LL7: Manipularea datelor folosind limbajul Python. Utilizarea librăriei Pandas pentru importarea și manipularea datelor.

Biblioteca pandas oferă funcționalități pentru manipularea facilă a datelor.

Vom analiza datele climatologice publicate de Guvernul României (https://data.gov.ro/dataset/date-climatologice-de-la-cele-23-de-statii-esentiale-pentru-anul-2016).

Documentația oficială pentru pandas poate fi gasită aici ((https://pandas.pydata.org/pandas-docs/stable/) - acesta este locul unde veți găsi răspuns la toate întrebările legate de această bibliotecă.

Formatul în care datele sunt colectate și stocate depinde foarte mult de tipul de date, sursa datelor, cantitatea de date și tipul de analiză pe care dorim să îl facem. Un format popular pentru acest tip de date este **CSV** (*Comma Separated Values*). Este doar un fișier text în care putem scrie un tabel ale cărui celulele sunt separate prin virgule, similar cu un tabel *Microsoft Excel*. Motivul pentru care acest format este atât popular este simplitatea lui: este doar un fișier text, ce poate fi prelucrat de orice editor de text. Alte formate pentru fișiere, precum *.xls*, folosit de Excel, au nevoie de programe specializate, ce adesea necesită licențe scumpe.

Un exemplu de fișier CSV este:

```
CODST,ALT,LAT,LON,DATCLIM,TMED,TMAX,TMIN,R24
15015,503.00,47.7769444,23.9405556,2016/01/01 00, -10.0,  -5.7, -13.6,
15015,503.00,47.7769444,23.9405556,2016/01/02 00, -11.0,  -5.9, -14.4,
15015,503.00,47.7769444,23.9405556,2016/01/03 00, -12.2,  -5.5, -16.4,
15015,503.00,47.7769444,23.9405556,2016/01/04 00, -11.0,  -5.3, -16.6,    .0
15015,503.00,47.7769444,23.9405556,2016/01/05 00,  -5.7,  -4.6,  -8.1,   1.5
```

Acest fișier conține date climatologice de la 23 de stații esențiale de pe teritoriul României. Prima linie conține o descriere a coloanelor, iar valorile sunt separate prin virgulă.

Biblioteca *pandas* pune la dispoziție funcționalități pentru **citirea datelor** din multe formate comune, precum *CSV* sau *Excel*.

```
In [28]:
# Este prima dată când folosim pandas, deci trebuie să importăm biblioteca
# O importăm cu prescurtarea pd - o tehnică comună pentru a scurta codul
import pandas as pd

# Fișierul CSV poate fi citit folosind funcția read_csv
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html#pandas.read_csv
```

```
# După cum observați, are un număr impresionant de argumente ce permit o mare flexibilitate
climatic_2016 = pd.read_csv('Data/RoGovClimatic/climrbsn2016.csv')

In [29]:
# Tipul de date returnat de read_csv este DataFrame - tipul fundamental în Pandas pentru manipularea datelor
# Putem inspecta un DataFrame doar scriindu-i numele într-o celula
climatic_2016
Out[29]:
```

	CODST	ALT	LAT	LON	DATCLIM	TMED	TMAX	TMIN	R24
0	15015	503.0	47.776944	23.940556	2016/01/01 00	-10.0	-5.7	-13.6	NaN
1	15015	503.0	47.776944	23.940556	2016/01/02 00	-11.0	-5.9	-14.4	NaN
2	15015	503.0	47.776944	23.940556	2016/01/03 00	-12.2	-5.5	-16.4	NaN
3	15015	503.0	47.776944	23.940556	2016/01/04 00	-11.0	-5.3	-16.6	0.0
4	15015	503.0	47.776944	23.940556	2016/01/05 00	-5.7	-4.6	-8.1	1.5
...	...	...	...	...	...	...	...	...	...
8413	15480	12.8	44.213889	28.645556	2016/12/27 00	2.1	5.6	-0.6	0.3
8414	15480	12.8	44.213889	28.645556	2016/12/28 00	0.7	2.1	-1.5	0.4
8415	15480	12.8	44.213889	28.645556	2016/12/29 00	1.6	2.7	0.5	0.0
8416	15480	12.8	44.213889	28.645556	2016/12/30 00	0.4	1.5	-0.5	0.6
8417	15480	12.8	44.213889	28.645556	2016/12/31 00	-1.5	1.5	-2.8	0.0

8418 rows × 9 columns
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.shape.html

Înainte de orice analiză, trebuie să înțelegem datele. Uitându-ne la rezultatul celulei de mai sus, putem face multe observații despre structura datelor noastre.

