipedia	0.96	0.955	55		Wikipedia	0.421	47.9	1467

Method	Protein dataset				Method	Peptide dataset		
	RMSD	%	Outliers			RMSD	%	Outliers
IPC_protein	0.874	0	46		IPC_peptide	0.251	0	232
Toseland	0.934	14.9	52		Solomon	0.255	0.9	235
Bjellqvist	0.944	17.7	47		Lehninger	0.262	2.5	236
Dawson	0.945	17.8	56		EMBOSS	0.325	18.5	372
Wikipedia	0.955	20.5	55		Wikipedia	0.421	47.9	1467
Rodwell	0.963	22.8	58		Toseland	0.425	49.1	990
ProMoST	0.966	23.6	52		Sillero	0.428	50.3	1223
Grimsley	0.968	24.2	60		Dawson	0.435	52.9	1432
Solomon	0.97	24.8	58		Thurlkill	0.481	69.7	1361
Lehninger	0.97	25	59		Rodwell	0.502	78.4	1359
pIR	1.013	38	58		DTASelect	0.550	99.1	1714
Nozaki	1.024	41.3	56		Nozaki	0.602	124.3	1368
Thurlkill	1.03	43.4	61		Grimsley	0.616	131.4	1550
DTASelect	1.032	44.1	58		Bjellqvist	0.669	161.5	1583
pIPredict	1.048	49.4	56		pIPredict	1.024	493.6	2720
EMBOSS	1.056	52.3	69		ProMoST	1.239	873.4	2649
Sillero	1.059	53.2	63		pIR	1.881	4159.7	3358
Patrickios	2.392	3201.8	227		Patrickios	1.998	5479.1	2739
Avg_pI	0.96	22.1	53		Avg_pI	0.454	59.6	1571

Method	Protein dataset				Method	Peptide dataset		
	RMSD	%	Outliers			RMSD	%	Outliers
IPC_protein	0.874	0	46		IPC_peptide	0.251	0	232
Toseland	0.934	14.9	52		Solomon	0.255	0.9	235
Bjellqvist	0.944	17.7	47		Lehninger	0.262	2.5	236
Dawson	0.945	17.8	56		EMBOSS	0.325	18.5	372
Wikipedia	0.955	20.5	55		Wikipedia	0.421	47.9	1467
Rodwell	0.963	22.8	58		Toseland	0.425	49.1	990
ProMoST	0.966	23.6	52		Sillero	0.428	50.3	1223
Grimsley	0.968	24.2	60		Dawson	0.435	52.9	1432
Solomon	0.97	24.8	58		Thurlkill	0.481	69.7	1361
Lehninger	0.97	25	59		Rodwell	0.502	78.4	1359
pIR	1.013	38	58		DTASelect	0.550	99.1	1714
Nozaki	1.024	41.3	56		Nozaki	0.602	124.3	1368
Thurlkill	1.03	43.4	61		Grimsley	0.616	131.4	1550
DTASelect	1.032	44.1	58		Bjellqvist	0.669	161.5	1583
pIPredict	1.048	49.4	56		pIPredict	1.024	493.6	2720
EMBOSS	1.056	52.3	69		ProMoST	1.239	873.4	2649
Sillero	1.059	53.2	63		pIR	1.881	4159.7	3358
Patrickios	2.392	3201.8	227		Patrickios	1.998	5479.1	2739
Avg_pI	0.96	22.1	53		Avg_pI	0.454	59.6	1571

Sort (decide how, use html if possible)

Method	Protein dataset				Method	Peptide dataset		
	RMSD	%	Outliers			RMSD	%	Outliers
IPC_protein	0.874	0	46		IPC_peptide	0.251	0	232
Toseland	0.934	14.9	52		Solomon	0.255	0.9	235
Bjellqvist	0.944	17.7	47		Lehninger	0.262	2.5	236
Dawson	0.945	17.8	56		EMBOSS	0.325	18.5	372
Wikipedia	0.955	20.5	55		Wikipedia	0.421	47.9	1467
Rodwell	0.963	22.8	58		Toseland	0.425	49.1	990
ProMoST	0.966	23.6	52		Sillero	0.428	50.3	1223
Grimsley	0.968	24.2	60		Dawson	0.435	52.9	1432
Solomon	0.97	24.8	58		Thurlkill	0.481	69.7	1361
Lehninger	0.97	25	59		Rodwell	0.502	78.4	1359
pIR	1.013	38	58		DTASelect	0.550	99.1	1714
Nozaki	1.024	41.3	56		Nozaki	0.602	124.3	1368
Thurlkill	1.03	43.4	61		Grimsley	0.616	131.4	1550
DTASelect	1.032	44.1	58		Bjellqvist	0.669	161.5	1583
pIPredict	1.048	49.4	56		pIPredict	1.024	493.6	2720
EMBOSS	1.056	52.3	69		ProMoST	1.239	873.4	2649
Sillero	1.059	53.2	63		pIR	1.881	4159.7	3358
Patrickios	2.392	3201.8	227		Patrickios	1.998	5479.1	2739
Avg_pI	0.96	22.1	53		Avg_pI	0.454	59.6	1571

Method	Protein dataset			Method	Peptide dataset		
	RMSD	%	Outliers		RMSD	%	Outliers
IPC_protein	0.874	0	46	IPC_peptide	0.251	0	232
Toseland	0.934	14.9	52	Solomon	0.255	0.9	235
Bjellqvist	0.944	17.7	47	Lehninger	0.262	2.5	236
Dawson	0.945	17.8	56	EMBOSS	0.325	18.5	372
Wikipedia	0.955	20.5	55	Wikipedia	0.421	47.9	1467
Rodwell	0.963	22.8	58	Toseland	0.425	49.1	990
ProMoST	0.966	23.6	52	Sillero	0.428	50.3	1223
Grimsley	0.968	24.2	60	Dawson	0.435	52.9	1432
Solomon	0.97	24.8	58	Thurlkill	0.481	69.7	1361
Lehninger	0.97	25	59	Rodwell	0.502	78.4	1359
pIR	1.013	38	58	DTASelect	0.550	99.1	1714
Nozaki	1.024	41.3	56	Nozaki	0.602	124.3	1368
Thurlkill	1.03	43.4	61	Grimsley	0.616	131.4	1550
DTASelect	1.032	44.1	58	Bjellqvist	0.669	161.5	1583
pIPredict	1.048	49.4	56	pIPredict	1.024	493.6	2720
EMBOSS	1.056	52.3	69	ProMoST	1.239	873.4	2649
Sillero	1.059	53.2	63	pIR	1.881	4159.7	3358
Patrickios	2.392	3201.8	227	Patrickios	1.998	5479.1	2739
Avg_pI	0.96	22.1	53	Avg_pI	0.454	59.6	1571

Bold & Align, the same font (size, type, consider using monotype font for better alignment)

Optimal width of columns and vertical and horizontal alignment, avoid blank spaces

Method	Protein dataset				Method	Peptide dataset		
	RMSD	%	Outliers			RMSD	%	Outliers
IPC_protein	0.874	0.0	46		IPC_peptide	0.251	0.0	232
Toseland	0.934	14.9	52		Solomon	0.255	0.9	235
Bjellqvist	0.944	17.7	47		Lehninger	0.262	2.5	236
Dawson	0.945	17.8	56		EMBOSS	0.325	18.5	372
Wikipedia	0.955	20.5	55		Wikipedia	0.421	47.9	1467
Rodwell	0.963	22.8	58		Toseland	0.425	49.1	990
ProMoST	0.966	23.6	52		Sillero	0.428	50.3	1223
Grimsley	0.968	24.2	60		Dawson	0.435	52.9	1432
Solomon	0.970	24.8	58		Thurlkill	0.481	69.7	1361
Lehninger	0.970	25.0	59		Rodwell	0.502	78.4	1359
pIR	1.013	38.0	58		DTASelect	0.550	99.1	1714
Nozaki	1.024	41.3	56		Nozaki	0.602	124.3	1368
Thurlkill	1.030	43.4	61		Grimsley	0.616	131.4	1550
DTASelect	1.032	44.1	58		Bjellqvist	0.669	161.5	1583
pIPredict	1.048	49.4	56		pIPredict	1.024	493.6	2720
EMBOSS	1.056	52.3	69		ProMoST	1.239	873.4	2649
Sillero	1.059	53.2	63		pIR	1.881	4159.7	3358
Patrickios	2.392	3201.8	227		Patrickios	1.998	5479.1	2739
Avg_pI	0.960	22.1	53		Avg_pI	0.454	59.6	1571

Use the same decimal point (do not round at different levels)

