Visualization

Types of visualizations

- Data/information visualization (“charts”)
- Scientific visualization
- Scientific illustrations / cartoons (can use in combination with programming but not necessarily)

- Static
- Dynamic
  - animation – GIF, movies
  - graphical user interface (GUI) / interactive plots

When to use which?

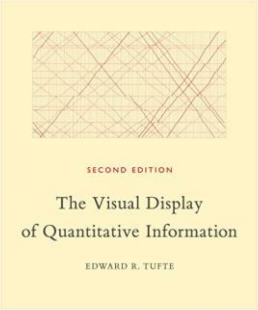
- Who is the audience?
  - Internal (project team) / external (client)

- How much time does it take the audience to digest the information?
  - A sequence of snapshots may be preferred over an animation or movie

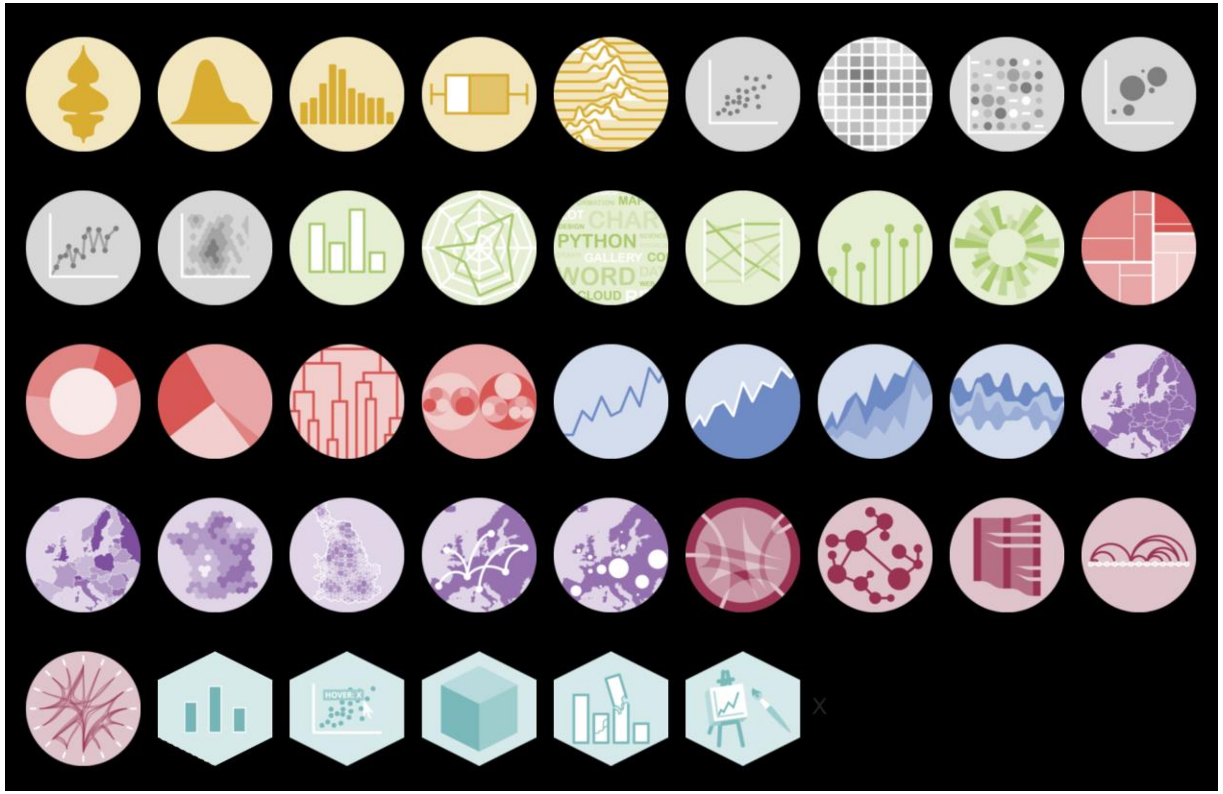
- Is an interactive plot necessary?
  - if the parameter dimensionality/space to explore is large and nonlinear
  - does the client need an interactive plotting tool or can/should *you* summarize the results

- Do aesthetic enhancements obfuscate the important trends?

Charts



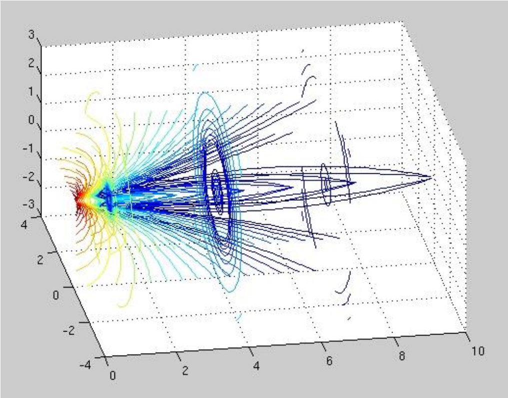
- Summarize data
- Can be generated programmatically in many data analysis languages
- Common method of visualization
- Use only as much “ink” as necessary

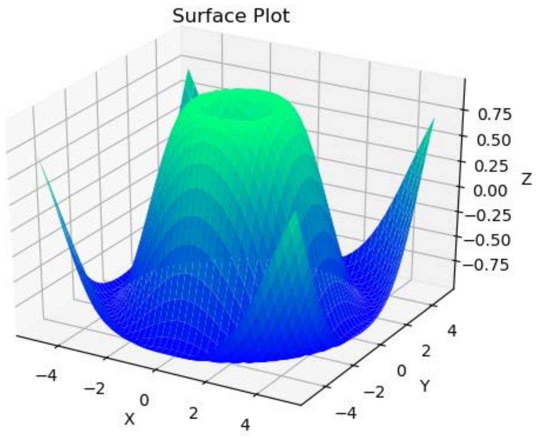


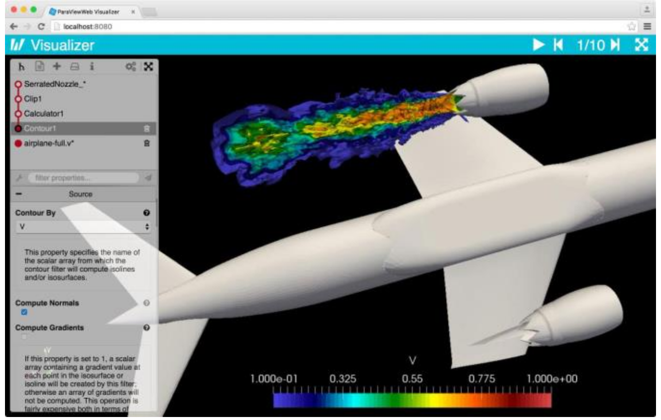
https://r-graph-gallery.com/

Scientific visualization

- Visualize (vector) fields
- Often (but not always) dealing with physical representations
- May involve rendering (adding realistic reflections, shading, etc.)
- Can do some in Python/MATLAB
- Often requires specialized tools
  - OpenGL
  - VTK
  - MayaVi
  - Blender
  - Rhinocerous 3D

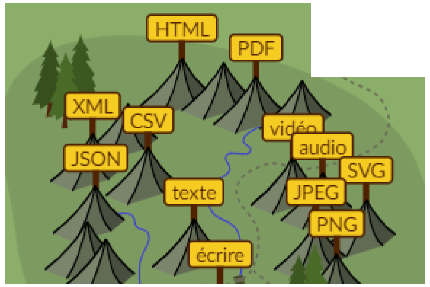






https://www.bu.edu/tech/support/research/training-consulting/online-tutorials/visualization-with-matlab/
https://www.cdslab.org/python/notes/visualization/3d/surface3D.png
https://vtk.org/flavors

Scientific illustrations / cartoons

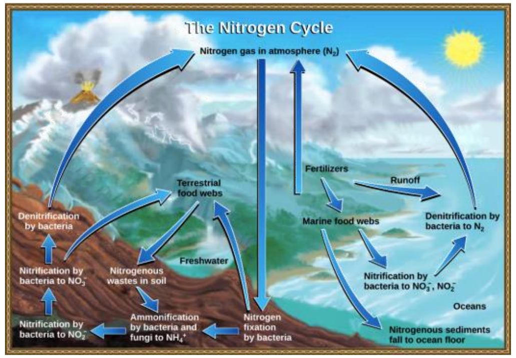


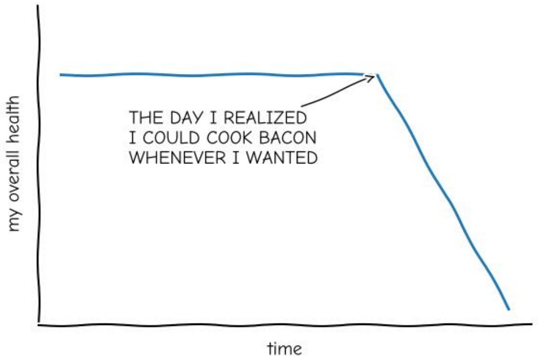
- By hand (traditional medium)

- **Raster graphics** (jpg, gif, png)
  → Adobe Photoshop / GIMP
  → efficient storage for images