Descrierea oficială ne spune că tabelul conține: *Date climatologice de la cele 23 de stații meteorologice esențiale (flux RBSN: 15015-Ocna Sugatag, 15020-Botoșani, 15090-Iași, 15108-Ceahlău Toaca, 15120-Cluj-Napoca, 15150-Bacău, 15170-Miercurea Ciuc, 15200-Arad, 15230-Deva, 15260-Sibiu, 15280-Vârfu Omu, 15292-Caransebeș, 15310-Galați, 15335-Tulcea, 15346-Râmnicu Valcea, 15350-Buzău, 15360-Sulina, 15410-Drobeta Turnu Severin, 15420-București-Băneasa, 15450-Craiova, 15460-Călărași, 15470-Roșiorii de Vede, 15480-Constanța) care includ temperatura minimă zilnică, temperatura maximă zilnică, temperatura medie zilnică și cantitatea de precipitații zilnice pe intervalul 1961-prezent.* Noi analizăm pentru moment doar datele din anul 2016.

Datele sunt aranjate într-un tabel cu 9 coloane, fiecare coloană reprezentând o valoare masurată, iar fiecare linie o măsurătoare. Există o coloană adițională, fără nume, care reprezintă un *index*. Orice tabel trebuie să aibă una sau mai multe coloane cu rol de index: un identificator unic pentru fiecare observație. În cazul acestor date, nu avem (momentan) un index *natural*, ca parte din date, așa că vom folosi index-ul construit automat de Pandas.

Pentru claritate, semificația coloanelor este:
CODST - Codul de identificare al stației meteo
ALT - Altitudine
LAT - Latitudine
LON - Longitudine
DATCLIM - Data și ora observației
TMED - Temperatura medie zilnică
TMAX - Temperatura maximă zilnică
TMIN - Temperatura minimă zilnică
R24 - Cantitatea de precipitații zilnice

Este important să înțelegem *dimensiunea* datelor pe care le avem disponibile și să verificăm dacă sunt date lipsă. Conform descrierii oficiale, ar trebui să avem câte o observație (linie) pe zi din 23 de stații meteorologice pentru fiecare din cele 366 de zile ale anului 2016 (an bisect). Așadar, ne așteptăm la 366×23=8418366×23=8418 linii în tabel:

```
In [30]:
print('Dimensiunea tabelului este de:', climatic_2016.shape)
Dimensiunea tabelului este de: (8418, 9)
```

În orice set de date trebuie să investigăm dacă avem *date lipsă* sau *eronate*. Așa cum am observat mai devreme, avem multe observații cărora le lipsește valoarea R24. Aceste valori lipsesc chiar din fișier, deci nu e o problemă cu modul de citire al datelor.

Acum trebuie să răspundem la două întrebari: de ce lipsesc aceste valori și cum le tratăm? De exemplu, am putea elimina observațiile incomplete. Din păcate, sunt multe observații fără aceasta valoare, și nu este clar motivul lipsei valorii. Am putea să le înlocuim cu valoarea 0, dar, întrucât există observații cu 0 precipitații, se pare că pur și simplu nu a fost înregistrată valoarea în zilele respective. A pune 0 ar însemna să alterăm datele, întrucât nu suntem siguri că lipsesc observații doar în zilele fără precipitații. Tot ce putem face e să păstrăm coloana așa cum e și să luăm în considerare acest lucru în timpul analizei.

Manipularea datelor - separarea stațiilor meteo
Observăm că valorile din coloanele ALT, LAT și LON descriu doar stația meteo, fiind copiate pentru fiecare observație de la fiecare stație. Pe lângă faptul că această copiere introduce redundanță în tabel, poate îngreuna anumite analize. Separăm aceste informații într-un alt DataFrame, ce descrie stațiile meteo, și în acest tabel să păstrăm doar indicativul din coloana CODST.