Method	Protein dataset				Method	Peptide dataset		
	RMSD	%	Outliers			RMSD	%	Outliers
IPC_protein	0.874	0.0	46		IPC_peptide	0.251	0.0	232
Toseland	0.934	14.9	52		Solomon	0.255	0.9	235
Bjellqvist	0.944	17.7	47		Lehninger	0.262	2.5	236
Dawson	0.945	17.8	56		EMBOSS	0.325	18.5	372
Wikipedia	0.955	20.5	55		Wikipedia	0.421	47.9	1467
Rodwell	0.963	22.8	58		Toseland	0.425	49.1	990
ProMoST	0.966	23.6	52		Sillero	0.428	50.3	1223
Grimsley	0.968	24.2	60		Dawson	0.435	52.9	1432
Solomon	0.970	24.8	58		Thurlkill	0.481	69.7	1361
Lehninger	0.970	25.0	59		Rodwell	0.502	78.4	1359
pIR	1.013	38.0	58		DTASelect	0.550	99.1	1714
Nozaki	1.024	41.3	56		Nozaki	0.602	124.3	1368
Thurlkill	1.030	43.4	61		Grimsley	0.616	131.4	1550
DTASelect	1.032	44.1	58		Bjellqvist	0.669	161.5	1583
pIPredict	1.048	49.4	56		pIPredict	1.024	493.6	2720
EMBOSS	1.056	52.3	69		ProMoST	1.239	873.4	2649
Sillero	1.059	53.2	63		pIR	1.881	4159.7	3358
Patrickios	2.392	3201.8	227		Patrickios	1.998	5479.1	2739
Avg_pI	0.960	22.1	53		Avg_pI	0.454	59.6	1571

Less is more (hide some of the borders and make them ticker)

	Protein dataset				Peptide dataset		
Method	RMSD	%	Outliers	Method	RMSD	%	Outliers
IPC_protein	0.874	0.0	46	IPC_peptide	0.251	0.0	232
Toseland	0.934	14.9	52	Solomon	0.255	0.9	235
Bjellqvist	0.944	17.7	47	Lehninger	0.262	2.5	236
Dawson	0.945	17.8	56	EMBOSS	0.325	18.5	372
Wikipedia	0.955	20.5	55	Wikipedia	0.421	47.9	1467
Rodwell	0.963	22.8	58	Toseland	0.425	49.1	990
ProMoST	0.966	23.6	52	Sillero	0.428	50.3	1223
Grimsley	0.968	24.2	60	Dawson	0.435	52.9	1432
Solomon	0.970	24.8	58	Thurlkill	0.481	69.7	1361
Lehninger	0.970	25.0	59	Rodwell	0.502	78.4	1359
pIR	1.013	38.0	58	DTASelect	0.550	99.1	1714
Nozaki	1.024	41.3	56	Nozaki	0.602	124.3	1368
Thurlkill	1.030	43.4	61	Grimsley	0.616	131.4	1550
DTASelect	1.032	44.1	58	Bjellqvist	0.669	161.5	1583
pIPredict	1.048	49.4	56	pIPredict	1.024	493.6	2720
EMBOSS	1.056	52.3	69	ProMoST	1.239	873.4	2649
Sillero	1.059	53.2	63	pIR	1.881	4159.7	3358
Patrickios	2.392	3201.8	227	Patrickios	1.998	5479.1	2739
Avg_pI*	0.960	22.1	53	Avg_pI	0.454	59.6	1571

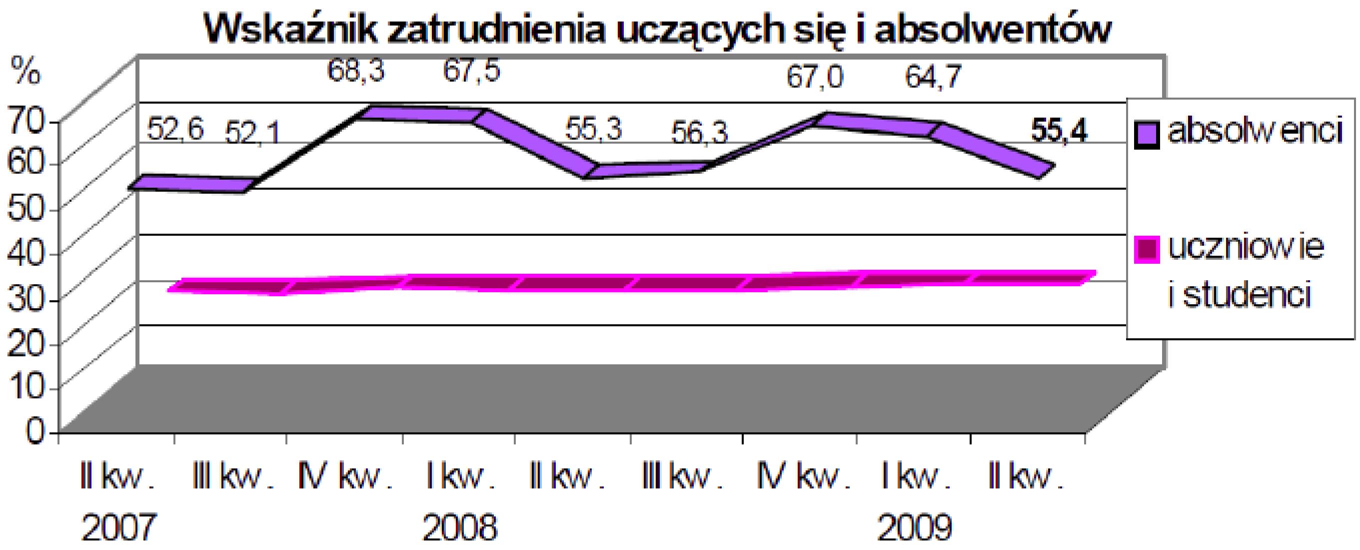
Less is more

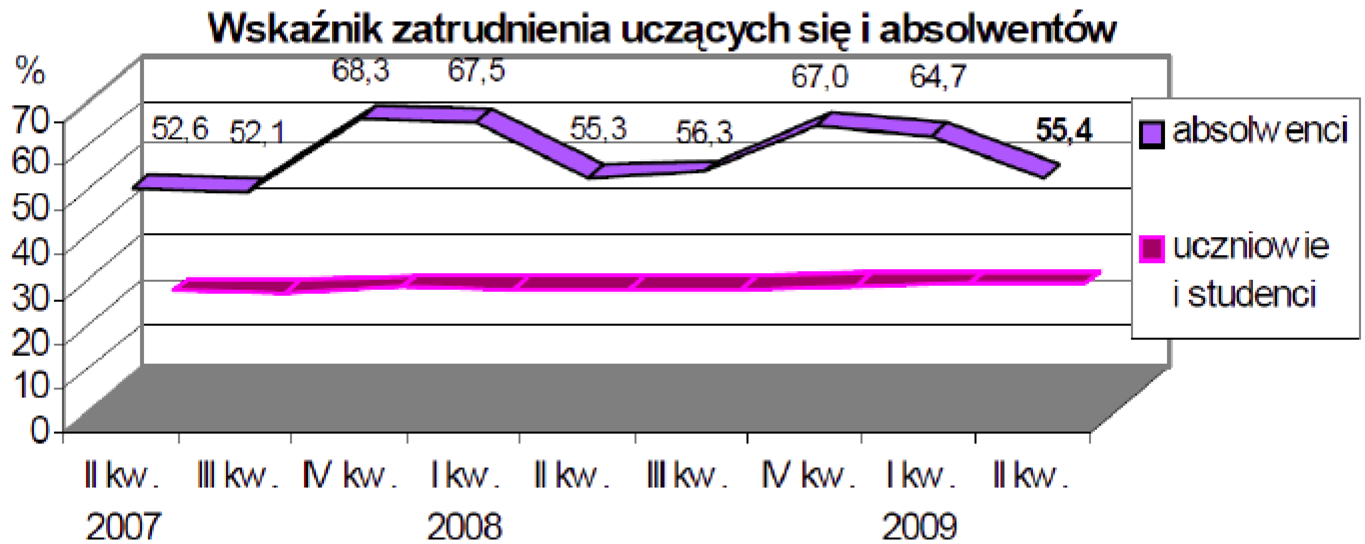
avoid as many blank space as possible, correct the width of columns

	Protein dataset				Peptide dataset		
Method	RMSD	%	Outliers	Method	RMSD	%	Outliers
IPC_protein	0.874	0.0	46	IPC_peptide	0.251	0.0	232
Toseland	0.934	14.9	52	Solomon	0.255	0.9	235
Bjellqvist	0.944	17.7	47	Lehninger	0.262	2.5	236
Dawson	0.945	17.8	56	EMBOSS	0.325	18.5	372
Wikipedia	0.955	20.5	55	Wikipedia	0.421	47.9	1467
Rodwell	0.963	22.8	58	Toseland	0.425	49.1	990
ProMoST	0.966	23.6	52	Sillero	0.428	50.3	1223
Grimsley	0.968	24.2	60	Dawson	0.435	52.9	1432
Solomon	0.970	24.8	58	Thurlkill	0.481	69.7	1361
Lehninger	0.970	25.0	59	Rodwell	0.502	78.4	1359
pIR	1.013	38.0	58	DTASelect	0.550	99.1	1714
Nozaki	1.024	41.3	56	Nozaki	0.602	124.3	1368
Thurlkill	1.030	43.4	61	Grimsley	0.616	131.4	1550
DTASelect	1.032	44.1	58	Bjellqvist	0.669	161.5	1583
pIPredict	1.048	49.4	56	pIPredict	1.024	493.6	2720
EMBOSS	1.056	52.3	69	ProMoST	1.239	873.4	2649
Sillero	1.059	53.2	63	pIR	1.881	4159.7	3358
Patrickios	2.392	3201.8	227	Patrickios	1.998	5479.1	2739
Avg_pI*	0.960	22.1	53	Avg_pI	0.454	59.6	1571