- **Vector graphics** (pdf, svg)
  → Adobe Illustrator / Inkscape
  → small files for simple plots (points and lines), large files for complex plots (many points or images)

- Cartoons – useful for showing imprecise of the data



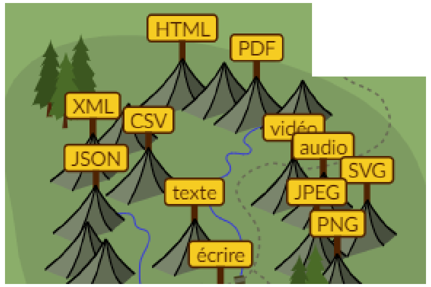


"Stove Ownership" from xkcd by Randall Munroe

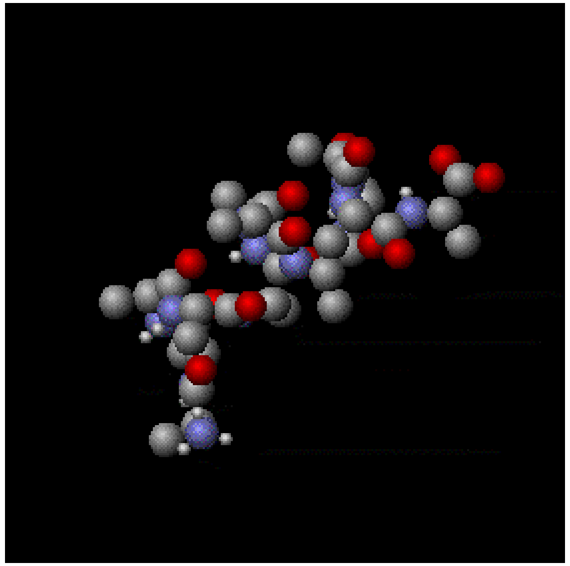
https://www.khanacademy.org/science/biology/ecology/biogeochemical-cycles/a/the-nitrogen-cycle

https://matplotlib.org/stable/gallery/showcase/xkcd.html

Animation



- GIFs: sequence of images
  - runs on any machine
- Movies: sequence of images stored as “frames”, can have accompanying audio track
  - more compact than GIFs
  - compressed with codec – may require OS/platform-specific codec to decode – test on different machines
- Can make with Python, MATLAB, and other tools



https://commons.wikimedia.org/wiki/File:Thermally_Agitated_Molecule.gif

User interfaces

- Command line interface (terminal)
  - run program (with input files)
  - run program with input arguments
  - prompt user input from keyboard

- Graphical user interface (GUI)
  - toolkits – e.g., Tkinter, wxPython, PyQt
  - dashboards (web application) – e.g., Dash, Streamlit, Shiny
  - web application on web framework – e.g., Flask/Django with client-side (front end)
  - app – e.g., Dart

Use cases

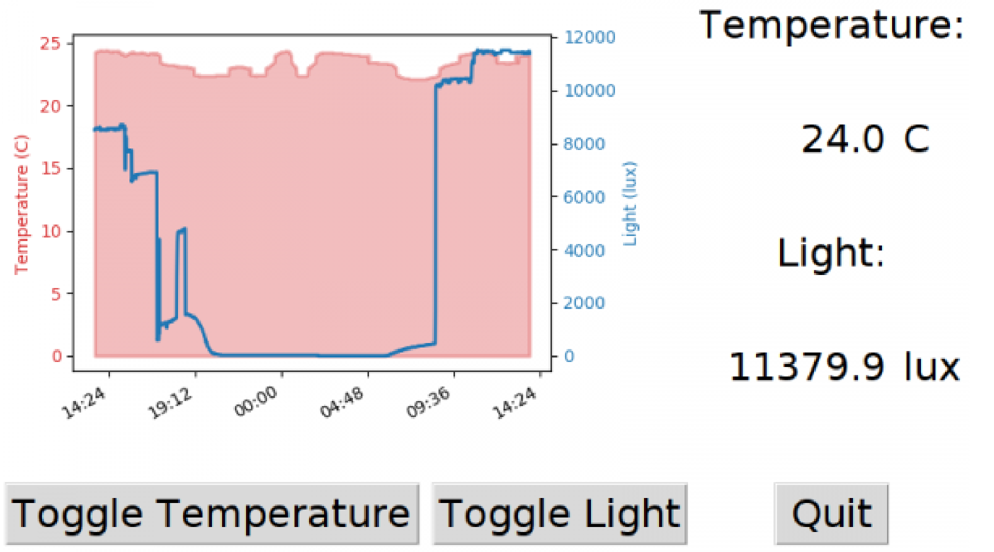
- command line
  - programmable and reproducible
  - when prompting for user input from the keyboard, note that input must be saved to be reproducible

- GUIs
  - interactive
  - exploratory

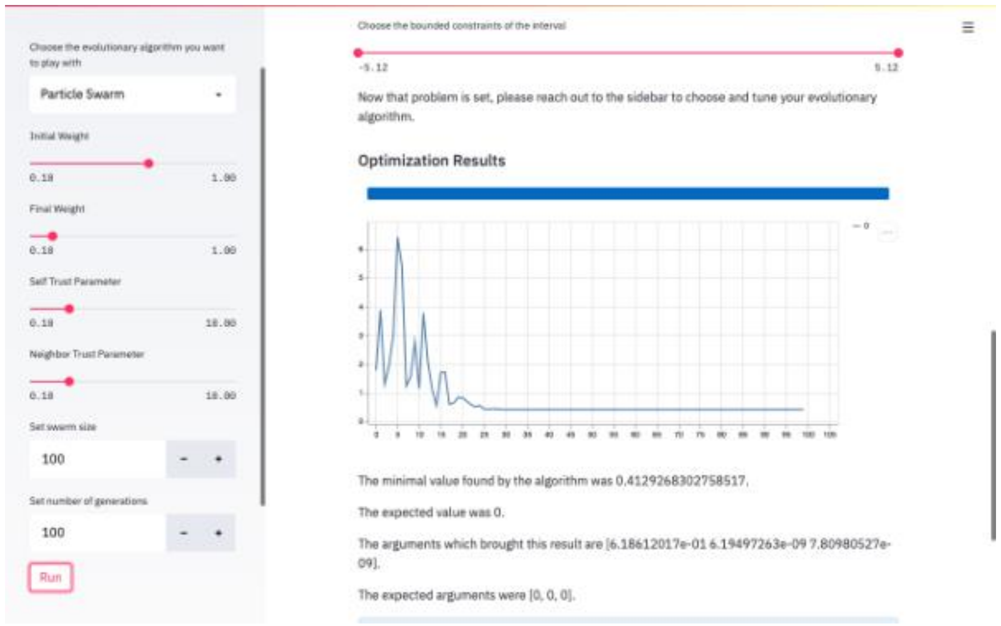
Not part of ENG-270 project!

Graphical User Interfaces (GUIs)

- Drive program execution through graphical elements rather than command line arguments or file input parameters
- Uses – “Human Intelligence Tasks”
  - interactive plots – high-dimensional data exploration
  - data labeling
- Implementation
  - desktop/mobile app
  - web app
- Some GUI elements are built into interactive plots (MATLAB, matplotlib, etc.)
  - scatterplot brushing
  - spanning



https://learn.sparkfun.com/tutorials/python-gui-guide-introduction-to-tkinter/all



https://streamlit.io/

Not part of ENG-270 project!

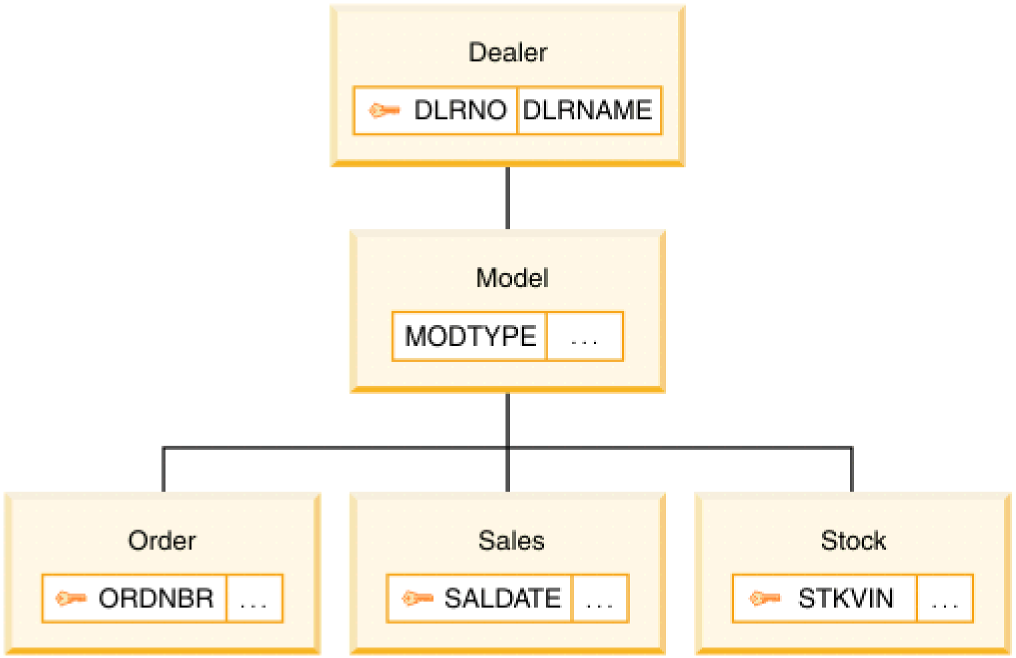
Alternatives to user interfaces (Recommended)