Începem prin a separa coloanele CODST, ALT, LAT și LON într-un alt DataFrame:

In [44]:
```
# Folosim paranteze duble pentru că accesam o listă de coloane
statii_meteo = climatic_2016[['CODST', 'ALT', 'LAT', 'LON']]

# Ștergem coloanele suplimentare - păstrăm doar codul stației meteo
climatic_2016 = climatic_2016.drop(columns=['ALT', 'LAT', 'LON'])

statii_meteo
```
Out[44]:

	CODST	ALT	LAT	LON
0	15015	503.0	47.776944	23.940556
1	15015	503.0	47.776944	23.940556
2	15015	503.0	47.776944	23.940556
3	15015	503.0	47.776944	23.940556
4	15015	503.0	47.776944	23.940556
...	...	...	...	...
8413	15480	12.8	44.213889	28.645556
8414	15480	12.8	44.213889	28.645556
8415	15480	12.8	44.213889	28.645556
8416	15480	12.8	44.213889	28.645556
8417	15480	12.8	44.213889	28.645556

8418 rows × 4 columns

In [54]:
```
# Eliminăm observațiile duplicate
# https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.DataFrame.drop_duplicates.html
statii_meteo = statii_meteo.drop_duplicates()
statii_meteo
```
Out[54]:

	ALT	LAT	LON	Nume
CODST				
15015	503.000000	47.776944	23.940556	Ocna Sugatag

	ALT	LAT	LON	Nume
CODST				
15020	161.000000	47.735556	26.645556	Botosani
15090	74.290001	47.163333	27.627222	Iasi
15108	1897.000000	46.977500	25.950000	Ceahlau Toaca
15120	410.000000	46.777778	23.571389	Cluj-Napoca
15150	174.000000	46.557778	26.896667	Bacau
15170	661.000000	46.371389	25.772500	Miercurea Ciuc
15200	116.589996	46.133611	21.353611	Arad
15230	240.000000	45.865000	22.898889	Deva
15260	443.000000	45.789444	24.091389	Sibiu
15280	2504.000000	45.445833	25.456667	Varfu Omu
15292	241.000000	45.416667	22.229167	Caransebes
15310	69.000000	45.473056	28.032222	Galati
15335	4.360000	45.190556	28.824167	Tulcea
15346	237.000000	45.088889	24.362778	Ramnicu Valcea
15350	97.000000	45.132778	26.851667	Buzau
15360	12.690000	45.162222	29.726944	Sulina
15410	77.000000	44.626389	22.626111	Drobeta Turnu Severin
15420	90.000000	44.510556	26.078056	Bucuresti-Baneasa
15450	192.000000	44.310278	23.866944	Craiova
15460	18.719999	44.205833	27.338333	Calarasi
15470	102.150002	44.107222	24.978611	Rosiorii de Vede
15480	12.800000	44.213889	28.645556	Constanta

Observați că indexul a rămas bazat pe indexul din vechiul DataFrame. L-am putea recalcula, folosind metoda reset_index:

```
In [46]:
statii_meteo.reset_index()
```

Out[46]:

	index	CODST	ALT	LAT	LON
0	0	15015	503.000000	47.776944	23.940556
1	366	15020	161.000000	47.735556	26.645556
2	732	15090	74.290001	47.163333	27.627222
3	1098	15108	1897.000000	46.977500	25.950000
4	1464	15120	410.000000	46.777778	23.571389
5	1830	15150	174.000000	46.557778	26.896667
6	2196	15170	661.000000	46.371389	25.772500
7	2562	15200	116.589996	46.133611	21.353611
8	2928	15230	240.000000	45.865000	22.898889
9	3294	15260	443.000000	45.789444	24.091389
10	3660	15280	2504.000000	45.445833	25.456667
11	4026	15292	241.000000	45.416667	22.229167
12	4392	15310	69.000000	45.473056	28.032222
13	4758	15335	4.360000	45.190556	28.824167
14	5124	15346	237.000000	45.088889	24.362778
15	5490	15350	97.000000	45.132778	26.851667
16	5856	15360	12.690000	45.162222	29.726944
17	6222	15410	77.000000	44.626389	22.626111
18	6588	15420	90.000000	44.510556	26.078056
19	6954	15450	192.000000	44.310278	23.866944
20	7320	15460	18.719999	44.205833	27.338333
21	7686	15470	102.150002	44.107222	24.978611
22	8052	15480	12.800000	44.213889	28.645556

dar acest `DataFrame` are un index natural: `CODST`. Așa că vom seta coloana `CODST` ca index, folosind metoda set_index:

In [47]:

```
statii_meteo = statii_meteo.set_index('CODST')
statii_meteo
```

Out[47]:

CODST	ALT	LAT	LON
15015	503.000000	47.776944	23.940556
15020	161.000000	47.735556	26.645556
15090	74.290001	47.163333	27.627222
15108	1897.000000	46.977500	25.950000
15120	410.000000	46.777778	23.571389
15150	174.000000	46.557778	26.896667
15170	661.000000	46.371389	25.772500
15200	116.589996	46.133611	21.353611
15230	240.000000	45.865000	22.898889
15260	443.000000	45.789444	24.091389
15280	2504.000000	45.445833	25.456667
15292	241.000000	45.416667	22.229167
15310	69.000000	45.473056	28.032222
15335	4.360000	45.190556	28.824167
15346	237.000000	45.088889	24.362778
15350	97.000000	45.132778	26.851667
15360	12.690000	45.162222	29.726944
15410	77.000000	44.626389	22.626111
15420	90.000000	44.510556	26.078056
15450	192.000000	44.310278	23.866944
15460	18.719999	44.205833	27.338333
15470	102.150002	44.107222	24.978611
15480	12.800000	44.213889	28.645556

Ar fi util să avem și numele statiei în acest tabel. Vom pleca de la datele din descriere și vom aplica niște transformări:

In [48]:
```
nume_statii_str = '15015-Ocna Sugatag, 15020-Botosani, 15090-Iasi, 15108-
Ceahlau Toaca, 15120-Cluj-Napoca, 15150-Bacau, 15170-Miercurea Ciuc, 15200-
Arad, 15230-Deva, 15260-Sibiu, 15280-Varfu Omu, 15292-Caransebes, 15310-
Galati, 15335-Tulcea, 15346-Ramnicu Valcea, 15350-Buzau, 15360-Sulina, 15410-
Drobeta Turnu Severin, 15420-Bucuresti-Baneasa, 15450-Craiova, 15460-
Calarasi, 15470-Rosiorii de Vede, 15480-Constanta'

# Împărțim string-ul într-o listă de string-uri
nume_statii_list = [x.split('-', 1) for x in nume_statii_str.split(', ')]
nume_statii_list
```

Out[48]:

```
[['15015', 'Ocna Sugatag'],
 ['15020', 'Botosani'],
 ['15090', 'Iasi'],
 ['15108', 'Ceahlau Toaca'],
 ['15120', 'Cluj-Napoca'],
 ['15150', 'Bacau'],
 ['15170', 'Miercurea Ciuc'],
 ['15200', 'Arad'],
 ['15230', 'Deva'],
 ['15260', 'Sibiu'],
 ['15280', 'Varfu Omu'],
 ['15292', 'Caransebes'],
 ['15310', 'Galati'],
 ['15335', 'Tulcea'],
 ['15346', 'Ramnicu Valcea'],
 ['15350', 'Buzau'],
 ['15360', 'Sulina'],
 ['15410', 'Drobeta Turnu Severin'],
 ['15420', 'Bucuresti-Baneasa'],
 ['15450', 'Craiova'],
 ['15460', 'Calarasi'],
 ['15470', 'Rosiorii de Vede'],
 ['15480', 'Constanta']]
```

In [49]:

```
# Transformăm lista într-un mic DataFrame, avem grijă la numele coloanelor,
la tipul lor și la index
# Tipul de date al CODST trebuie să se potrivească cu tipul de date al CODST
din celălalt DataFrame
# Observați înlanțuirea operațiilor
nume_statii = pd.DataFrame(nume_statii_list, columns=['CODST',
'Nume']).astype({'CODST': 'int64'}).set_index('CODST')
```

```
# Combinăm DataFrame-urile, bazându-ne pe coloana CODST, detectată automat după nume
statii_meteo = statii_meteo.join(nume_statii)
statii_meteo
```

