Mini projects – Tools and tips

GM-3-S1-DATA

All version

A. Platzer, L. Girier, C. Le Bourlot

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

2025-2026

Contents

1 Tutorial for the mini projects **1**
1.1 Collaborative environment . . . 1
1.2 Before you start . . . 1
1.3 Loading your data set . . . 1
1.3.1 With NumPy, numerical data only . . . 2
1.3.2 With NumPy, for numerical and categorical data . . . 3
1.3.3 With Pandas . . . 6
1.4 Replacing categorical with numerical data . . . 8

1 Tutorial for the mini projects

1.1 Collaborative environment

You must complete the mini-projects as pairs. To facilitate your project you can work within collaborative environments such as

- DeepNote
- Google Colab (similar to Google Docs but for Jupyter Notebooks)

Most advanced users can also use the Gitlab instance provided by INSA. To discover Git and Gitlab, you can start with this video where Github = Gitlab.

1.2 Before you start

Follow the flowchart with the data set you have selected for your project and answer the few questions below.

- What is the file format?
- What are the metadata? where does the data comes from? what does it represent (say it with a short sentence)?
- What kind of data are we dealing with? cf. lecture
  - numerical
  - categorical
  - ...
- What is the dimensionality (1D, 2D, 3D, ..)?
- How is the data stored?
  - binary?
  - ascii encoding?
  - decimal separator?

1.3 Loading your data set

In this tutorial, we use the data set from Tutorial 3 once again as it may be representative of the data set you selected for your project: it contains both numerical and categorical data (numbers and text altogether).

The aim of this tutorial is to provide you with snippets of code to help you load your data set with NumPy or Pandas (another library that we have not seen in class, but that is available in the `dataenv` environment provided by the Install Party. You have nothing to do if you followed it).

As a reminder, the data set, downloaded from Meteo France public database consisted of two files in the `data/` folder:

- `Q_69_latest-2023-2024_RR-T-Vent.csv.gz`
- `Q_descriptif_champs_RR-T-Vent.txt`, containing a description of each field (metadata)

1.3.1 With NumPy, numerical data only

[1]:
```
import numpy as np
```

If you use NumPy `genfromtxt()` function with the standard arguments we have used so far with mixed numerical and categorical data, here is what happens:

[2]:
```
data = np.genfromtxt("data/Q_69_latest-2023-2024_RR-T-Vent.csv",
                     delimiter=";",
                     skip_header=1,
                     )
data, data[:,1], data[:,2]
```

[2]:
```
(array([[6.9008001e+07,           nan, 4.5843333e+01, …,           nan,
                    nan,           nan],
        [6.9008001e+07,           nan, 4.5843333e+01, …,           nan,
                    nan,           nan],
        [6.9008001e+07,           nan, 4.5843333e+01, …,           nan,
                    nan,           nan],
        …,
        [6.9299001e+07,           nan, 4.5726500e+01, …, 9.0000000e+00,
         0.0000000e+00, 9.0000000e+00],
        [6.9299001e+07,           nan, 4.5726500e+01, …, 9.0000000e+00,
         0.0000000e+00, 9.0000000e+00],
        [6.9299001e+07,           nan, 4.5726500e+01, …, 9.0000000e+00,
         3.5000000e+01, 9.0000000e+00]], shape=(11592, 58)),
 array([nan, nan, nan, …, nan, nan, nan], shape=(11592,)),
 array([45.843333, 45.843333, 45.843333, …, 45.7265  , 45.7265  ,
        45.7265  ], shape=(11592,)))
```

Some columns contains `nan` which means **N**ot**AN**umber: it is the default value NumPy will replace anything that is not a float by.

To overcome this issue in Tutorial 3, we used the arguments `usecols` which allowed us to select only on the numerical fields: `NUM_POSTE`, `AAAAMMJJ` and `TM`, corresponding respectively to column number `0`, `5` and `16`.