- For each scenario, save input parameters to file – e.g., JSON, CSV
- Code loops through each scenario and generates outputs saved to different files / folders
- Another code harvests results and constructs summary tables and plots

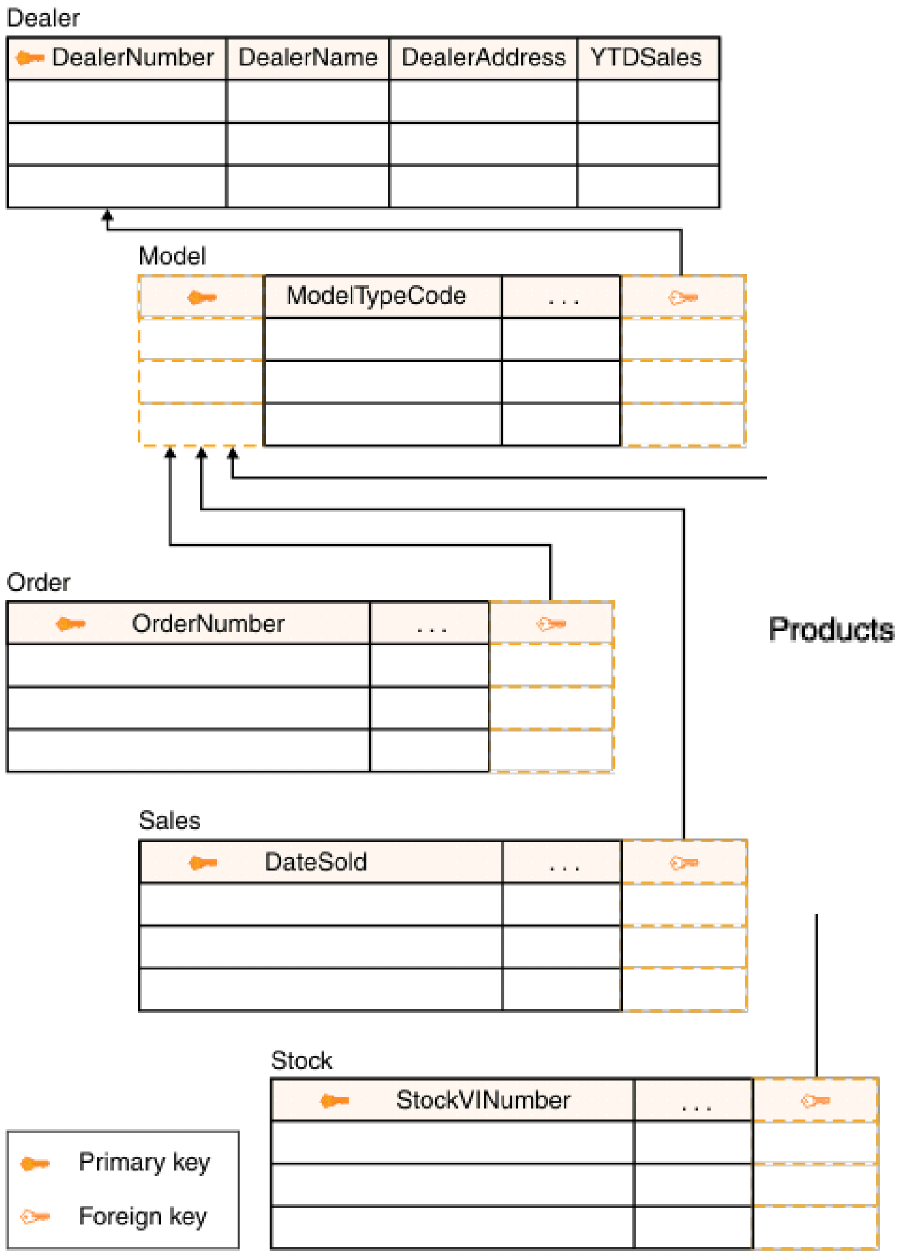
Working with relational tables and statistical visualizations

[Web link]

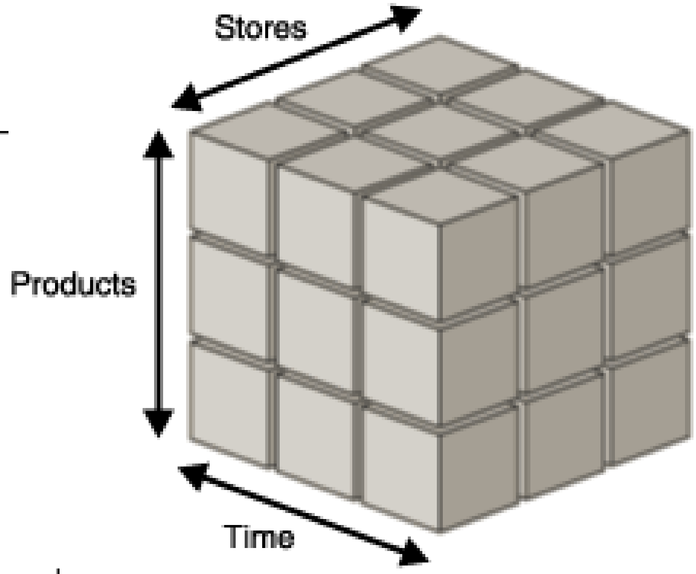
Data models



Dealer
DLRNO
DLRNAME
Model
MODTYPE
...
Order
ORDNBR
...
Sales
SALDATE
...
Stock
STKVIN
...

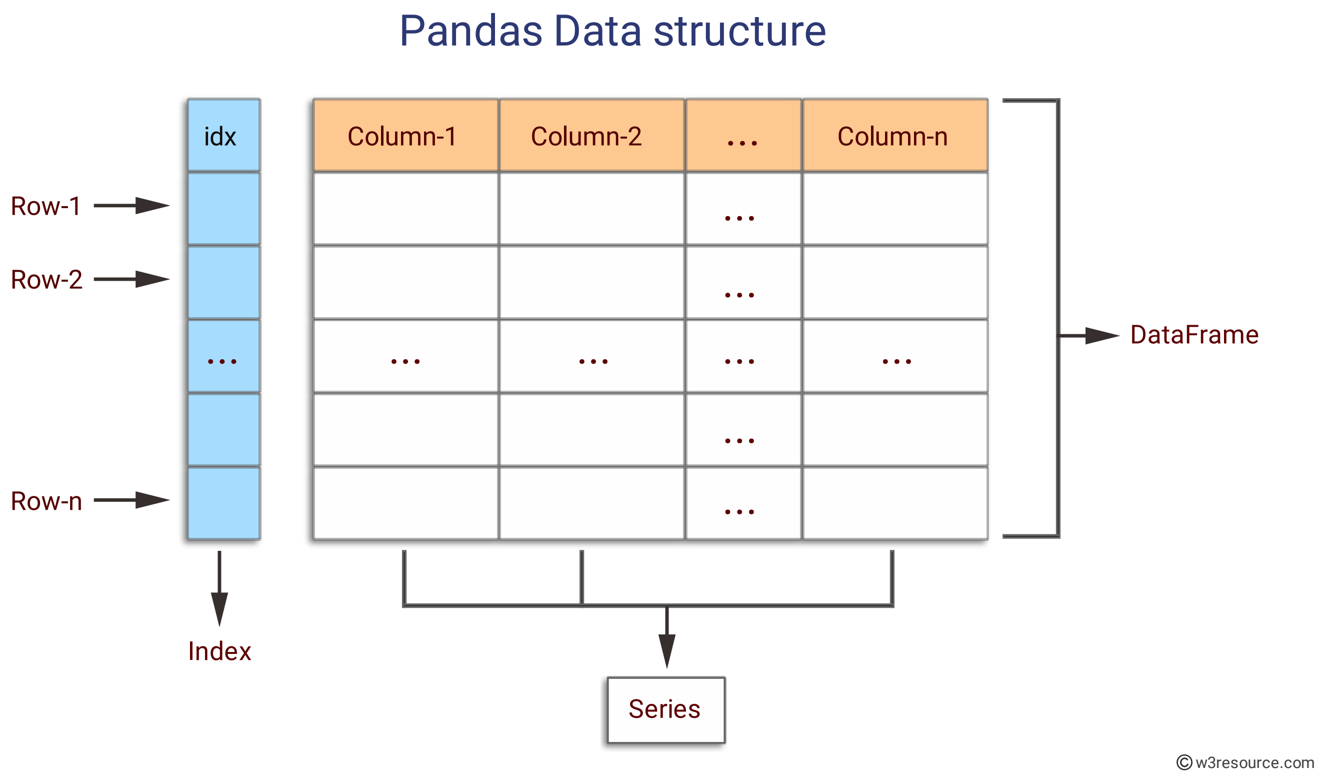


Dealer
DealerNumber
DealerName
DealerAddress
YTDSales
Model
ModelTypeCode
...
Order
OrderNumber
...
Sales
DateSold
...
Stock
StockVINumber
...
Primary key
Foreign key