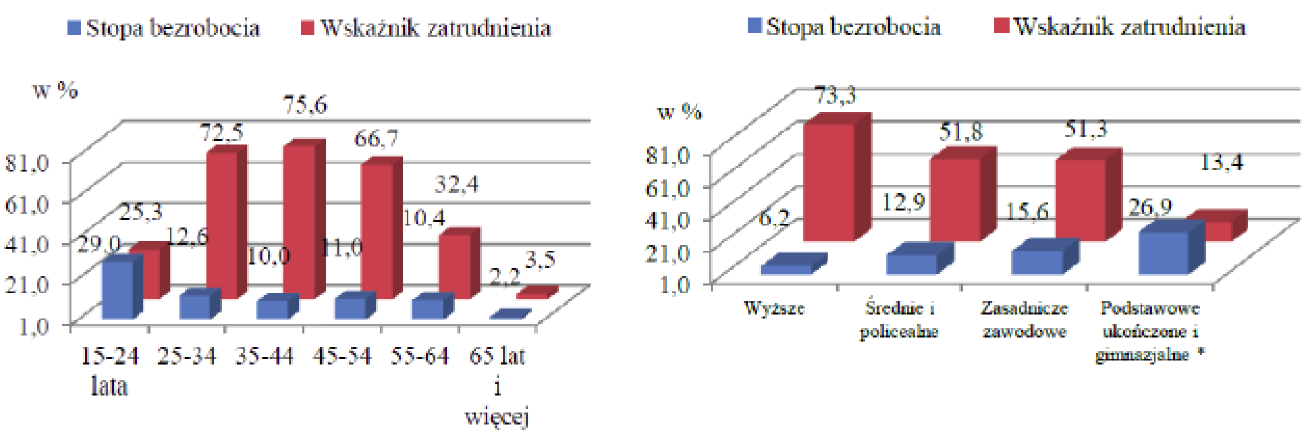
Each table should be single script


CSV
json
XML


Wskaźnik zatrudnienia uczących się i absolwentów
%
70
60
50
40
30
20
10
0
52,6
52,1
68,3
67,5
55,3
56,3
67,0
64,7
55,4
absolwenci
uczniowie i studenci
II kw. 2007
III kw.
IV kw.
I kw. 2008
II kw.
III kw.
IV kw.
I kw. 2009
II kw.



Wykres 7. Wskaźnik zatrudnienia oraz stopa bezrobocia według grup wieku i poziomu wykształcenia w 2011 r.



* łącznie z wykształceniem podstawowym nieukończonym i bez wykształcenia szkolnego

Data formats


csv
>
json
>
XML


CSV

>


json

>


XML


CSV

CSV

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

CSV

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

printable ASCII or Unicode characters.

CSV

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

printable ASCII or Unicode characters.

Other popular delimiters include the tab (\t), colon (:) and semi-colon (;) characters. Properly parsing a CSV file requires us to know which delimiter is being used.

CSV

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

printable ASCII or Unicode characters.

Other popular delimiters include the tab (\t), colon (:) and semi-colon (;) characters. Properly parsing a CSV file requires us to know which delimiter is being used.

CSV files are very easy to work with programmatically. Any language that supports text file input and string manipulation (like Python) can work with CSV files directly.

CSV

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

printable ASCII or Unicode characters.

Other popular delimiters include the tab (\t), colon (:) and semi-colon (;) characters. Properly parsing a CSV file requires us to know which delimiter is being used.

CSV files are very easy to work with programmatically. Any language that supports text file input and string manipulation (like Python) can work with CSV files directly.

Nice to work with unix commands like: **head, tail, split, cat,** etc.

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

Here's code to read it:

Python

```
import csv

with open('employee_birthday.txt') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, and was born
            line_count += 1
    print(f'Processed {line_count} lines.')
```

https://realpython.com/python-csv/

CSV

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

Here's code to read it:

Python

```python
import csv

with open('employee_birthday.txt') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, and was born 
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Shell

```
Column names are name, department, birthday month
    John Smith works in the Accounting department, and was born in November.
    Erica Meyers works in the IT department, and was born in March.
Processed 3 lines.
```

https://realpython.com/python-csv/

```
import csv

import pandas
```

https://realpython.com/python-csv/

```
import csv

import pandas
df = pandas.read_csv('hrdata.csv')
print(df)
```

Shell

```
             Name Hire Date   Salary  Sick Days remaining
0  Graham Chapman  03/15/14  50000.0                   10
1     John Cleese  06/01/15  65000.0                    8
2       Eric Idle  05/12/14  45000.0                   10
3     Terry Jones  11/01/13  70000.0                    3
4   Terry Gilliam  08/12/14  48000.0                    7
5   Michael Palin  05/23/13  66000.0                    8
```

https://realpython.com/python-csv/

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```


```
import json

person = '{"name": "Bob", "languages": ["English", "Fench"]}'
person_dict = json.loads(person)

# Output: {'name': 'Bob', 'languages': ['English', 'Fench']}
print( person_dict)

# Output: ['English', 'French']
print(person_dict['languages'])
```


```python
import json

person = '{"name": "Bob", "languages": ["English", "Fench"]}'
person_dict = json.loads(person)

# Output: {'name': 'Bob', 'languages': ['English', 'Fench']}
print( person_dict)

# Output: ['English', 'French']
print(person_dict['languages'])
```

```python
import json

person_dict = {'name': 'Bob',
'age': 12,
'children': None
}
person_json = json.dumps(person_dict)

# Output: {"name": "Bob", "age": 12, "children": null}
print(person_json)
```

Python	JSON Equivalent
`dict`	object
`list`, `tuple`	array
`str`	string
`int`, `float`, `int`	number
`True`	true
`False`	false
`None`	null

MUST be in UTF-8

https://www.programiz.com/python-programming/json

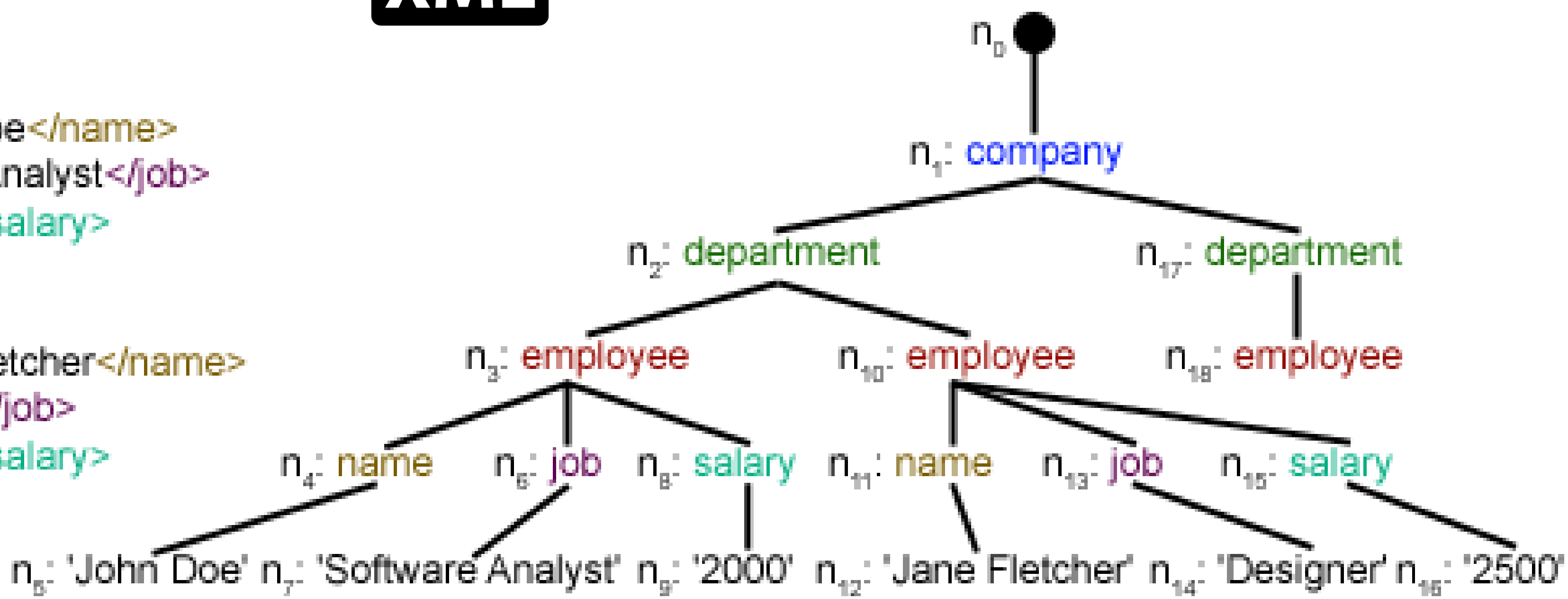
```
$ python test_serialization_speed.py
--------------------
   Encoding Tests
--------------------
Encoding: 100000 x {'m': 'asdsasdqwqw', 't': 3}
[      json] 1.12385 seconds for 100000 runs. avg: 0.011239ms
[simplejson] 0.44356 seconds for 100000 runs. avg: 0.004436ms
[     cjson] 0.09593 seconds for 100000 runs. avg: 0.000959ms