[3]:
```
# load the data set with only the relevant three columns (numerical values␣
  ↪only)
data = np.genfromtxt("data/Q_69_latest-2023-2024_RR-T-Vent.csv",
                     delimiter=";",
                     usecols=[0,5,16],  # select fields NUM_POSTE, AAAAMMJJ,␣
  ↪TM
                     skip_header=1,
                     )
data
```

[3]:
```
array([[6.9008001e+07, 2.0230101e+07, 1.2500000e+01],
       [6.9008001e+07, 2.0230102e+07, 9.5000000e+00],
       [6.9008001e+07, 2.0230103e+07, 6.1000000e+00],
       …,
       [6.9299001e+07, 2.0241003e+07, 1.2900000e+01],
       [6.9299001e+07, 2.0241004e+07, 1.1100000e+01],
       [6.9299001e+07, 2.0241005e+07, 1.0400000e+01]], shape=(11592, 3))
```

After the loading is complete, you could operate on the resulting NumPy arrays to extract the relevant part (temperature of LYON-BRON station from July 1st 2024 to August 31st 2024):

[4]:
```
# filter out the relevant the data set
# method 1: keep original data array safe
mask = np.logical_and(data[:,0] == 69029001,
                      np.logical_and(data[:, 1] >= 20240701,
                                     data[:, 1] < 20240901))
data_masked = data[mask]
temp = data_masked[:,2] # keep only temperature values, 3rd column
print(temp, data_masked.shape)

# method 2: erases original data arrays as you go
# data = data[data[:, 0] == 69029001] # LYON-BRON
# data = data[data[:, 1] >= 20240701] # from July 1st 2024
# data = data[data[:, 1] < 20240901]  # to Sept 1st 2024
# temp = data[:,2] # keep only temperature values, 3rd column
# print(temp, data.shape)
```

```
[21.5 18.7 18.1 20.9 23.5 19.9 19.1 22.6 26.6 23.5 24.2 20.7 18.5 21.4
 23.3 22.8 22.7 24.  23.9 25.9 22.2 22.4 23.2 22.2 22.4 24.4 27.  24.7
 26.6 29.  29.1 28.  25.9 24.7 23.1 23.9 26.2 26.9 24.4 25.  25.8 29.
 30.3 27.4 23.3 24.4 25.8 23.9 18.2 20.5 21.4 20.2 19.4 23.7 25.8 19.2
 19.3 20.7 26.1 26.8 26.2 26.9] (62, 3)
```

You should end up with an array of 62 values

1.3.2 With NumPy, for numerical and categorical data

Advanced arguments of the numpy.genfromtxt() function (check the docs) can be leveraged to deal with both numerical and categorical data.

To simplify the demonstration, we will only load the 5 first columns:

```
NUM_POSTE   : numéro Météo-France du poste sur 8 chiffres
NOM_USUEL   : nom usuel du poste
LAT         : latitude, négative au sud (en degrés et millionièmes de degré)
LON         : longitude, négative à l'ouest de GREENWICH (en degrés et millionièmes de deg
ALTI        : altitude du pied de l'abri ou du pluviomètre si pas d'abri (en m)
```

The second column contains categorical data: the name of the station as a string.

dtype argument To specify the data type of each column, we can use `dtype` argument and set it to a coma separated fields:

- first column contains integers, short-hand is `i`
- second column contains unicode characters, of size unknown, short-hand is `U` with the number of characters in the string. Let us use `U25` for a 25-long string, which should be enough for all station names
- third, fourth and fifth columns contain floating numbers, short-hand is `f`

Help : To learn more about data types (`dtype`) in NumPy, refer to the docs

[5]:
```
data_dtype = np.genfromtxt("data/Q_69_latest-2023-2024_RR-T-Vent.csv",
                           usecols=range(5),
                           delimiter=";",
                           skip_header=1,
                           dtype=np.dtype("i,U25,f,f,f"))
data_dtype
```