Out[49]:

CODST	ALT	LAT	LON	Nume
15015	503.000000	47.776944	23.940556	Ocna Sugatag
15020	161.000000	47.735556	26.645556	Botosani
15090	74.290001	47.163333	27.627222	Iasi
15108	1897.000000	46.977500	25.950000	Ceahlau Toaca
15120	410.000000	46.777778	23.571389	Cluj-Napoca
15150	174.000000	46.557778	26.896667	Bacau
15170	661.000000	46.371389	25.772500	Miercurea Ciuc
15200	116.589996	46.133611	21.353611	Arad
15230	240.000000	45.865000	22.898889	Deva
15260	443.000000	45.789444	24.091389	Sibiu
15280	2504.000000	45.445833	25.456667	Varfu Omu
15292	241.000000	45.416667	22.229167	Caransebes
15310	69.000000	45.473056	28.032222	Galati
15335	4.360000	45.190556	28.824167	Tulcea
15346	237.000000	45.088889	24.362778	Ramnicu Valcea
15350	97.000000	45.132778	26.851667	Buzau
15360	12.690000	45.162222	29.726944	Sulina
15410	77.000000	44.626389	22.626111	Drobeta Turnu Severin
15420	90.000000	44.510556	26.078056	Bucuresti-Baneasa
15450	192.000000	44.310278	23.866944	Craiova
15460	18.719999	44.205833	27.338333	Calarasi
15470	102.150002	44.107222	24.978611	Rosiorii de Vede
15480	12.800000	44.213889	28.645556	Constanta

Exemplu de interogări asupra datelor:

1. Care au fost temperaturile minime și maxime în anul 2016.

Pentru asta, putem să ne uităm individual la coloane:

```
In [50]:
print('Minima înregistrată în anul 2016 a fost de %.1f grade Celsius' %
climatic_2016.TMIN.min())
Minima înregistrată în anul 2016 a fost de -24.0 grade Celsius
```

Sau putem să folosim funcția `describe`, care raportează valori statistice pentru fiecare coloana.

```
In [51]:
climatic_2016.describe()
Out[51]:
```

	CODST	TMED	TMAX	TMIN	R24
count	8418.000000	8418.000000	8418.000000	8418.000000	3155.000000
mean	15275.043478	10.355523	15.862235	5.842493	5.341394
std	141.501830	9.670633	10.988680	8.798955	8.610003
min	15015.000000	-22.400000	-19.700001	-24.000000	0.000000
25%	15150.000000	3.700000	7.800000	0.000000	0.500000
50%	15292.000000	10.100000	15.500000	6.100000	2.000000
75%	15410.000000	18.500000	25.500000	12.800000	6.450000
max	15480.000000	29.500000	37.799999	26.100000	92.000000

Am obținut informații statistice despre fiecare coloană:

Prima linie indică numărul de valori nenule (valide) din fiecare coloană, unde vedem din nou valorile lipsă din coloana R24.

A doua linie indică media fiecarei coloane. Putem observa o medie a temperaturilor maxime zilnice de aproximativ 16 grade Celsius.

A treia linie este o valoare statistică, **deviatia standard.**

A patra linie reprezintă valorile minime. Cum era de așteptat, pentru precipitații minimul este 0, în zilele însorite. Cea mai mică temperatură înregistrată a fost de -24 grade Celsius.

Următoarele trei linii sunt iărași măsuri statistice ce dau informații despre cum sunt *distribuite* datele.

Ultima linie reprezintă, evident, valoarea maximă.

2. Să aflăm stația care a înregistrat valoarea minimă:

```
In [53]:
# Putem selecta linii dintr-un DataFrame indexându-l cu un Series cu valori
boolean
# Pentru a găsi linia ce conține temperatura minimă, obținem întâi un Series
ce are True doar la indecșii unde întâlnim valoarea minimă
# și folosim acest Series pentru a indexa DataFrame-ul inițial.

temperatura_minima = climatic_2016[climatic_2016.TMIN ==
climatic_2016.TMIN.min()]
temperatura_minima
Out[53]:
```

	CODST	DATCLIM	TMED	TMAX	TMIN	R24
3682	15280	2016-01-23	-22.1	-19.5	-24.0	NaN

Temperatura minimă din 2016 a fost înregistrată pe 23 ianuarie. Să aflăm și numele stației!

```
In [68]:
# Parametrul `on` specifică numele coloanei folosită când sunt combinate
tabelele
temperatura_minima.join(statii_meteo, on='CODST')
Out[68]:
```

	CODST	DATCLIM	TMED	TMAX	TMIN	R24	ALT	LAT	LON	Nume
3682	15280	2016-01-23	-22.1	-19.5	-24.0	NaN	2504.0	45.445833	25.456667	Varfu Omu

Temperatura minimă a fost pe Vârfu Omu, la 2504 metri altitudine.

