Machine Learning for IoT
Lab 1 – Getting Started

The following instructions describe how to setup the development environment that will be used for the labs and homework. Part of the activities will take place on a cloud environment; the rest will take place locally on your personal computer.

HowTo 1: Cloud Environment Setup

1.1 Computing: Deepnote
The cloud development environment is hosted on Deepnote (deepnote.com), a collaborative platform where developers can create, run, and share Jupyter notebooks.
Join the *ML4IoT 23 LABs* workspace by clicking on the invitation link received on your student mailbox (sXXXXXX@studenti.polito.it).

The workspace is organized in *projects*. Each project is a collection of Jupyter notebooks and files, that all run in the same environment.

The *ML4IoT 23 LABs* workspace contains the following projects:
- LABs Material: (read-only) Notebooks, code, and files with all the reference material.
- Live Coding: (read-only) Notebooks created during the labs.
- Q&A: (comment-only) project where students can post their questions.
- TeamXX: projects associated to a student team, where students belonging to the team can create, edit, and run the code needed to solve the labs' exercises.
- Personal Material: private project where each student can try her/his personal code.

Open the project *Labs Material* and see to notebook *0. Getting started* to learn how to setup your personal/team projects.

1.2 Storage: Redis Cloud
The cloud storage is hosted on Redis Cloud. Redis is a key-value database, i.e., each entry of the database consists of a simple string (the key) that is unique and a data field (the value).
Redis integrates a time-series data structure, where the timestamp is the key and the data is the value. Moreover, Redis offers many features to query and post-process time-series data efficiently.

Sign up for a FREE account on https://redis.com/try-free.

HowTo 2: Local Environment Setup

2.1. Download & Install Visual Studio Code (VS Code)
   a. Download the installer from https://code.visualstudio.com/Download
   b. Run the installer

2.2. Check if Python is installed:

a. In VS Code, open a terminal: go to *Terminal*, *New Terminal*
b. (Windows) Run the following command:

```
py -3.10 --version
```

c. (Ubuntu and macOS): Run the following command:

```
python3.10 --version
```

d. The expected output is:

```
Python 3.10.7
```

e. If you get the expected output (any version >3.10 is ok), go to 2.4.
If you get an error message, go to 2.3.

2.3. Install Python 3.10 (if not already installed)

a. (Windows) Download the Python installer and follow the installation instructions:
https://www.python.org/ftp/python/3.10.7/python-3.10.7-amd64.exe

b. (Ubuntu) Run the following commands:

```
sudo apt update
sudo apt upgrade -y
sudo apt install build-essential -y
sudo apt install software-properties-common -y
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt update
sudo apt install python3.10 -y
```

c. (macOS) Install Homebrew (see https://brew.sh) and run the following commands:

```
xcode-select --install
brew install python@3.10
```

2.4. (Ubuntu and macOS) Install other dependencies:

a. (Ubuntu) Run the following commands:

```
sudo apt install python3.10-venv -y
sudo apt install python3.10-dev -y
sudo apt install libportaudio2 -y
```

b. (macOS) Run the following commands:

```
brew install portaudio
python3.10 -m pip install --user --upgrade pip
python3.10 -m pip install --user virtualenv
```

2.5. Install extensions in VS Code

a. Go to *View*, *Command Palette…*
b. Type *Extensions: Install Extensions*
c. In the search bar, type *Python*
d. Install the *Python* extension by Microsoft
e. In the search bar, type *REST Client*
f. Install the *REST Client* extension by Huachao Mao
g. In the search bar, type *audio-preview*
h. Install the *audio-preview* extension by sukumo28

2.6. Create your Working Directory

a. In VS Code, open a terminal (*Terminal, New Terminal)*.
b. Create a new directory with the following command:

```
mkdir ml4iot
```

2.7. Open the Working Directory in VS Code:

a. Go to *Explorer* (CTRL+SHIFT+E), *Open Folder*, select the *ml4iot* directory, and press *OK*.