[5]:
```
array([(69008001, 'ANCY_SAPC', 45.843334, 4.508167, 626.),
       (69008001, 'ANCY_SAPC', 45.843334, 4.508167, 626.),
       (69008001, 'ANCY_SAPC', 45.843334, 4.508167, 626.), …,
       (69299001, 'LYON-ST EXUPERY', 45.7265  , 5.077833, 235.),
       (69299001, 'LYON-ST EXUPERY', 45.7265  , 5.077833, 235.),
       (69299001, 'LYON-ST EXUPERY', 45.7265  , 5.077833, 235.)],
      shape=(11592,), dtype=[('f0', '<i4'), ('f1', '<U25'), ('f2', '<f4'),
('f3', '<f4'), ('f4', '<f4')])
```

Now we can see that the second column with the station names is correctly read by NumPy.

Notice however that each line of the data array is now a **tuple**, so we cannot easily slice it to have the corresponding column, the usual command yields an error:

[6]:
```
data_dtype[:,1]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[6], line 1
----> 1 data_dtype[:,1]
```

```
IndexError: too many indices for array: array is 1-dimensional, but 2 were␣
  ↪indexed
```

names argument A workaround is to use another argument of `genfromtxt()`: `names=True`. Then, the columns are no longer referred to by their number but by their names, taken from the first line of the CSV file. Hence, we remove the `skip_header` argument to actually read and use this first line:

[7]:
```
data_names = np.genfromtxt("data/Q_69_latest-2023-2024_RR-T-Vent.csv",
                           usecols=range(5),
                           delimiter=";",
                           dtype=np.dtype("i,U25,f,f,f"),
                           names=True)
data_names
```

[7]:
```
array([(69008001, 'ANCY_SAPC', 45.843334, 4.508167, 626.),
       (69008001, 'ANCY_SAPC', 45.843334, 4.508167, 626.),
       (69008001, 'ANCY_SAPC', 45.843334, 4.508167, 626.), …,
       (69299001, 'LYON-ST EXUPERY', 45.7265  , 5.077833, 235.),
       (69299001, 'LYON-ST EXUPERY', 45.7265  , 5.077833, 235.),
       (69299001, 'LYON-ST EXUPERY', 45.7265  , 5.077833, 235.)],
      shape=(11592,), dtype=[('NUM_POSTE', '<i4'), ('NOM_USUEL', '<U25'),
('LAT', '<f4'), ('LON', '<f4'), ('ALTI', '<f4')])
```

To view a certain column of the data set, you now pass the name of the column in brackets:

[8]:
```
data_names["NUM_POSTE"]
```

[8]:
```
array([69008001, 69008001, 69008001, …, 69299001, 69299001, 69299001],
      shape=(11592,), dtype=int32)
```

[9]:
```
data_names["NOM_USUEL"]
```

[9]:
```
array(['ANCY_SAPC', 'ANCY_SAPC', 'ANCY_SAPC', …, 'LYON-ST EXUPERY',
       'LYON-ST EXUPERY', 'LYON-ST EXUPERY'], shape=(11592,), dtype='<U25')
```

Note that when we used `dtype` argument only, the name of the columns, also called *fields*, were given the default value `f0`, `f1`, ... `f5`:

[10]:
```
data_dtype["f0"]
```

[10]:
```
array([69008001, 69008001, 69008001, …, 69299001, 69299001, 69299001],
      shape=(11592,), dtype=int32)
```

unpack arguments To directly have all columns as a list of one-dimensional arrays and not worry about accessing the columns with their number or their names, you can use the `unpack` argument which changes the output of the `genfromxtxt()` function:

[11]:
```
data_unpack = np.genfromtxt("data/Q_69_latest-2023-2024_RR-T-Vent.csv",
                            usecols=range(5),
                            delimiter=";",
                            dtype=np.dtype("i,U25,f,f,f"),
                            names=True,
                            unpack=True)
data_unpack
```