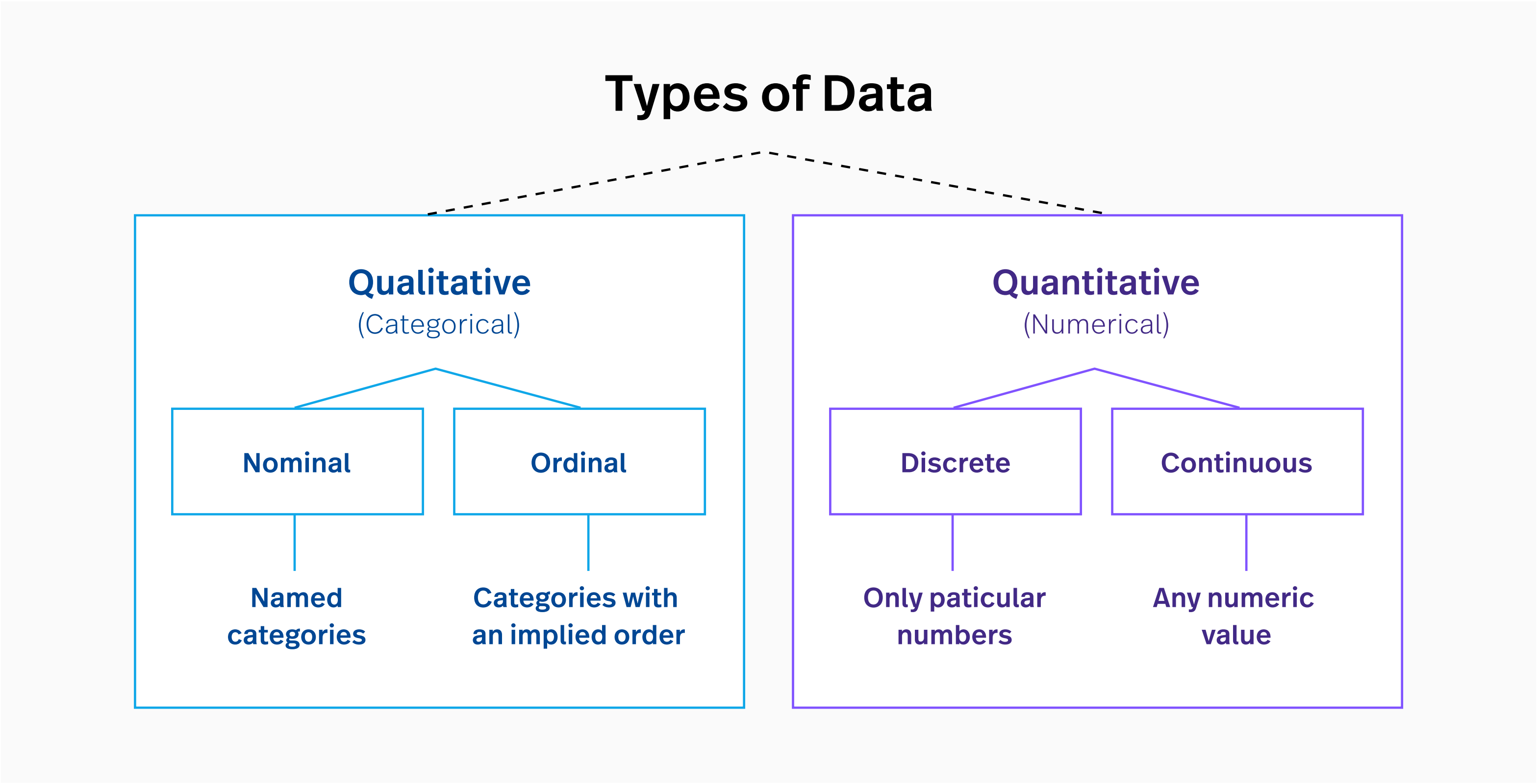


Stores
Products
Time

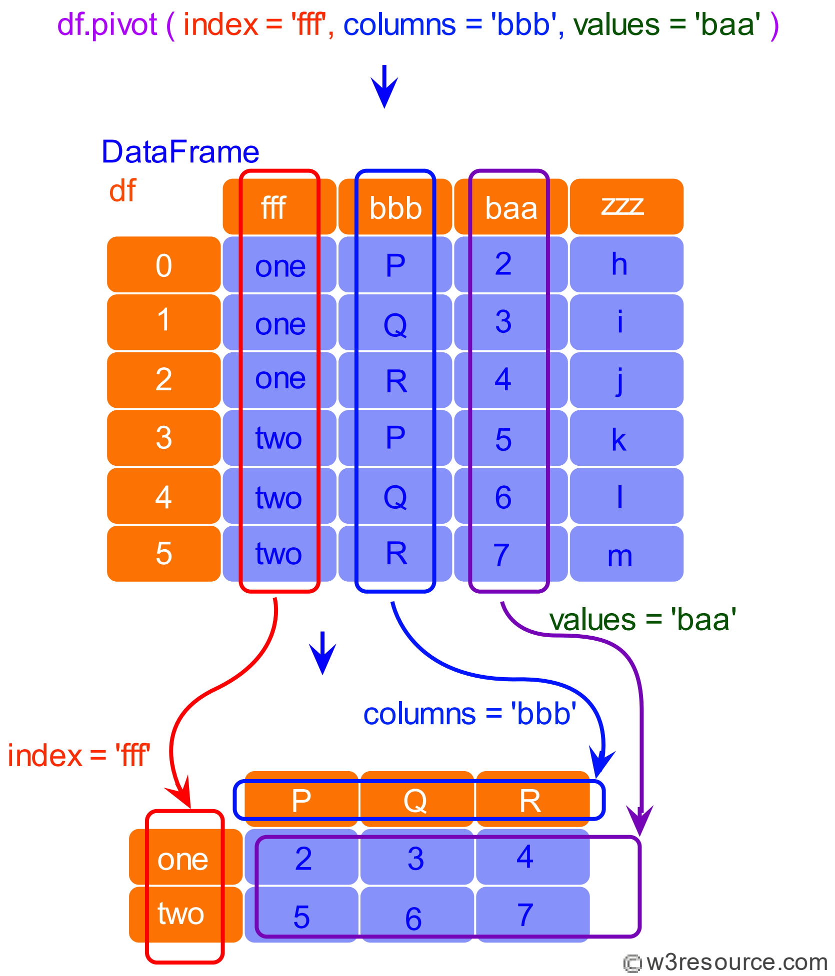
DataFrame in Python pandas



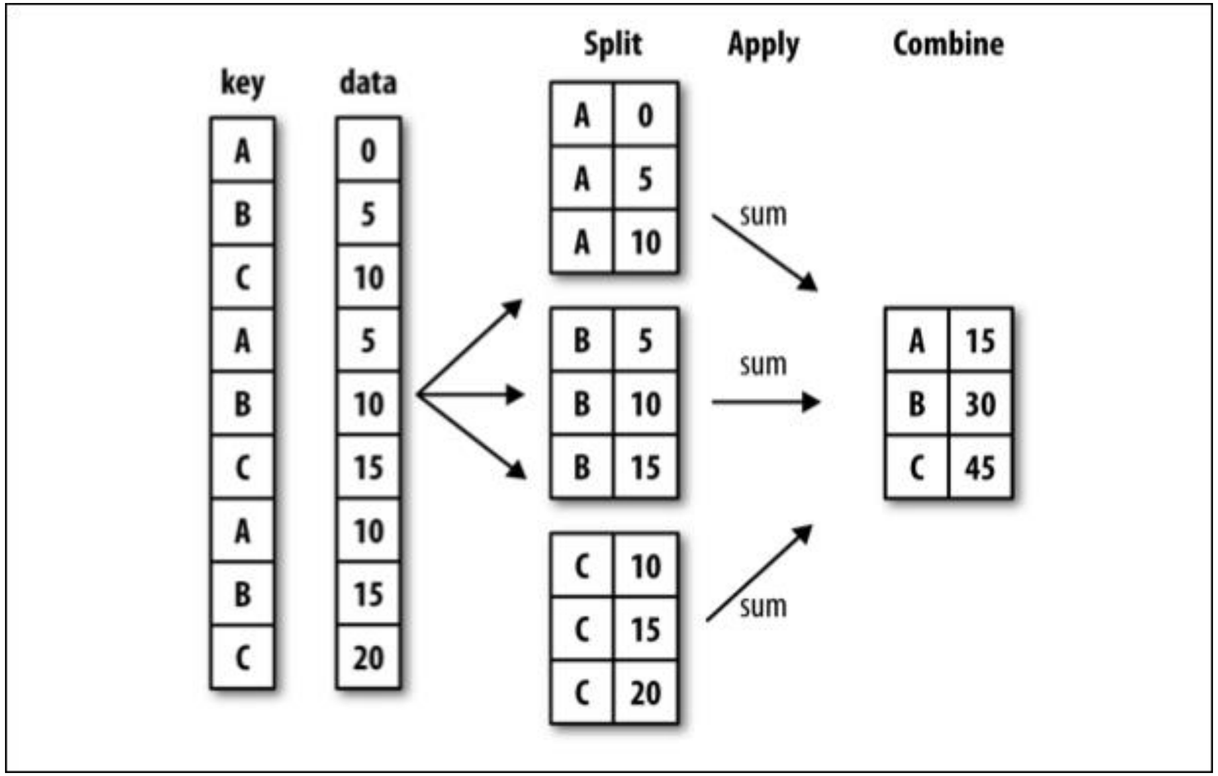
New data type: categorical variables



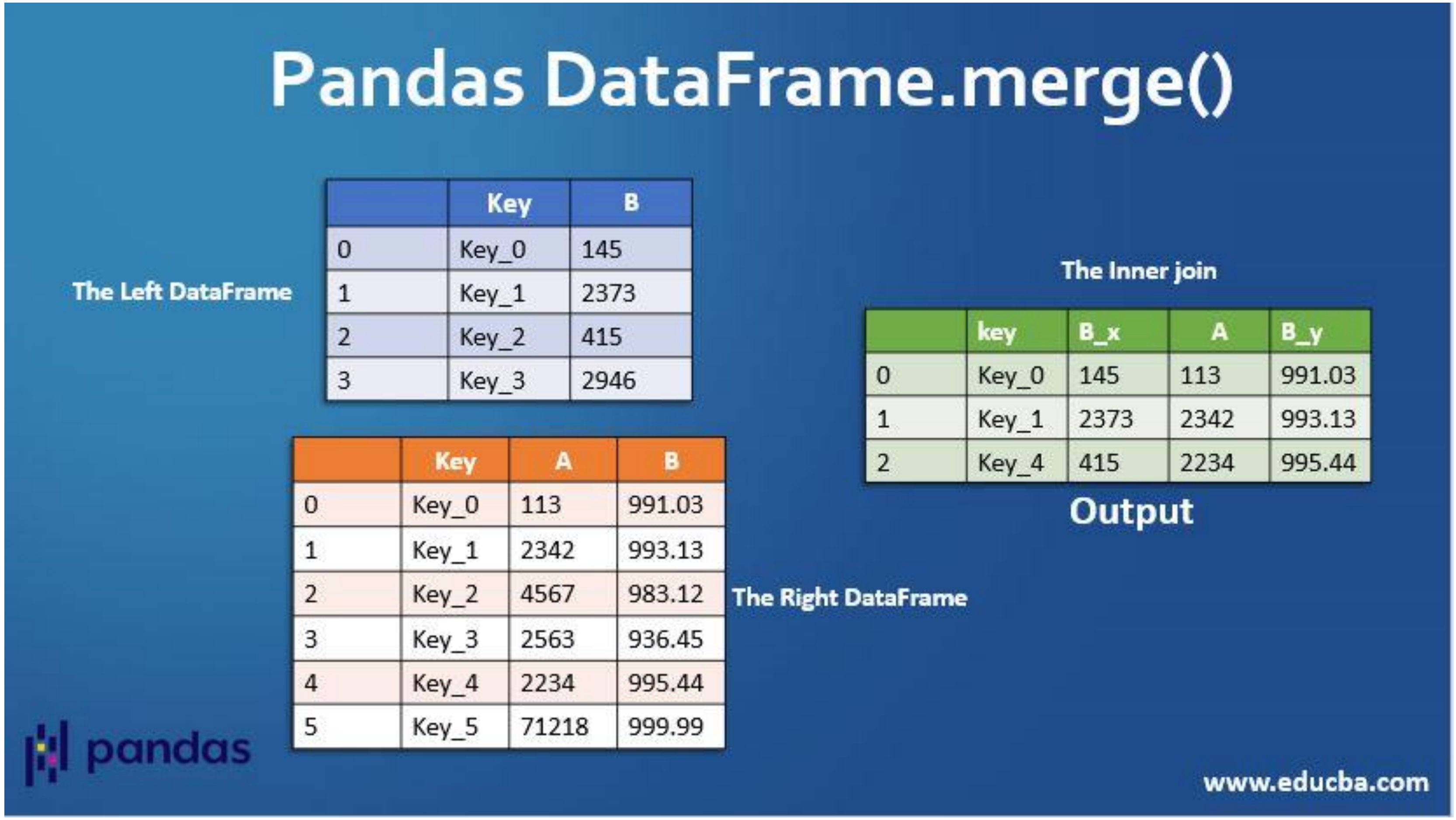
Pivoting / reshaping



Split-Apply-Combine approach



Merging