2.8. Create a Python Virtual Environment

a. In VS Code, open a terminal (*Terminal, New Terminal)*.
b. (Windows) Create a Python virtual environment named *py310*

```
py -3.10 -m venv py310
```

c. (Ubuntu and macOS) Create a Python virtual environment named *py310*

```
python3.10 -m venv py310
```

The virtual environment is a local copy of the system-wide Python installation. It helps to keep dependencies required by different projects separate by creating isolated environments for them. All the Python packages installed will be stored in the `py310` folder and will be available only inside the virtual environment.

d. (Windows) Switch from the system-wide Python to the local Python activating the virtual environment:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
.\py310\Scripts\activate
```

e. (Ubuntu and macOS) Switch from the system-wide Python to the local Python activating the virtual environment:

```
source py310/bin/activate
```

NB: Before running your code, always check that the virtual environment is active.

Suggestion: check the official Python documentation at https://docs.python.org/3/tutorial/venv.html

2.9. Install the required Python packages

a. Download from the *Portale della Didattica* the *requirements.txt* file and copy it to the *ml4iot* directory.

b. Upgrade the `pip` package manager:

```
pip install -U pip
```

c. Install the requirements running the `pip` command:

```
pip install -r requirements.txt
```

Exercise 1: Record Audio with the integrated/USB Microphone

In VS Code, write a Python script to record audio data with your PC and the integrated/USB microphone.

a. If you do not have an integrated microphone, connect an USB microphone to your PC.

b. Create a new Python file (e.g., *lab1_ex1.py*) and write a script that uses the `sounddevice` package to record audio data. Stop the recording when the Q key is pressed.

   Suggestion: check the documentation at https://python-sounddevice.readthedocs.io/en/0.4.5/api/streams.html

c. Modify the script to store the audio data on disk every second. Use the stream *callback* function to store data in parallel with recording. Use the *scipy.io.wavfile.write* function to store the audio data on disk. Use the timestamp of the recording as the filename.

   Suggestion: check the documentation at https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.write.html

d. Modify the script to let the user to disable/enable the audio storage by pressing the P key.

e. Modify the script to introduce the following parameters as command-line arguments (use the `argparse` package):

- Resolution (str): int16 or int32
- Sampling Rate in Hertz (int)
- Number of Channels (int)
- File duration in seconds (int)

f. Measure the output .wav size (in KB) with *os.path.getsize* method from the *os* package

g. Run the script with different parameters' values. Try different resolutions (int16, int32), sampling rates (16 kHz, 44.1 kHz and 48 kHz) and channels (1 and 2, if supported by your microphone).

h. Listen to the collected recording (with *audio-preview*) and check the audio quality in the different cases. Also, fill the table below and comment the results. Finally, define an equation that computes the wave file size as a function of resolution, sampling rate, #channels, and recording duration. Double-check the equation with the collected statistics.

Resolution	**Sampling Rate (kHz)**	**Channels**	**Size (KB)**
int16	16	1	
int16	16	2	
int16	44.1	1	
int16	44.1	2	
int16	48	1	
int16	48	2	
int32	16	1	
int32	16	2	
int32	44.1	1	
int32	44.1	2	
int32	48	1	
int32	48	2	

Exercise 2: Monitor your PC battery status with Python

2.1 In VS Code, write a Python script to monitor your PC battery status.

a. Create a new Python file (e.g., lab1_ex2.py) and write a script that uses the `psutil` module to monitor the battery level of your PC and check if the power is plugged.

   Suggestion: check the documentation at:
   https://psutil.readthedocs.io/en/latest/#psutil.sensors_battery

b. Every 2 seconds, print the battery status (battery level and power plugged) using the following format:

   year-month-day hour:minute:second.microseconds - *mac_address:*battery = *battery_level*
   year-month-day hour:minute:second.microseconds - *mac_address:*power = *power_plugged*

   where *mac_address* is the MAC address of your network card, *battery_level* is the battery level in percentage, and *power_plugged* is an integer equal to 1 if the power is plugged, 0 otherwise.

   Example:

```
2022-10-01 19:21:51.699254 - 0xf0b61e0bfe09:battery = 100
2022-10-01 19:21:51.699254 - 0xf0b61e0bfe09:power = 1
2022-10-01 19:21:53.701326 - 0xf0b61e0bfe09:battery = 100
2022-10-01 19:21:53.701326 - 0xf0b61e0bfe09:power = 1
```

c. Modify the script to store the battery level and power plugged flag to Redis Cloud. For storage, define two Redis TimeSeries, called *mac_address:*battery and *mac_address:*power, respectively (e.g., *0xf0b61e0bfe09:battery* and *0xf0b61e0bfe09:power*)

2.2 On Deepnote, create a new Python notebook to read the battery data from Redis and visualize it.