Encoding: 10000 x {'m': [['0', 1, '2', 3, '4', 5, '6', 7, '8', 9, '10', 11, '12', 13,
[      json] 7.76628 seconds for 10000 runs. avg: 0.776628ms
[simplejson] 0.51179 seconds for 10000 runs. avg: 0.051179ms
[     cjson] 0.44362 seconds for 10000 runs. avg: 0.044362ms

--------------------
   Decoding Tests
--------------------
Decoding: 100000 x {"m": "asdsasdqwqw", "t": 3}
[      json] 3.32861 seconds for 100000 runs. avg: 0.033286ms
[simplejson] 0.37164 seconds for 100000 runs. avg: 0.003716ms
[     cjson] 0.03893 seconds for 100000 runs. avg: 0.000389ms

Decoding: 10000 x {"m": [["0", 1, "2", 3, "4", 5, "6", 7, "8", 9, "10", 11, "12", 13,
[      json] 37.26270 seconds for 10000 runs. avg: 3.726270ms
[simplejson] 0.56643 seconds for 10000 runs. avg: 0.056643ms
[     cjson] 0.33007 seconds for 10000 runs. avg: 0.033007ms
```

JSON libraries in python:

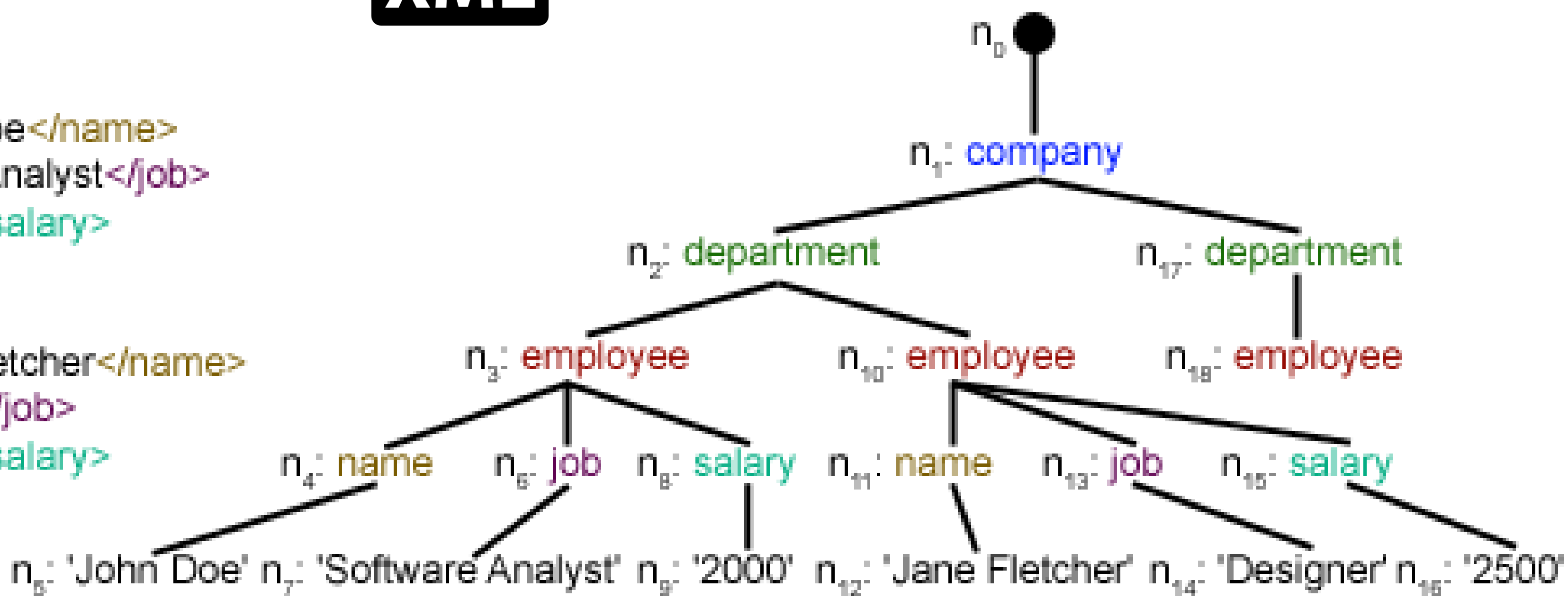
simplejson, ujson, cjson, ...

https://stackoverflow.com/questions/712791/what-are-the-differences-between-json-and-simplejson-python-modules

```xml
<company>
  <department>
    <employee>
      <name>John Doe</name>
      <job>Software Analyst</job>
      <salary>2000</salary>
    </employee>
    <employee>
      <name>Jane Fletcher</name>
      <job>Designer</job>
      <salary>2500</salary>
    </employee>
  </department>
  <department>
    <employee>
    </employee>
  </department>
</company>
```