[11]:
```
[array([69008001, 69008001, 69008001, …, 69299001, 69299001, 69299001],
       shape=(11592,), dtype=int32),
 array(['ANCY_SAPC', 'ANCY_SAPC', 'ANCY_SAPC', …, 'LYON-ST EXUPERY',
        'LYON-ST EXUPERY', 'LYON-ST EXUPERY'], shape=(11592,), dtype='<U25'),
 array([45.843334, 45.843334, 45.843334, …, 45.7265  , 45.7265  ,
        45.7265  ], shape=(11592,), dtype=float32),
 array([4.508167, 4.508167, 4.508167, …, 5.077833, 5.077833, 5.077833],
       shape=(11592,), dtype=float32),
 array([626., 626., 626., …, 235., 235., 235.],
       shape=(11592,), dtype=float32)]
```

`data_unpack` is a list of one-dimensional arrays, each corresponding to a column:

[12]:
```
data_unpack[1]
```

[12]:
```
array(['ANCY_SAPC', 'ANCY_SAPC', 'ANCY_SAPC', …, 'LYON-ST EXUPERY',
       'LYON-ST EXUPERY', 'LYON-ST EXUPERY'], shape=(11592,), dtype='<U25')
```

1.3.3 With Pandas

The above is somewhat an advanced usage of `numpy.genfromxtxt()` function and you still need to specify the data type of each column by hand.

To load complex data set, and in particular CSV files, a more convenient method is to used Pandas library, which is more suited to deal with mixed data set (containing both numerical and categorical data).

You can use `read_csv()` function which is analogous but more powerful than `genfromtxt()`. Check the docs for additional arguments, like `decimal`, but most of the defaults should work with your data set. The output is a **DataFrame** (often called `df`), the basis data structure of Pandas (a sort of boosted NumPy array). In the Jupyter Notebook, it is displayed as "real" table, with the very first column in bold on the left being the line indices:

[13]:
```
import pandas as pd
```

> ⚠ **Warning:** If this line does not work (`ModuleNotFounderError`), you need to install Pandas. Look at the tutorials provided for the Install Party, on Moodle.

[14]:
```
df = pd.read_csv("data/Q_69_latest-2023-2024_RR-T-Vent.csv",
                 delimiter=";")
```

```
df # display complete data frame
```

[14]: WARNING: cell output was removed by the exporter (too long).

The columns are again accessed with their names in brackets, as shown below:

[15]:
```
df["NOM_USUEL"]
```

[15]:
```
0              ANCY_SAPC
1              ANCY_SAPC
2              ANCY_SAPC
3              ANCY_SAPC
4              ANCY_SAPC
              …
11587    LYON-ST EXUPERY
11588    LYON-ST EXUPERY
11589    LYON-ST EXUPERY
11590    LYON-ST EXUPERY
11591    LYON-ST EXUPERY
Name: NOM_USUEL, Length: 11592, dtype: object
```

Notice that we have not skipped the header, so the columns are properly labeled. Their names are used as keys, as in a Python dictionary:

[16]:
```
df.keys()
```

[16]:
```
Index(['NUM_POSTE', 'NOM_USUEL', 'LAT', 'LON', 'ALTI', 'AAAAMMJJ', 'RR',␣
  ↪'QRR',
       'TN', 'QTN', 'HTN', 'QHTN', 'TX', 'QTX', 'HTX', 'QHTX', 'TM', 'QTM',
       'TNTXM', 'QTNTXM', 'TAMPLI', 'QTAMPLI', 'TNSOL', 'QTNSOL', 'TN50',
       'QTN50', 'DG', 'QDG', 'FFM', 'QFFM', 'FF2M', 'QFF2M', 'FXY', 'QFXY',
       'DXY', 'QDXY', 'HXY', 'QHXY', 'FXI', 'QFXI', 'DXI', 'QDXI', 'HXI',
       'QHXI', 'FXI2', 'QFXI2', 'DXI2', 'QDXI2', 'HXI2', 'QHXI2', 'FXI3S',
       'QFXI3S', 'DXI3S', 'QDXI3S', 'HXI3S', 'QHXI3S', 'DRR', 'QDRR'],
      dtype='object')
```