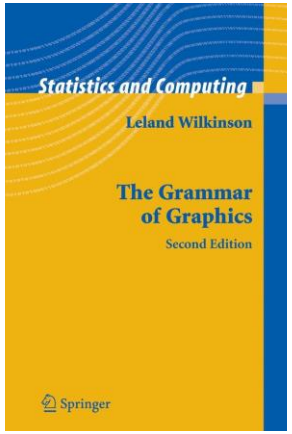


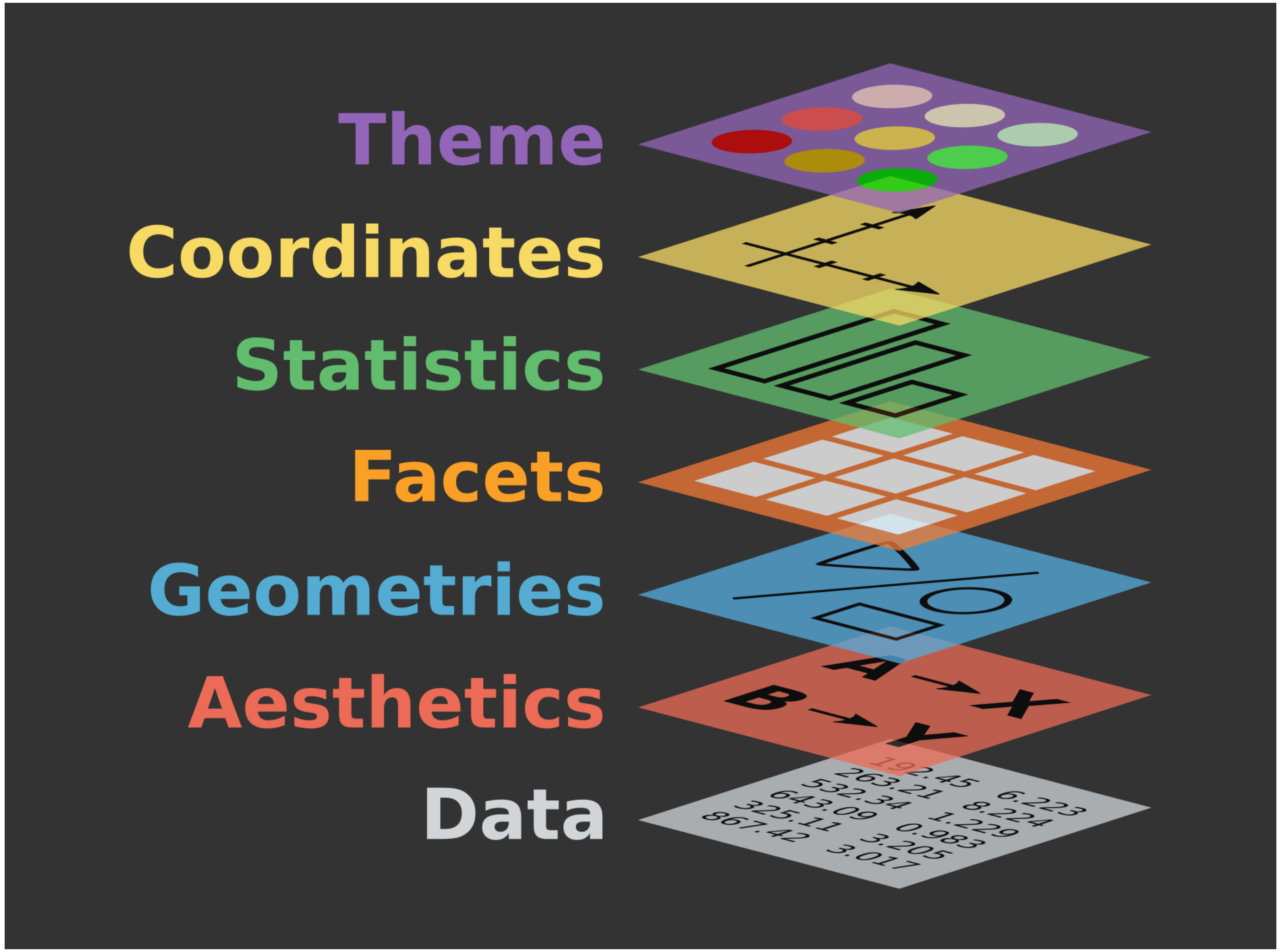
	Key	B
0	Key_0	145
1	Key_1	2373
2	Key_2	415
3	Key_3	2946

	Key	A	B
0	Key_0	113	991.03
1	Key_1	2342	993.13
2	Key_2	4567	983.12
3	Key_3	2563	936.45
4	Key_4	2234	995.44
5	Key_5	71218	999.99

	key	B_x	A	B_y
0	Key_0	145	113	991.03
1	Key_1	2373	2342	993.13
2	Key_4	415	2234	995.44