```xml
<company>
  <department>
    <employee>
      <name>John Doe</name>
      <job>Software Analyst</job>
      <salary>2000</salary>
    </employee>
    <employee>
      <name>Jane Fletcher</name>
      <job>Designer</job>
      <salary>2500</salary>
    </employee>
  </department>
  <department>
    <employee>
    </employee>
  </department>
</company>
```

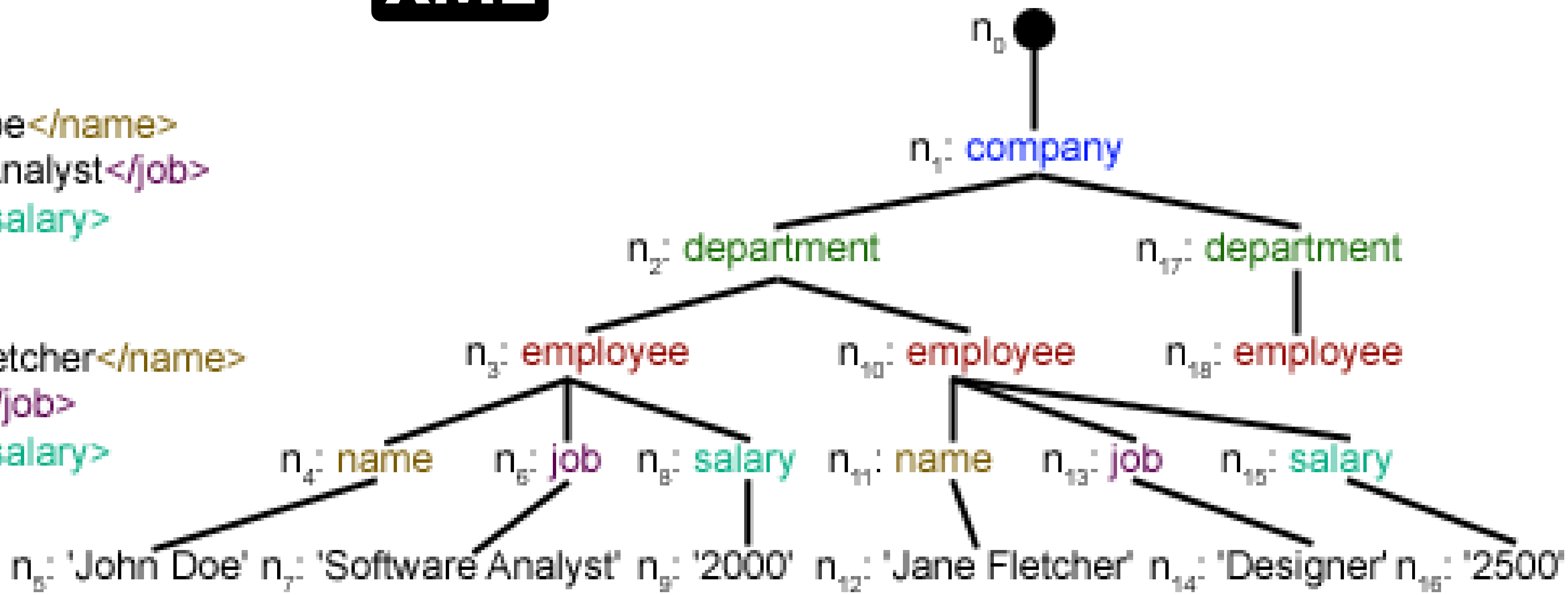


xml.etree.ElementTree

xml.dom.minidom

etree.XMLParser() [from lxml]

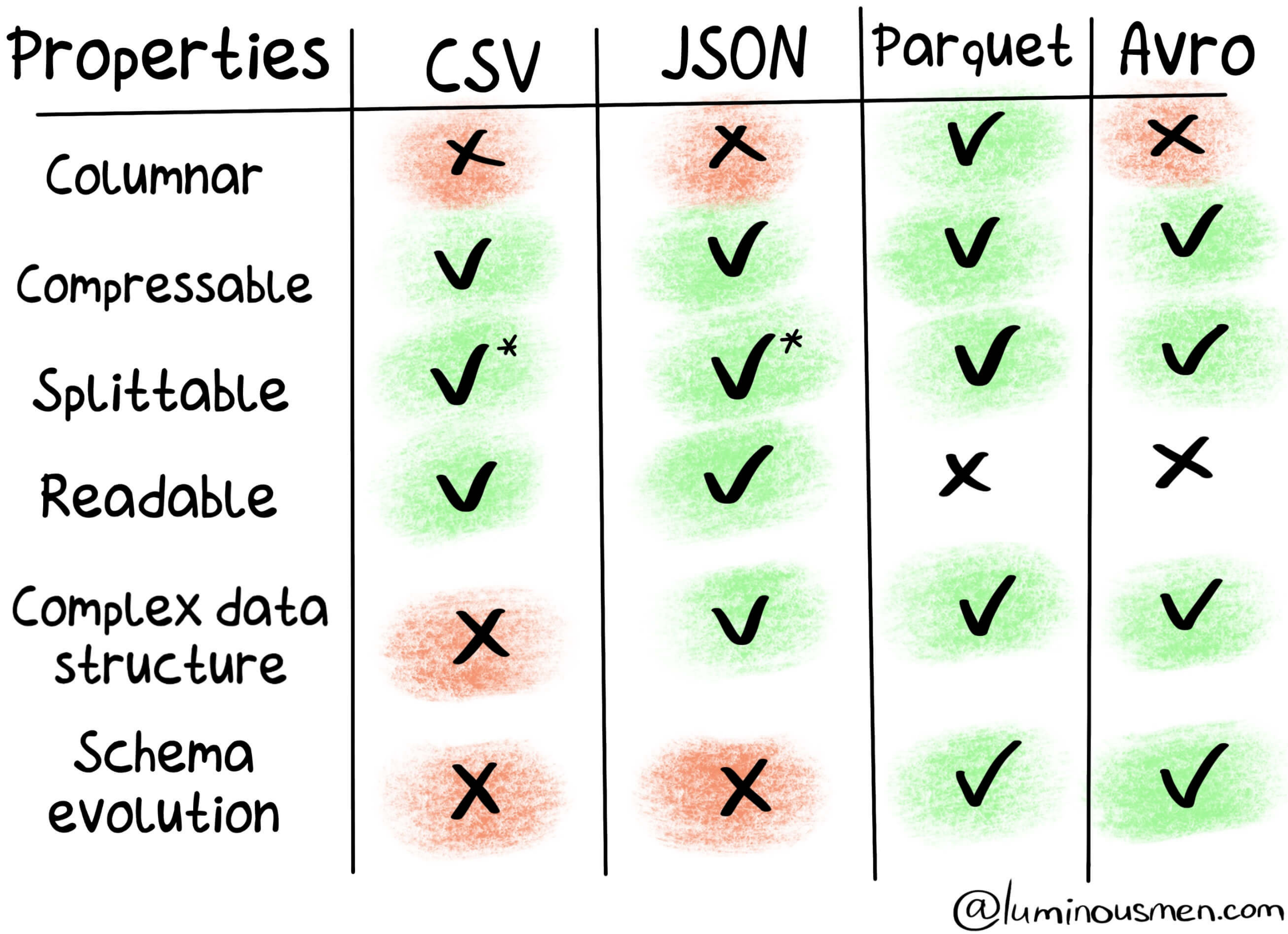
```xml
<company>
  <department>
    <employee>
      <name>John Doe</name>
      <job>Software Analyst</job>
      <salary>2000</salary>
    </employee>
    <employee>
      <name>Jane Fletcher</name>
      <job>Designer</job>
      <salary>2500</salary>
    </employee>
  </department>
  <department>
    <employee>