[17]:
```
df.keys()[0]
```

[17]:
```
'NUM_POSTE'
```

To select the reduced data set as in Tutorial 3 the code now is:

[18]:
```
dfr = df[df["NOM_USUEL"] == "LYON-BRON"]
dfr = dfr[dfr["AAAAMMJJ"] >= 20240701]
dfr = dfr[dfr["AAAAMMJJ"] < 20240901]
dfr = dfr[["NUM_POSTE", "AAAAMMJJ", "TM"]]
dfr # display reduced data set
```

[18]:
```
      NUM_POSTE  AAAAMMJJ    TM
1835   69029001  20240701  21.5
1836   69029001  20240702  18.7
1837   69029001  20240703  18.1
1838   69029001  20240704  20.9
1839   69029001  20240705  23.5
…          …         …     …
1892   69029001  20240827  20.7
1893   69029001  20240828  26.1
1894   69029001  20240829  26.8
1895   69029001  20240830  26.2
1896   69029001  20240831  26.9

[62 rows x 3 columns]
```

[19]:
```
temp = np.array(dfr["TM"]) # convert Series to NumPy array
temp
```

[19]:
```
array([21.5, 18.7, 18.1, 20.9, 23.5, 19.9, 19.1, 22.6, 26.6, 23.5, 24.2,
       20.7, 18.5, 21.4, 23.3, 22.8, 22.7, 24. , 23.9, 25.9, 22.2, 22.4,
       23.2, 22.2, 22.4, 24.4, 27. , 24.7, 26.6, 29. , 29.1, 28. , 25.9,
       24.7, 23.1, 23.9, 26.2, 26.9, 24.4, 25. , 25.8, 29. , 30.3, 27.4,
       23.3, 24.4, 25.8, 23.9, 18.2, 20.5, 21.4, 20.2, 19.4, 23.7, 25.8,
       19.2, 19.3, 20.7, 26.1, 26.8, 26.2, 26.9])
```

1.4 Replacing categorical with numerical data

Once you have loaded your data set with any of the methods above **you should only work with numerical data**.

Therefore, if some categorical fields are of interest for your project, replace the string values by a number (preferably integers).

For instance, here the names of the stations in `NOM_USUEL` were also encoded with the stations'ID in `NUM_POSTE`.

There are three stations in Lyon:

1. `LYON-BRON`
2. `LYON TETE D'OR`
3. `LYON-ST EXUPERY`

Let us number them instead of using their name or long ID. To replace categorical values by numerical values you could try something like this:

[20]:
```
nom_usuel = np.array(df["NOM_USUEL"])

nom_usuel_numbers = np.zeros(nom_usuel.shape)
index1 = nom_usuel == "LYON-BRON"
index2 = nom_usuel == "LYON TETE D'OR"
```

```
index3 = nom_usuel == "LYON ST-EXUPERY"
nom_usuel_numbers[index1] = 1
nom_usuel_numbers[index2] = 2
nom_usuel_numbers[index3] = 3

print("Number of changes: ", np.count_nonzero(nom_usuel_numbers))
```

```
Number of changes:  1288
```

Advanced equivalent code using a dictionary to map the names and the numbers and automatically change the values:

[21]:
```
names_to_numbers = {"LYON-BRON": 1,
                    "LYON TETE D'OR": 2,
                    "LYON ST-EXUPERY": 3}

nom_usuel_numbers = np.zeros(nom_usuel.shape)
for key, val in names_to_numbers.items():
    nom_usuel_numbers[nom_usuel == key] = val

print("Number of changes: ", np.count_nonzero(nom_usuel_numbers))
```

```
Number of changes:  1288
```