3. Să aflăm unde a fost temperatura minimă în afara munților, la altitudini mai mici de 1000 metri.

Avem doar 2 stații la peste 1000 de metri:

In [84]:
```
statii_meteo[statii_meteo.ALT > 1000]
```
Out[84]:

CODST	ALT	LAT	LON	Nume
15108	1897.0	46.977500	25.950000	Ceahlau Toaca
15280	2504.0	45.445833	25.456667	Varfu Omu

In [87]:
```
# Selectăm doar stațiile meteo de deal și de câmpie, aflate la altitudini sub
1000 metri
statii_meteo_deal_campie = statii_meteo[statii_meteo.ALT < 1000]

# Trebuie să combinăm stațiile meteo selectate cu datele climatice, și să
eliminăm observațiile din cele două stații meteo montane
# Tipul de join, sau felul în care sunt combinate cele două tabele, este
specificat de parametrul `how`
#
# Într-un left join (varianta implicită), vor fi păstrate toate liniile din
tabelul din stanga (aici climatic_2016),
# și coloanele adiționale (provenite de la statii_meteo_deal_campie) vor fi
populate cu NaN dacă nu s-a găsit un corespondent.
# Practic, am fi păstrat valorile temperaturilor pentru observațiile de la
stațiile montane, dar am fi avut NaN pentru ALT, LAT, LON, Nume.
#
# Într-un right join, totul funcționează identic, doar că sunt păstrate
liniile tabelului din dreapta.
#
# Într-un outer join am păstra toate liniile și completa cu NaN valorile care
lipsesc din oricare tabel.
#
# Într-un inner join păstrăm doar liniile complete, ce avem noi nevoie.
#
# Join-urile sunt operatii importante, și complexe. Nu vă faceți griji dacă
nu ați înțeles totul din acest scurt sumar: le vom explora mai în detaliu în
curând.
#
climatic_deal_campie = climatic_2016.join(statii_meteo_deal_campie,
on='CODST', how='inner')
```

```
climatic_deal_campie[climatic_deal_campie.TMIN ==
climatic_deal_campie.TMIN.min()]
Out[87]:
```

	CODST	DATCLIM	TMED	TMAX	TMIN	R24	ALT	LAT	LON	Nume
7706	15470	2016-01-21	-14.9	-9.6	-23.1	NaN	102.150002	44.107222	24.978611	Rosiorii de Vede

Minimul a fost înregistrat pe 21 ianuarie, la stația din Roșiorii de Vede.

Sarcina de realizat:

1. Scrieți secvența de cod care regrupează datele din cele 3 fișiere csv (obținute la laboratul precedent) după o caracteristică de bază (de ex: cursul valutar de la 3 bănci pentru aceeași zi).
2. Determinați min, max, media pentru valorile regrupate pentru fiecare zi aparte și min, max media pentru toate datele stocate.
3. Scrieți secvența de cod care va împărți fișierul csv obținut după concatinare în N fișiere, unde fiecare fișier individual va corespunde unei săptămâini (7 rânduri). Fișierele sunt denumite după prima și ultima dată pe care le conțin (de ex 20231103_20231109.csv).
4. Scrieți funcția ce are o dată de intrare tip datetime și returnează liniile din DataFrame (din fișierul cu date concatinate) pentru această dată sau None dacă nu există date pentru această dată.
5. Scrieți secvența de cod care permite afișarea datelor pe secvențe de timp. De ex. pentru fiecare săptămână separat (funcția are o dată de intrare tip datetime) și returnează liniile din DataFrame pentru această perioadă sau None dacă nu există date pentru zilele indicate.