    </employee>
  </department>
</company>
```



XML is not known for being short and sweet

Human-readable, although …

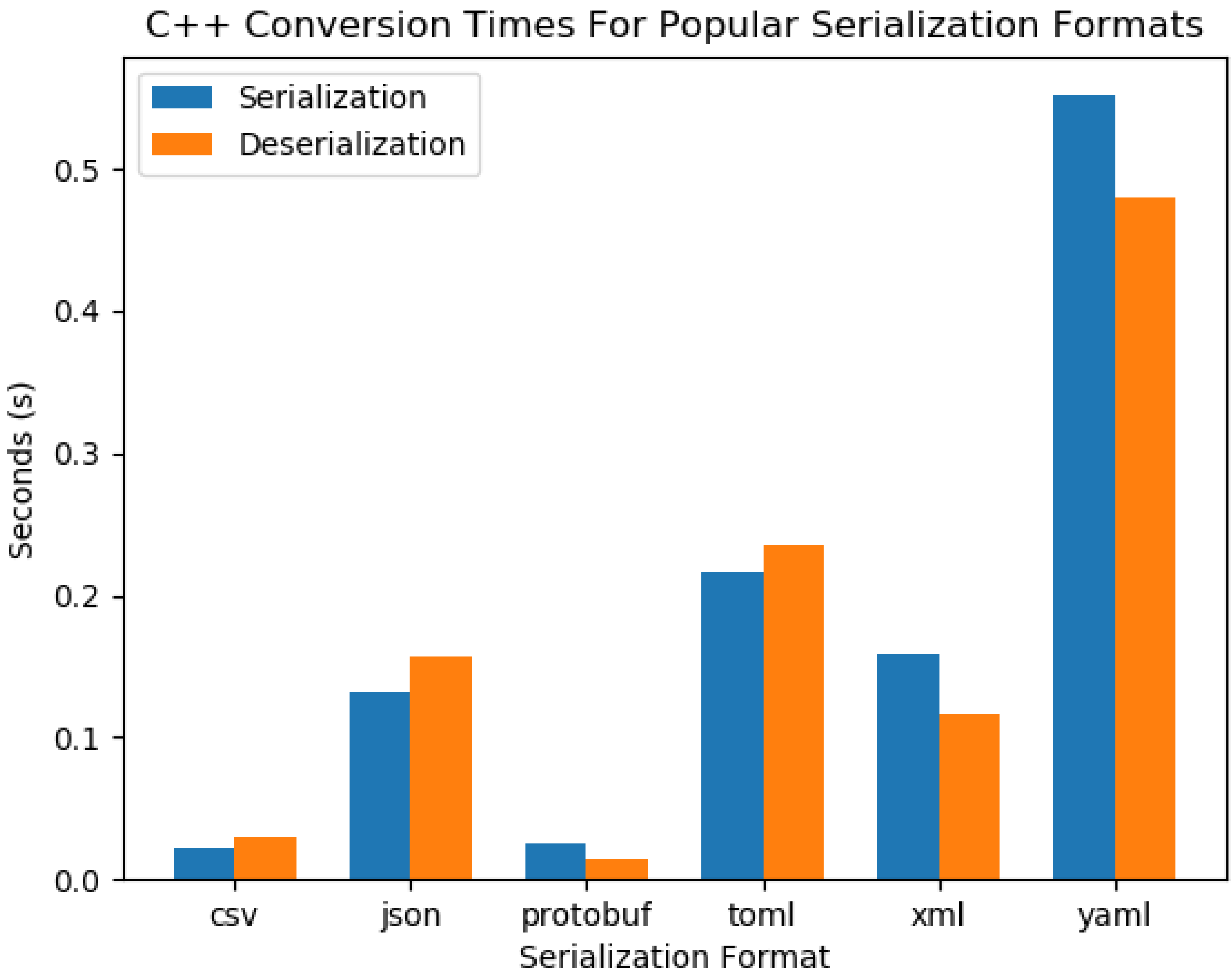
you can get lost in-between all the tags in-front of your eyes



Properties	CSV	JSON	Parquet	Avro
Columnar	✗	✗	✓	✗
Compressable	✓	✓	✓	✓
Splittable	✓*	✓*	✓	✓
Readable	✓	✓	✗	✗
Complex data structure	✗	✓	✓	✓
Schema evolution	✗	✗	✓	✓

Parquet (Apache) – column based format

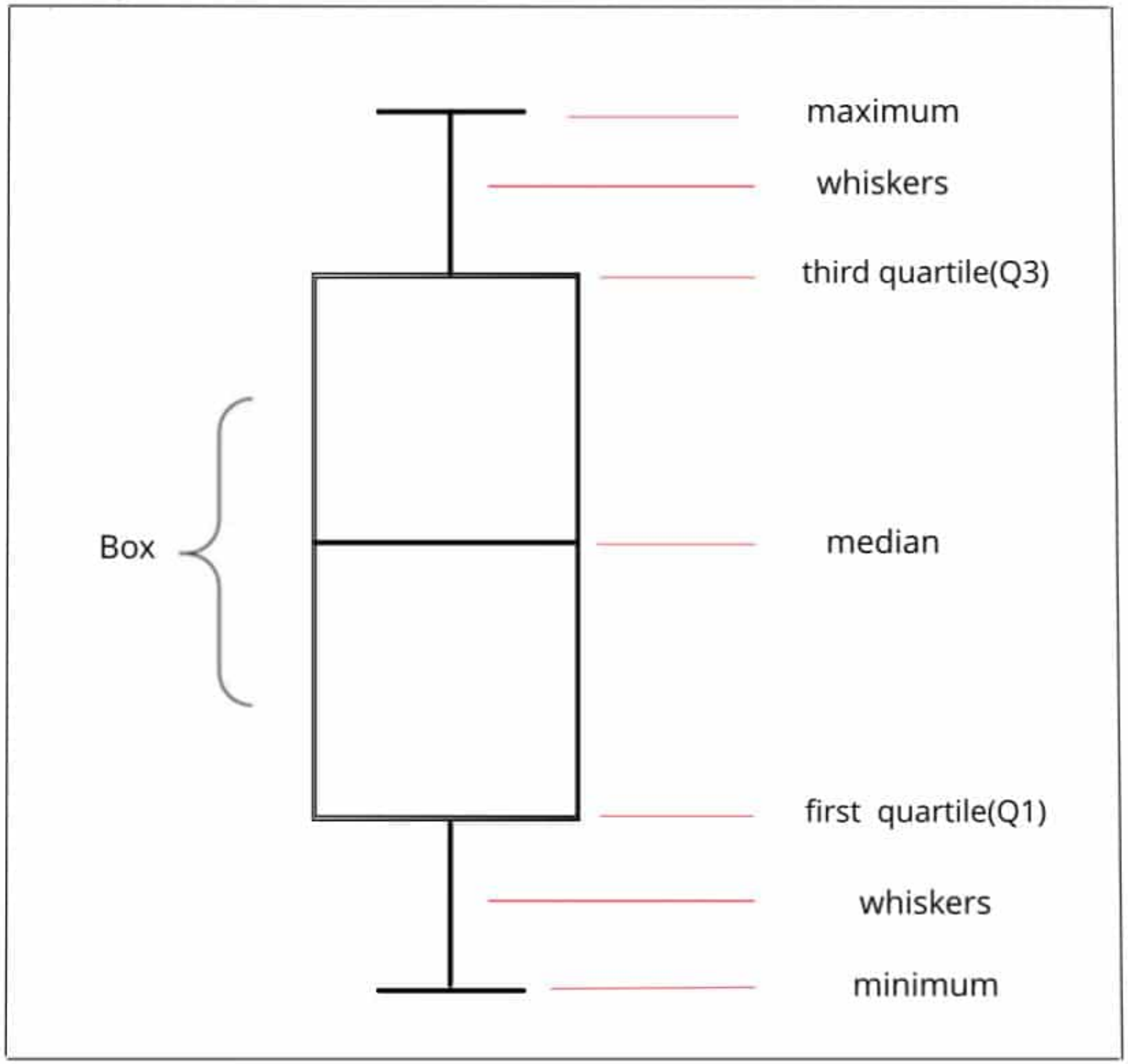
Avro (Hadoop) – row based format

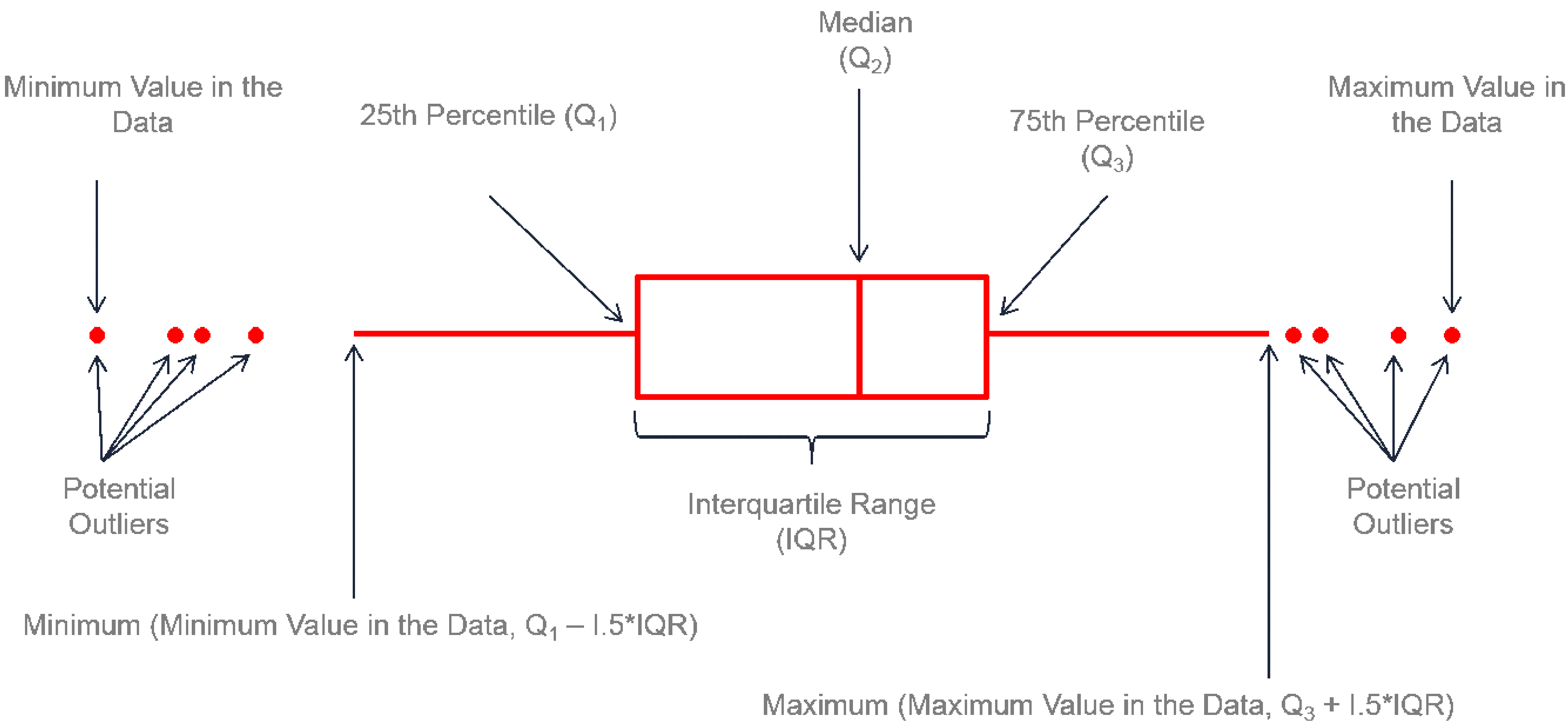


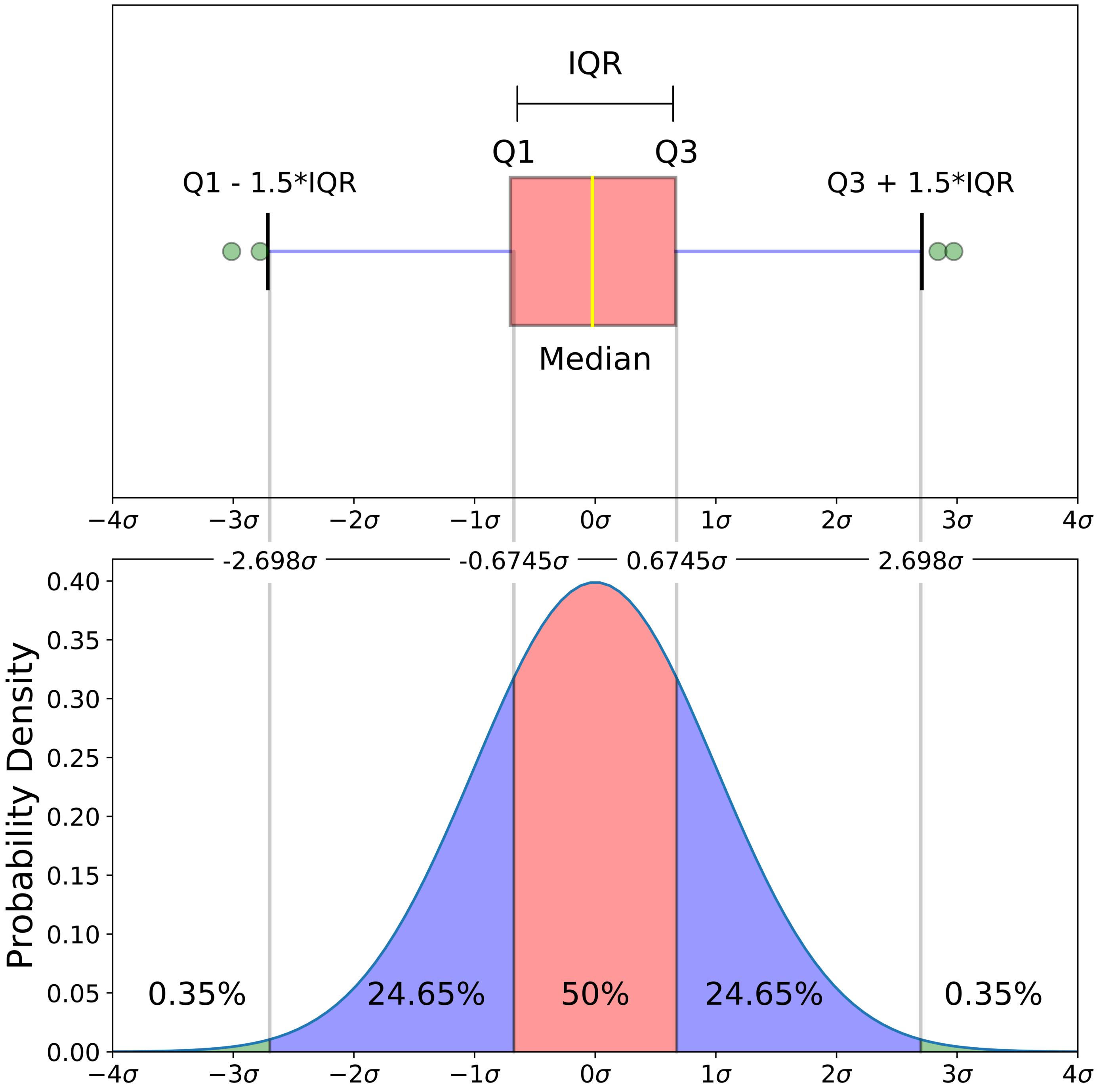
https://blog.mbedded.ninja/programming/serialization-formats/a-comparison-of-serialization-formats/

Format	File Size (MiB, 10k records)	File Size (MiB, 100k records)
csv	0.41	4.2
json	0.81	8.2
xml	1.50	15
yaml	0.80	8.1

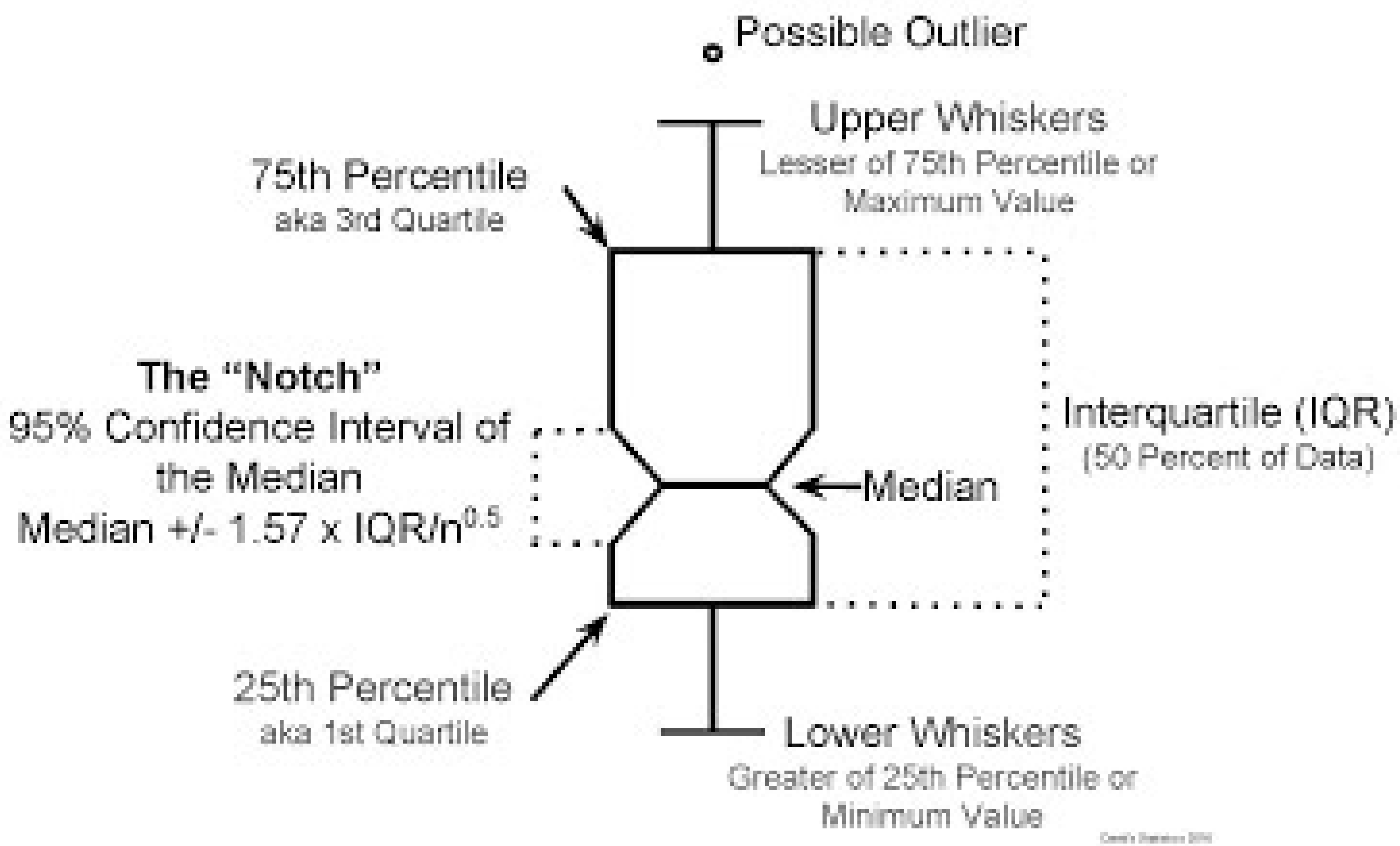