“Grammar of Graphics”





Data downloaded from NABEL network

```
Station: Lausanne-César-Roux
Urban, traffic
Daily means, O3: Maximum hourly mean of the day, PREC: Daily sums
Source: NABEL/MeteoSchweiz
Date/time;O3 [ug/m3];NO2 [ug/m3];CO [mg/m3];PM10 [ug/m3];PM2.5 [ug/m3];NOX [ug/m3 eq. NO2];TEMP [C];PREC [mm];RAD [W/m2]
01.01.2021;55.3;19.9;0.27;6.2;4.9;27.6;1.6;11.9;27.1
02.01.2021;45.1;10.4;0.23;12.8;10.5;13.0;1.3;0.0;37.1
03.01.2021;44.1;16.5;0.28;14.5;11.7;21.9;0.7;0.0;20.2
04.01.2021;31.3;18.8;0.29;19.5;15.6;25.2;0.3;0.0;25.7
05.01.2021;37.8;23.1;0.28;20.9;16.3;31.5;0.6;0.0;14.9
06.01.2021;37.4;28.1;0.32;17.2;12.9;41.1;0.3;0.0;24.3
07.01.2021;39.5;27.3;0.34;18.5;14.1;39.5;1.0;0.0;55.2
08.01.2021;48.6;25.0;0.31;14.7;10.4;36.3;0.4;0.0;81.0
09.01.2021;51.3;15.3;0.26;16.0;12.5;18.8;-1.6;0.0;62.6
10.01.2021;48.0;15.9;0.26;19.5;15.3;20.1;-1.7;0.0;82.6
11.01.2021;47.8;28.9;0.32;28.2;21.9;44.5;-2.8;0.0;66.9
12.01.2021;44.9;42.2;0.79;26.3;20.5;83.3;-0.9;11.4;13.4
13.01.2021;49.4;39.1;0.32;7.5;5.5;65.4;3.0;9.2;9.6
14.01.2021;50.9;31.2;0.27;5.4;4.0;45.9;4.3;5.6;10.6
15.01.2021;58.5;23.8;0.26;5.4;4.0;36.6;2.3;7.0;13.3
16.01.2021;51.2;27.4;0.30;15.7;12.5;43.0;-1.5;1.6;37.2
17.01.2021;50.3;32.3;0.31;9.0;7.4;46.5;1.1;7.9;16.0
18.01.2021;49.6;43.4;0.35;10.4;7.9;72.3;1.9;0.0;69.6
19.01.2021;60.6;43.5;0.35;12.2;7.8;71.7;1.6;0.0;92.3
20.01.2021;55.0;52.3;0.37;13.1;5.0;93.5;5.2;0.0;53.3
21.01.2021;82.6;39.1;0.31;7.4;4.1;65.1;8.3;0.6;17.6
22.01.2021;79.5;34.7;0.31;6.1;3.6;55.7;6.9;14.7;45.1
23.01.2021;70.1;22.3;0.24;4.6;2.9;36.3;3.0;4.7;46.3
24.01.2021;70.3;19.9;0.24;5.1;2.9;31.0;2.6;2.6;98.3
25.01.2021;69.7;29.8;0.25;5.6;3.1;49.4;0.9;6.2;48.7
26.01.2021;61.4;29.6;0.25;11.4;5.9;42.7;0.5;0.0;90.8
```

First steps

- Create a new table where the date/time information is usable

- Define categorical variables:
  - month
  - day of week
  - weekday/weekend

Preparing the new Data Frame

```
import numpy as np
import pandas as pd
import calendar
```

Read data

```
data = pd.read_table('./data/LAU_diurnal.csv', delimiter=';', encoding='latin-1', skiprows=4, header=0)
```

Rename data columns

```
data.columns = [x.split()[0] for x in data.columns]
```

```
Index(['Date/time', 'O3 [ug/m3]', 'NO2 [ug/m3]', 'CO [mg/m3]', 'PM10 [ug/m3]',
       'PM2.5 [ug/m3]', 'NOX [ug/m3 eq. NO2]', 'TEMP [C]', 'PREC [mm]',
       'RAD [W/m2]'],
      dtype='object')
```

```
Index(['Date/time', 'O3', 'NO2', 'CO', 'PM10', 'PM2.5', 'NOX', 'TEMP', 'PREC',
       'RAD'],
      dtype='object')
```

Parse date/time information

```
def parsedate(datestr):
    daystr, monthstr, yearstr = datestr.split('.')
    return {'day':int(daystr), 'month':int(monthstr),'year': int(yearstr)}

def month2season(months):
    season = {0:'DJF', 1:'MAM', 2:'JJA', 3:'SON'}
    lookup = np.vectorize(season.get)
    breaks = np.arange(3, 13, 3)
    return lookup(np.mod(np.digitize(months, breaks), 4))

def daycategory(day_name):
    days = list(calendar.day_name)
    return pd.Categorical(day_name, categories = days, ordered=True)

def daytype(day_num):
    types = ['Weekday', 'Weekend']
    return pd.Categorical(np.where(day_num < 5, 'Weekday', 'Weekend'), categories = types, ordered=True)
```

```
dates = pd.DataFrame([parsedate(x) for x in data['Date/time']])
dateobj = pd.to_datetime(dates)
dates['season'] = month2season(dates['month'])
dates['dayofweek'] = daycategory(dateobj.dt.day_name())
dates['daytype'] = daytype(dateobj.dt.dayofweek)
dates['month'] = pd.Categorical(dates['month'])
```

```
     day month  year season  dayofweek  daytype
0      1     1  2021    DJF     Friday  Weekday
1      2     1  2021    DJF   Saturday  Weekend
2      3     1  2021    DJF     Sunday  Weekend
3      4     1  2021    DJF     Monday  Weekday
4      5     1  2021    DJF    Tuesday  Weekday
..   ...   ...   ...    ...        ...      ...
360   27    12  2021    DJF     Monday  Weekday
361   28    12  2021    DJF    Tuesday  Weekday
362   29    12  2021    DJF  Wednesday  Weekday
363   30    12  2021    DJF   Thursday  Weekday
364   31    12  2021    DJF     Friday  Weekday

[365 rows x 6 columns]
```

Add new date information to original data frame

Merge using concat()

```
merged = pd.concat([
    pd.DataFrame(dateobj, columns=['datetime']),
    dates,
    data.iloc[:,1:] # take all after Date/time column
], axis=1)
```

	datetime	day	month	year	season	...	PM2.5	NOX	TEMP	PREC	RAD
0	2021-01-01	1	1	2021	DJF	...	4.9	27.6	1.6	11.9	27.1
1	2021-01-02	2	1	2021	DJF	...	10.5	13.0	1.3	0.0	37.1
2	2021-01-03	3	1	2021	DJF	...	11.7	21.9	0.7	0.0	20.2
3	2021-01-04	4	1	2021	DJF	...	15.6	25.2	0.3	0.0	25.7
4	2021-01-05	5	1	2021	DJF	...	16.3	31.5	0.6	0.0	14.9
..	...	...	...	...	...	...	...	...	...	...	...
360	2021-12-27	27	12	2021	DJF	...	7.9	53.3	5.6	6.5	8.7
361	2021-12-28	28	12	2021	DJF	...	3.8	37.4	7.5	14.3	6.3
362	2021-12-29	29	12	2021	DJF	...	4.0	35.6	7.4	15.0	5.2
363	2021-12-30	30	12	2021	DJF	...	8.5	75.9	10.7	0.0	48.7
364	2021-12-31	31	12	2021	DJF	...	9.0	62.0	9.2	0.0	69.8

[365 rows x 16 columns]

Pivot using melt()

```
lf = merged.melt(id_vars=['datetime'] + dates.columns.to_list())
```

	datetime	day	month	year	season	dayofweek	daytype	variable	value
0	2021-01-01	1	1	2021	DJF	Friday	Weekday	O3	55.3
1	2021-01-02	2	1	2021	DJF	Saturday	Weekend	O3	45.1
2	2021-01-03	3	1	2021	DJF	Sunday	Weekend	O3	44.1
3	2021-01-04	4	1	2021	DJF	Monday	Weekday	O3	31.3
4	2021-01-05	5	1	2021	DJF	Tuesday	Weekday	O3	37.8
...	...	...	...	...	...	...	...	...	...
3280	2021-12-27	27	12	2021	DJF	Monday	Weekday	RAD	8.7
3281	2021-12-28	28	12	2021	DJF	Tuesday	Weekday	RAD	6.3
3282	2021-12-29	29	12	2021	DJF	Wednesday	Weekday	RAD	5.2
3283	2021-12-30	30	12	2021	DJF	Thursday	Weekday	RAD	48.7
3284	2021-12-31	31	12	2021	DJF	Friday	Weekday	RAD	69.8

[3285 rows x 9 columns]

Some analyses

- Missing values by month
- Monthly exceedances of $PM_{2.5}$
- Monthly means
- Mean values by day of week

Missing values (how many are missing?)

```
missing = (
    lf
    .groupby(['month', 'variable'])
    .apply(lambda x: x['value'].isnull().sum())
)
```

```
print(missing)
```

```
month  variable
1      CO          0
       NO2         0
       NOX         0
       O3          0
       PM10        0
                  ..
12     PM10        0
       PM2.5       0
       PREC        0
       RAD         0
       TEMP        0
Length: 108, dtype: int64
```

```
print(missing.unstack())
```

```
variable  CO  NO2  NOX  O3  PM10  PM2.5  PREC  RAD  TEMP
month
1          0    0    0   0     0      0     0    0     0
2          0    0    0   0     1      1     3    0     0
3          0    0    0   0     0      0     0    0     0
4          0    0    0   1     0      0     0    0     0
5          0    1    1   1     0      0     0    0     0
6          0    1    1   1     6      6     0    0     0
7          0    0    0   0     0      0     0    0     0
8          0    0    0   0     0      0     0    0     0
9          0    0    0   0     0      0     0    0     0
10         0    0    0   0     0      0     0    0     0
11         0    0    0   0     1      1     1    0     0
12         0    0    0   0     0      0     0    0     0
```

Monthly exceedances of PM2.5

```
LIMIT_PM25 = 15
exceedances = (
    lf
    .loc[lf['variable'] == 'PM2.5']
    .groupby(['month'])
    .apply(lambda x: (x['value'] > LIMIT_PM25).sum())
)
```

```
print(exceedances)
```

```
month
1       5
2      12
3       8
4       0
5       0
6       0
7       2
8       1
9       0
10      3
11      8
12      8
dtype: int64
```

Monthly means and daily averages

```
means_monthly = (
    lf.groupby(['month', 'variable'])
    .apply(lambda x: x['value'].mean())
)
```

```
print(means_monthly.unstack())
```

```
variable        CO        NO2  ...         RAD       TEMP
month                          ...
1         0.307097  28.593548  ...   41.470968   2.112903
2         0.356071  34.732143  ...   88.896429   6.185714
3         0.303871  29.658065  ...  156.877419   7.245161
4         0.265333  23.076667  ...  226.886667   9.873333
5         0.261613  23.143333  ...  213.167742  12.419355
6         0.276333  23.300000  ...  260.486667  20.330000
7         0.284516  19.887097  ...  226.425806  19.983871
8         0.293871  17.687097  ...  219.406452  19.441935
9         0.321333  25.390000  ...  157.630000  18.130000
10        0.309032  25.654839  ...  110.690323  11.587097
11        0.305000  24.613333  ...   46.076667   5.840000
12        0.287419  33.167742  ...   29.341935   3.290323

[12 rows x 9 columns]
```

```
means_dailyJJA = (
    lf.loc[lf['season'] == 'JJA']
    .groupby(['dayofweek', 'variable'])
    .apply(lambda x: x['value'].mean())
    )
```

```
print(means_dailyJJA.unstack())
```

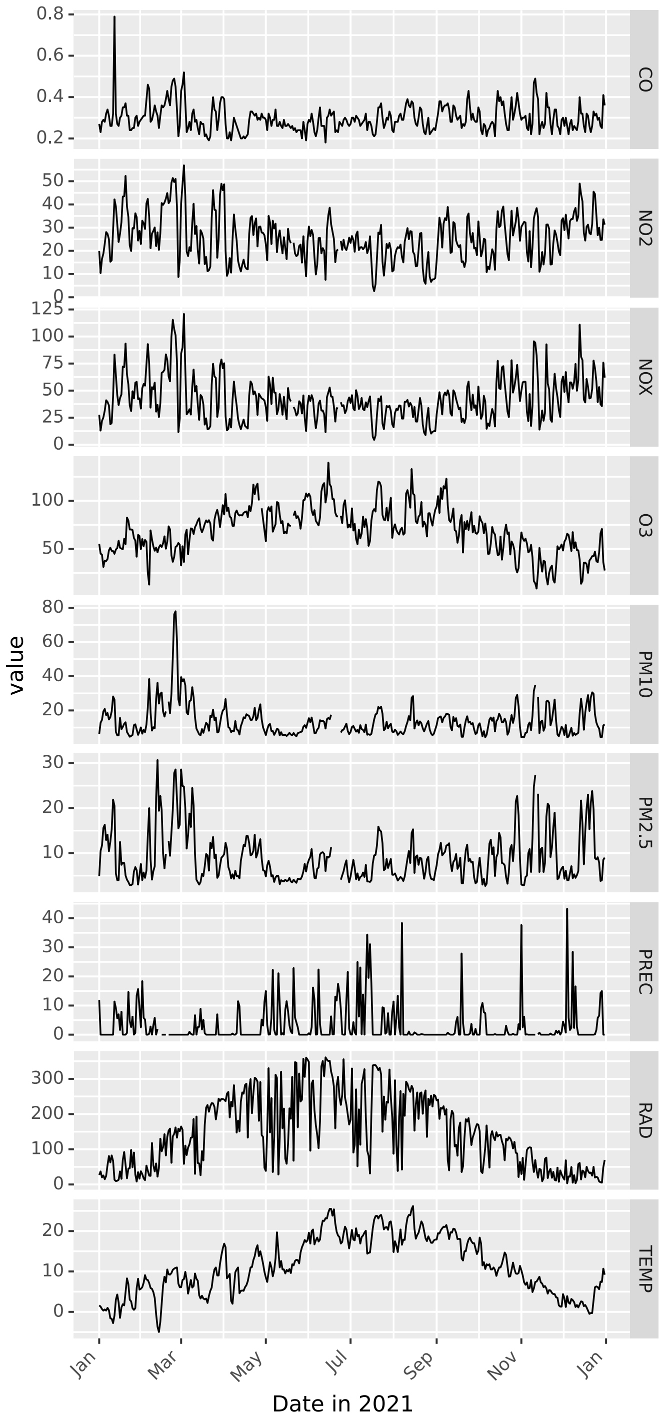
```
variable         CO        NO2  ...         RAD       TEMP
dayofweek                       ...
Monday     0.275385  17.730769  ...  241.392308  20.046154
Tuesday    0.278571  21.214286  ...  226.364286  19.150000
Wednesday  0.300769  23.791667  ...  201.284615  18.938462
Thursday   0.301538  23.930769  ...  234.476923  19.923077
Friday     0.297692  23.038462  ...  276.784615  20.784615
Saturday   0.283846  18.761538  ...  235.546154  20.453846
Sunday     0.257692  13.307692  ...  231.000000  20.161538

[7 rows x 9 columns]
```

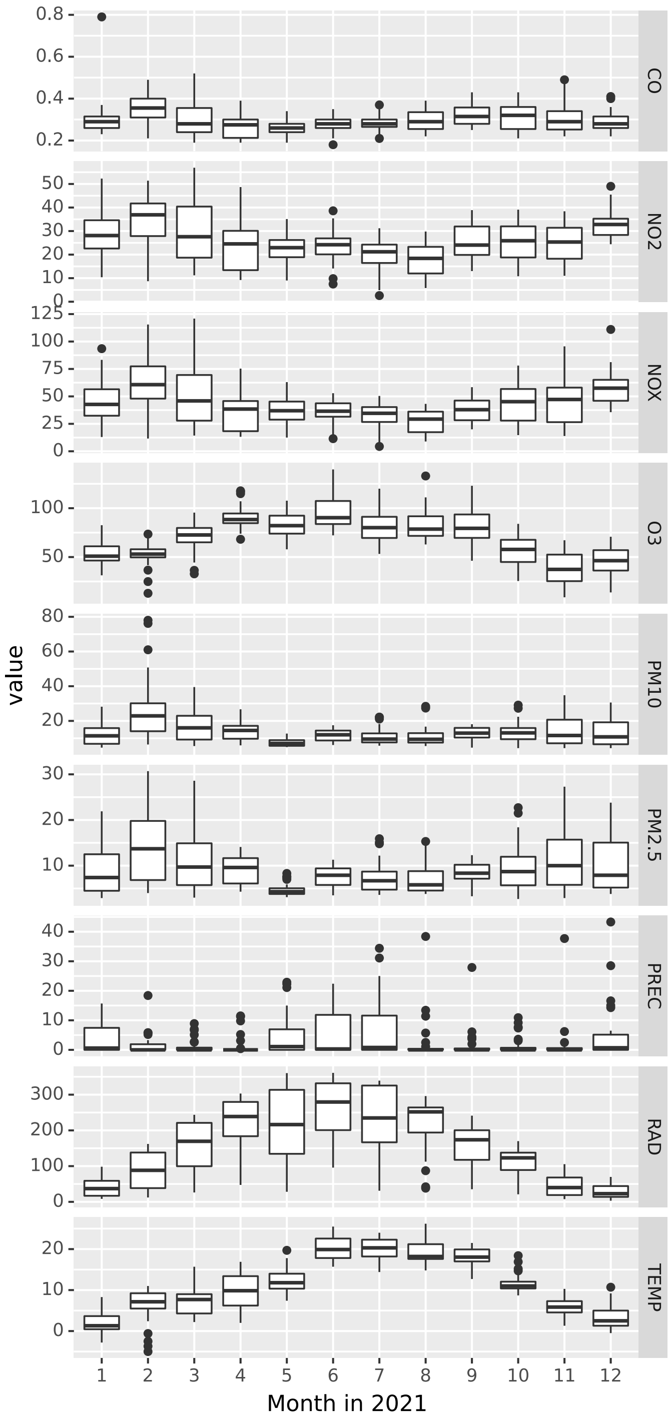
Plotnine package
Grammar of Graphics

```python
import plotnine as pn
from mizani.formatters import date_format
```