https://blog.mbedded.ninja/programming/serialization-formats/a-comparison-of-serialization-formats/


maximum
whiskers
third quartile(Q3)
Box
median
first quartile(Q1)
whiskers
minimum

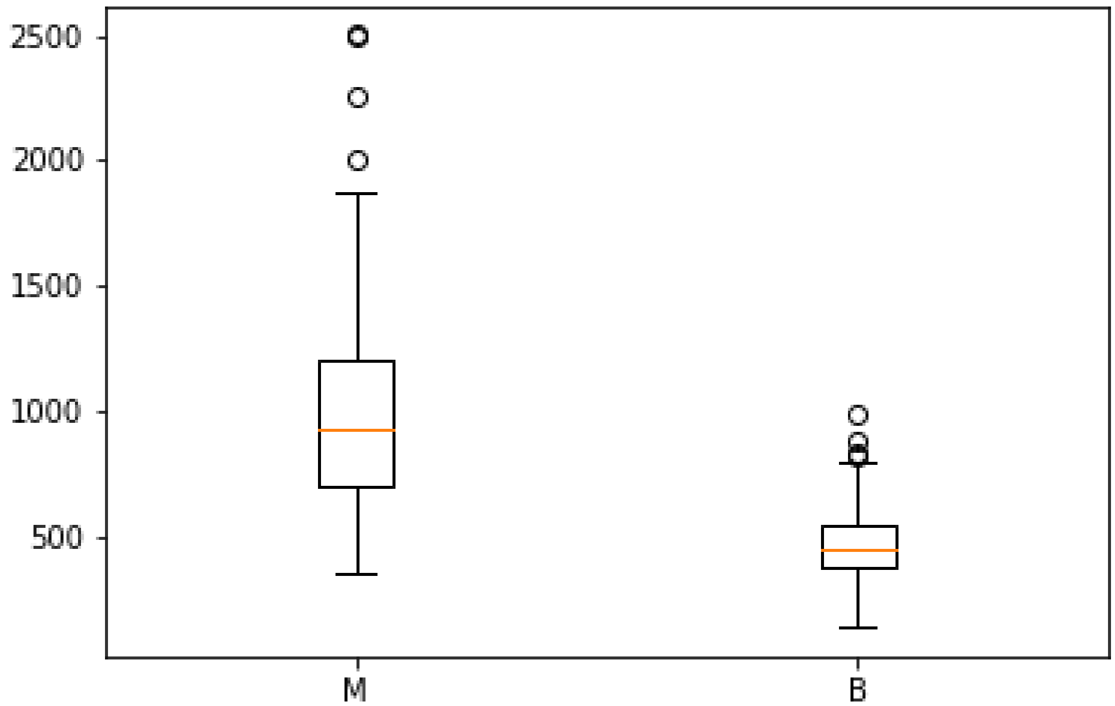

Median
(Q2)
Minimum Value in the Data
25th Percentile (Q1)
75th Percentile (Q3)
Maximum Value in the Data
Potential Outliers
Interquartile Range (IQR)
Potential Outliers
Minimum (Minimum Value in the Data, Q1 – 1.5*IQR)
Maximum (Maximum Value in the Data, Q3 + 1.5*IQR)

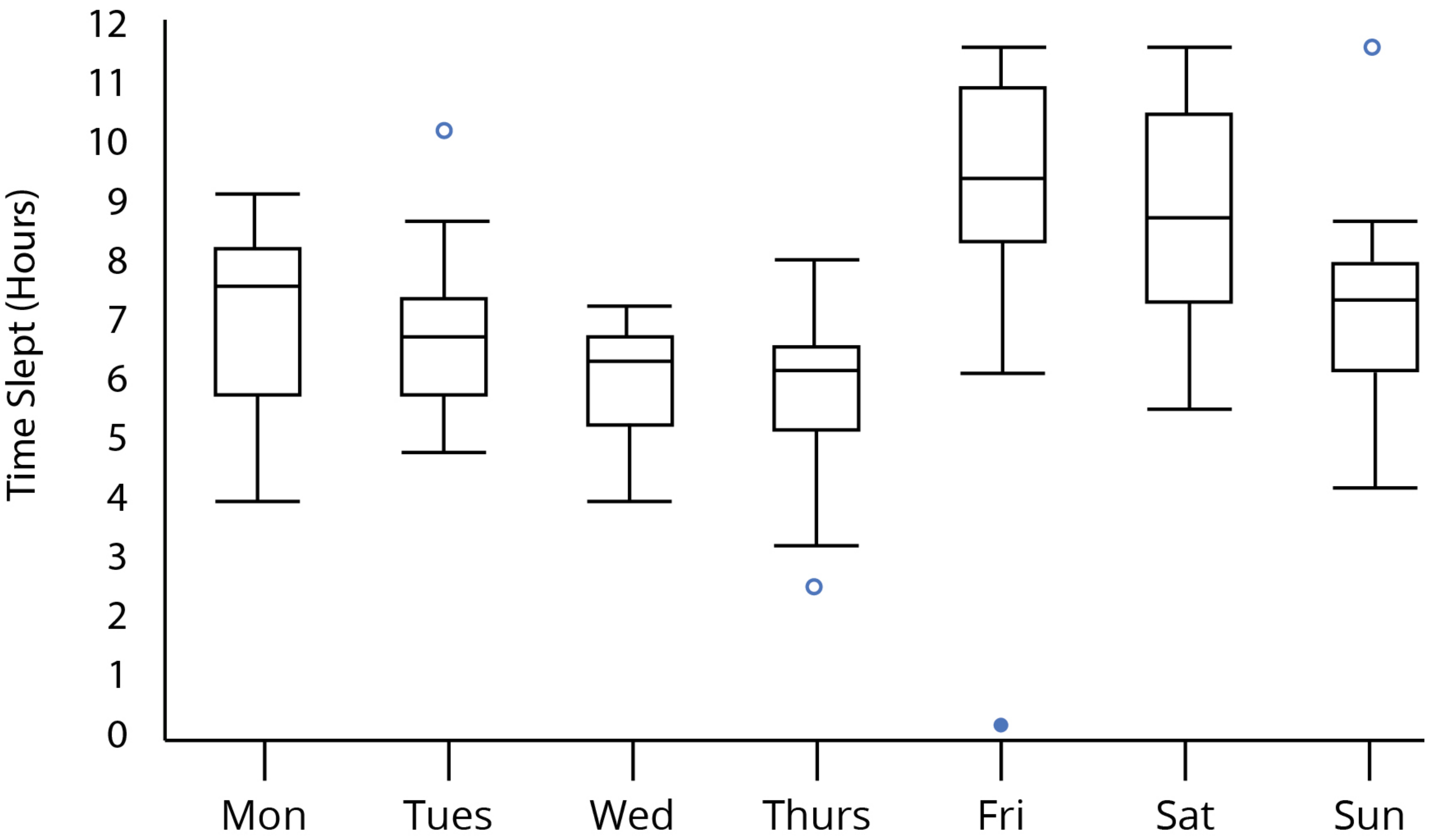


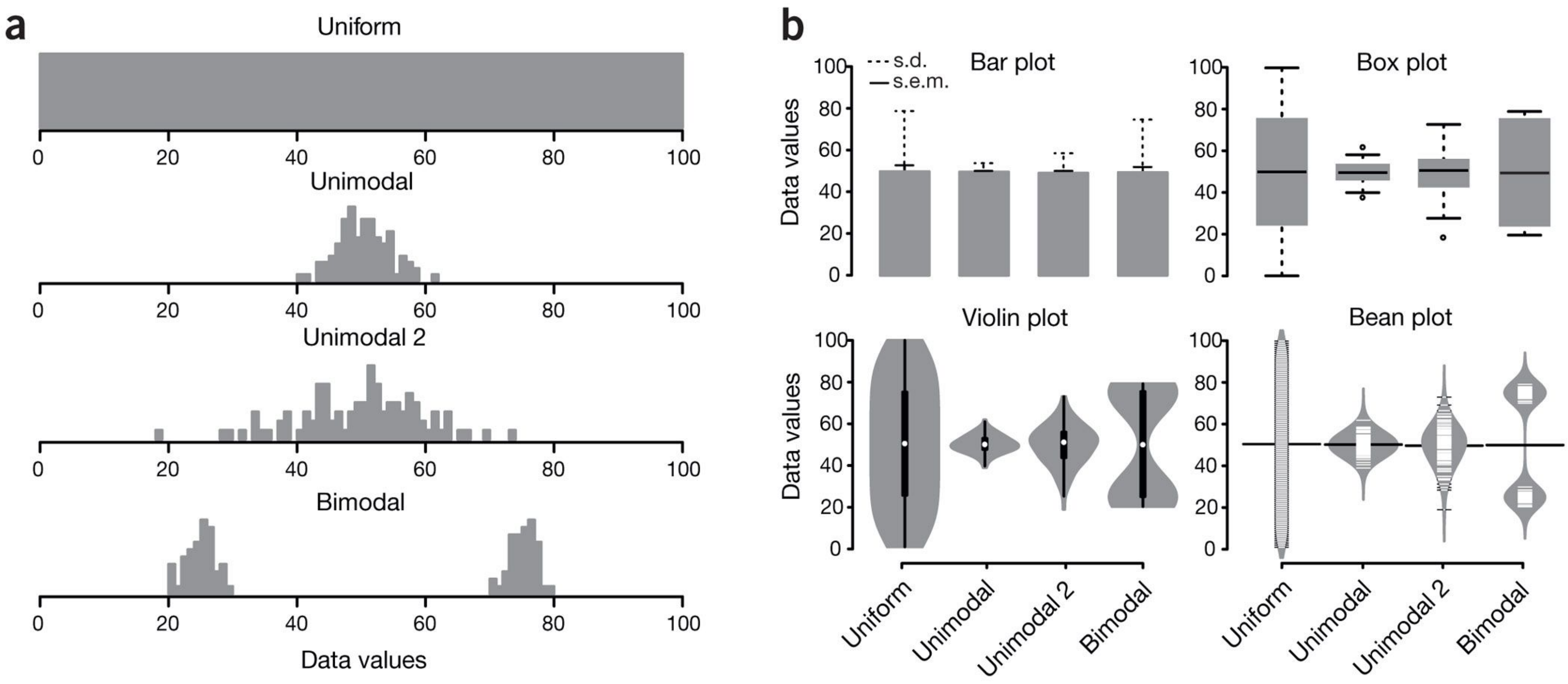
IQR
Q1
Q3
Q1 - 1.5*IQR
Q3 + 1.5*IQR
Median
−4σ −3σ −2σ −1σ 0σ 1σ 2σ 3σ 4σ
-2.698σ
-0.6745σ
0.6745σ
2.698σ
Probability Density
0.40 0.35 0.30 0.25 0.20 0.15 0.10 0.05 0.00
0.35%
24.65%
50%
24.65%
0.35%


Possible Outlier
Upper Whiskers
Lesser of 75th Percentile or
Maximum Value
75th Percentile
aka 3rd Quartile
The "Notch"
95% Confidence Interval of
the Median
Median +/- 1.57 x IQR/n^0.5
Interquartile (IQR)
(50 Percent of Data)
Median
25th Percentile
aka 1st Quartile
Lower Whiskers
Greater of 25th Percentile or
Minimum Value

```
malignant = df[df['diagnosis']=='M']['area_mean']
benign = df[df['diagnosis']=='B']['area_mean']

fig = plt.figure()
ax = fig.add_subplot(111)
ax.boxplot([malignant,benign], labels=['M', 'B'])
```




Time Slept (Hours)
0
1
2
3
4
5
6
7
8
9
10
11
12
Mon
Tues
Wed
Thurs
Fri
Sat
Sun


a
Uniform
Unimodal
Unimodal 2
Bimodal
0
20
40
60
80
100
Data values
b
Bar plot
s.d.
s.e.m.
Box plot
Violin plot
Bean plot
Data values
Uniform
Unimodal
Unimodal 2
Bimodal

Thank you for your time
and
See you at the next lecture

Any other
questions & comments

l.kozlowski@mimuw.edu.pl