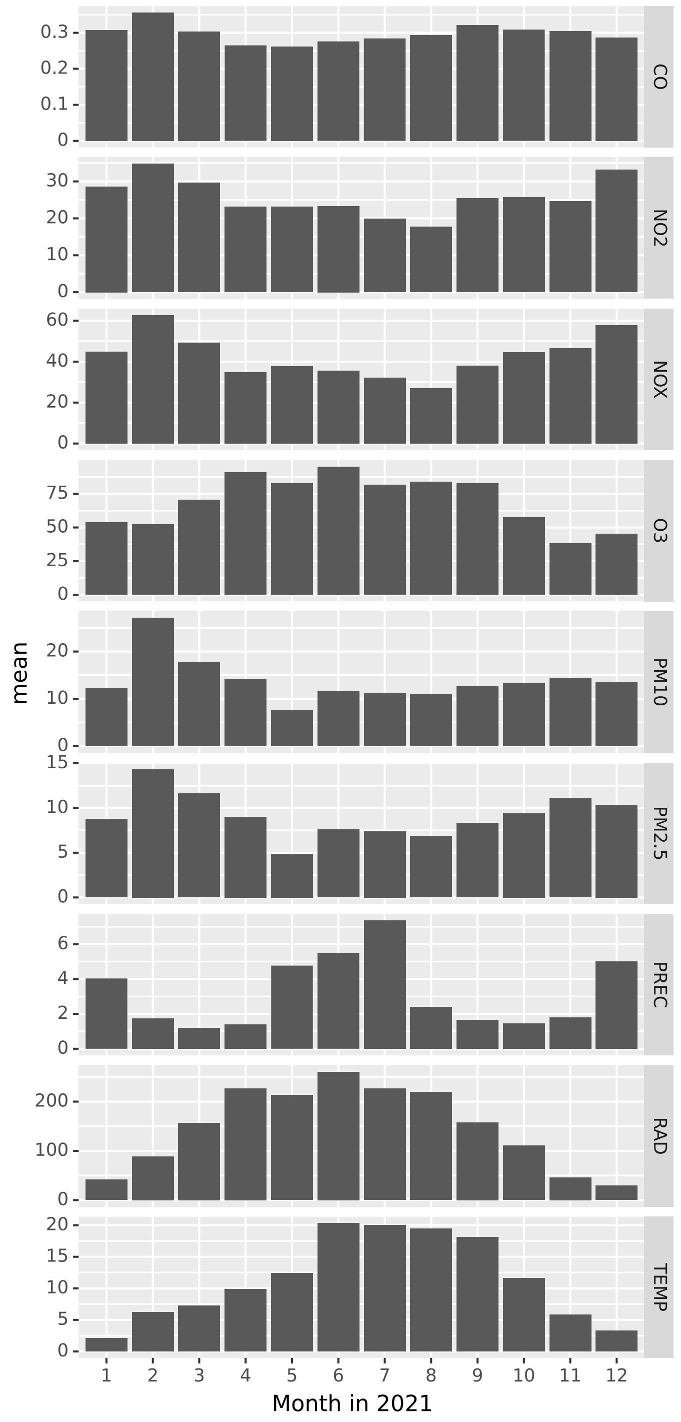
```python
p = (
    pn.ggplot(data=lf)
    + pn.geom_line(mapping=pn.aes(x='datetime', y='value'))
    + pn.facet_grid('variable ~ .', scales='free_y')
    + pn.scale_x_datetime(name = 'Date in 2021', labels = date_format('%b'))
    + pn.theme(axis_text_x = pn.element_text(angle=45, hjust=1))
    )
```



```
p = (
    pn.ggplot(data=lf)
    + pn.geom_boxplot(mapping=pn.aes(x='month', y='value'))
    + pn.facet_grid('variable ~ .', scales='free_y')
    + pn.scale_x_discrete(name = 'Month in 2021')
    )
```

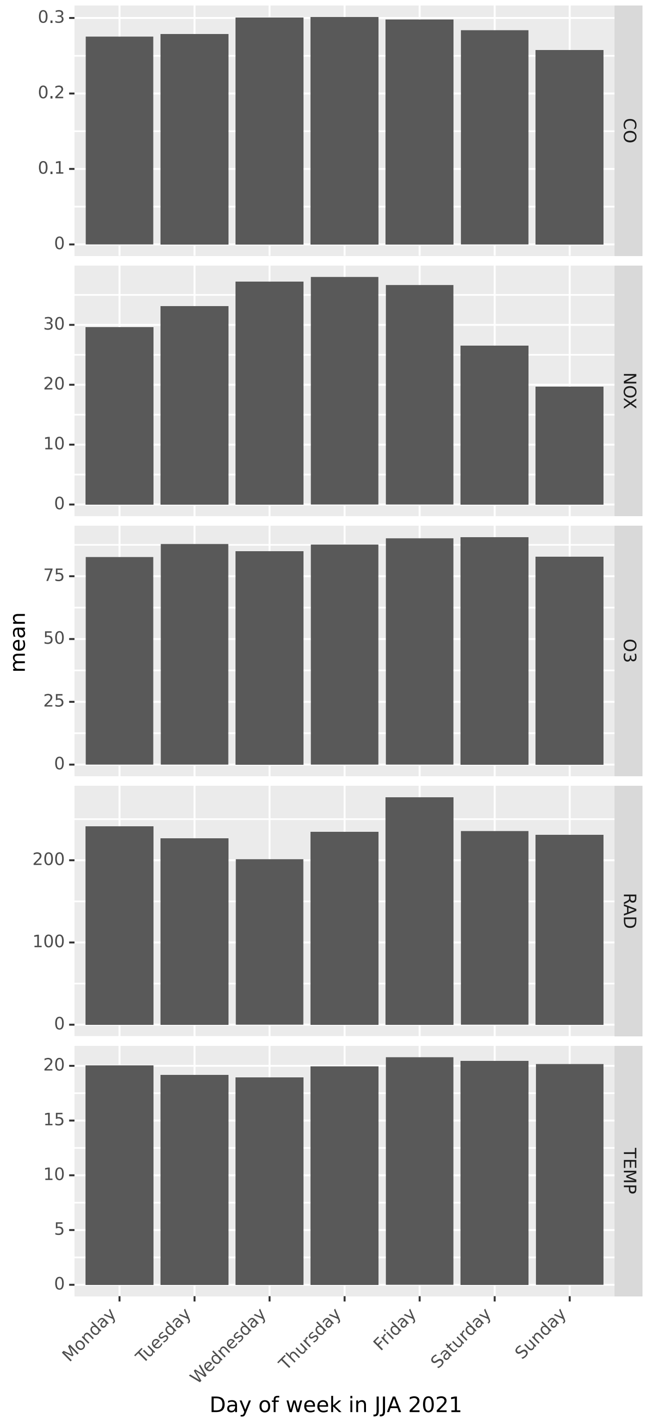


```
p = (
    pn.ggplot(data=means_monthly.reset_index(name = 'mean'))
    + pn.geom_col(mapping=pn.aes(x='month', y='mean'))
    + pn.scale_x_discrete(name = 'Month in 2021', breaks = np.arange(1, 13))
    + pn.facet_grid('variable ~ .', scales='free_y')
    )
```



```
wf = means_dailyJJA.reset_index(name = 'mean')
means_dailyJJA_subset = wf.loc[wf['variable'].isin(['O3', 'CO', 'NOX', 'RAD', 'TEMP'])]

p = (
    pn.ggplot(data=means_dailyJJA_subset)
    + pn.geom_col(mapping=pn.aes(x='dayofweek', y='mean'))
    + pn.scale_x_discrete(name = 'Day of week in JJA 2021')
    + pn.facet_grid('variable ~ .', scales='free_y')
    + pn.theme(axis_text_x = pn.element_text(angle=45, hjust=1))
    )
```



Swiss terrain exercise revisited (pivoting)

Read xyz data

```
xyz = pd.read_table('./data/DHM200.xyz', delimiter=' ', header=None)
print(xyz.head())
```

```
          0         1       2
0  655000.0  302000.0  835.01
1  655200.0  302000.0  833.11
2  655400.0  302000.0  831.20
3  655600.0  302000.0  829.30
4  655800.0  302000.0  828.80
```

Pivot to wide format

```
wide = xyz.pivot_table(index=0, columns=1, values=2, fill_value = 0)
print(wide.head())
```

```
1         74000.0   74200.0   ...   301800.0  302000.0
0                             ...
480000.0      0.0       0.0   ...        0.0       0.0
480200.0      0.0       0.0   ...        0.0       0.0
480400.0      0.0       0.0   ...        0.0       0.0
480600.0      0.0       0.0   ...        0.0       0.0
480800.0      0.0       0.0   ...        0.0       0.0

[5 rows x 1141 columns]
```

Plot

```
plt.axes().set_aspect('equal')
plt.pcolormesh(wide.index, wide.columns, wide.to_numpy().T,
               shading = 'auto')
plt.title('Suisse')
```