

IOT ONLINE COURSE

LoRa Gateway AI Framework

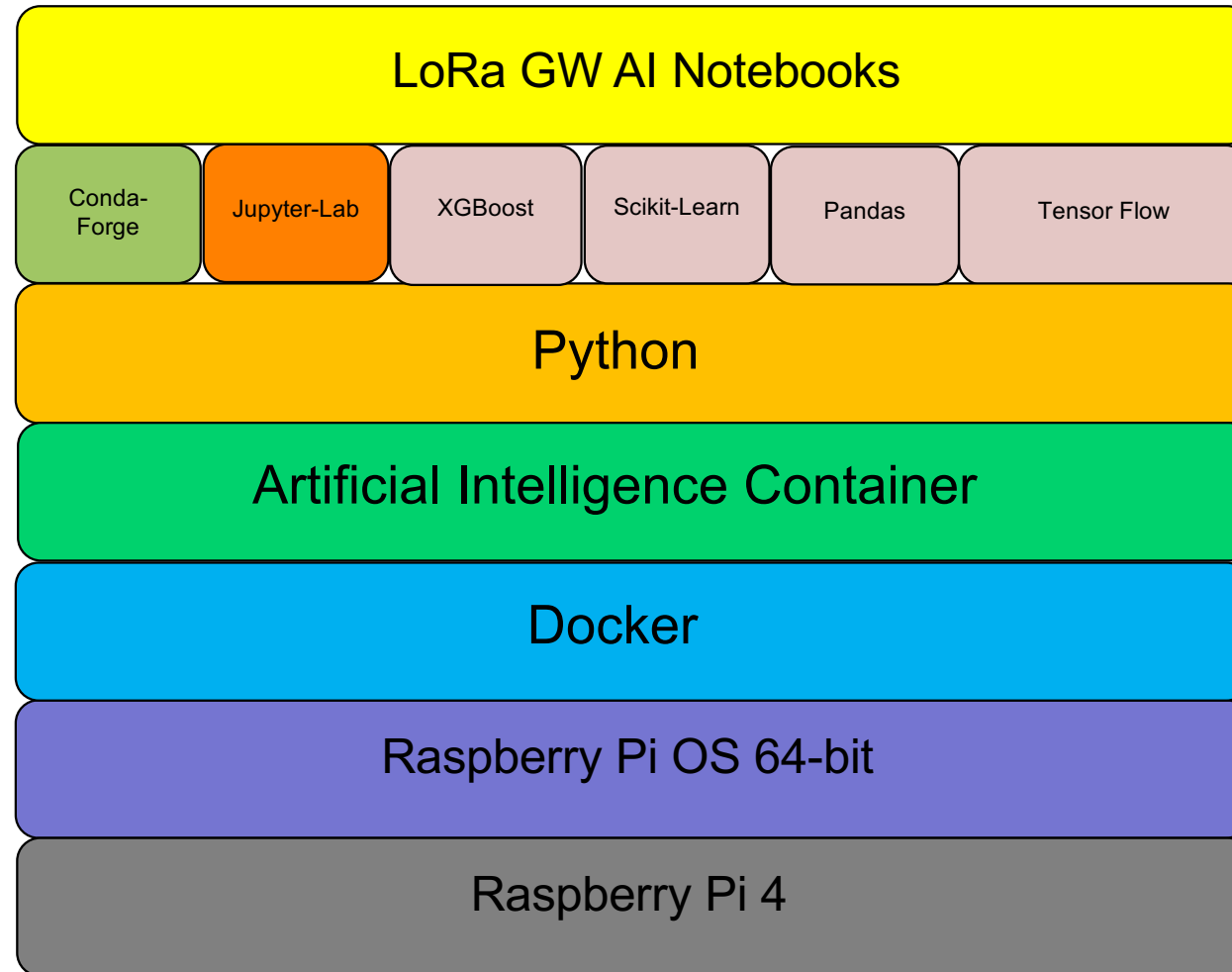
LGAIF-1: Introduction

Prof. Congduc Pham
<http://www.univ-pau.fr/~cpham>
Université de Pau, France

Slides realized by J. Mantilla

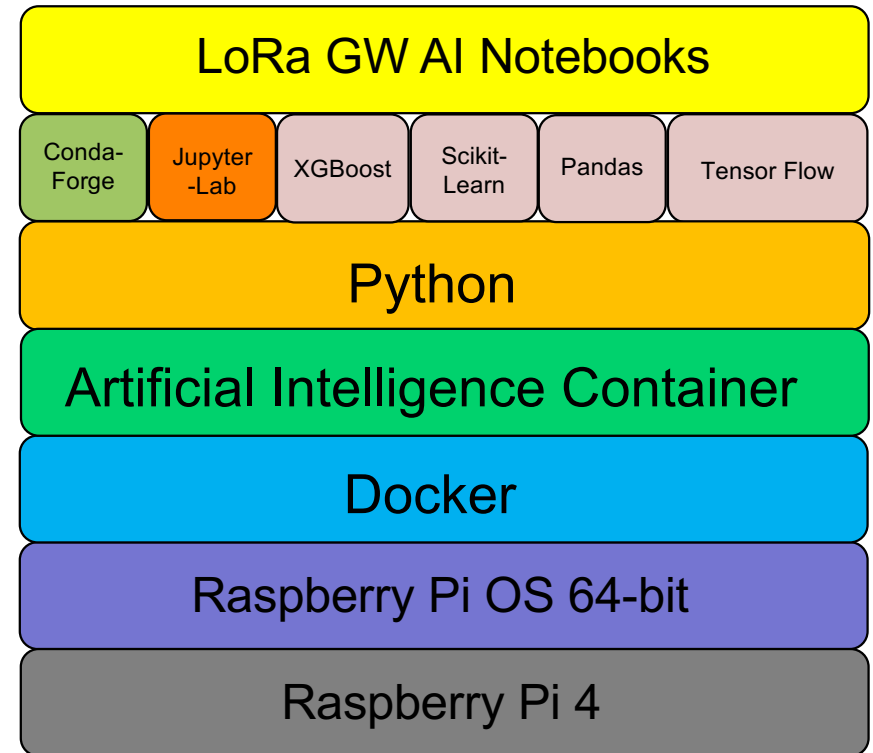


LoRa Gateway AI Framework



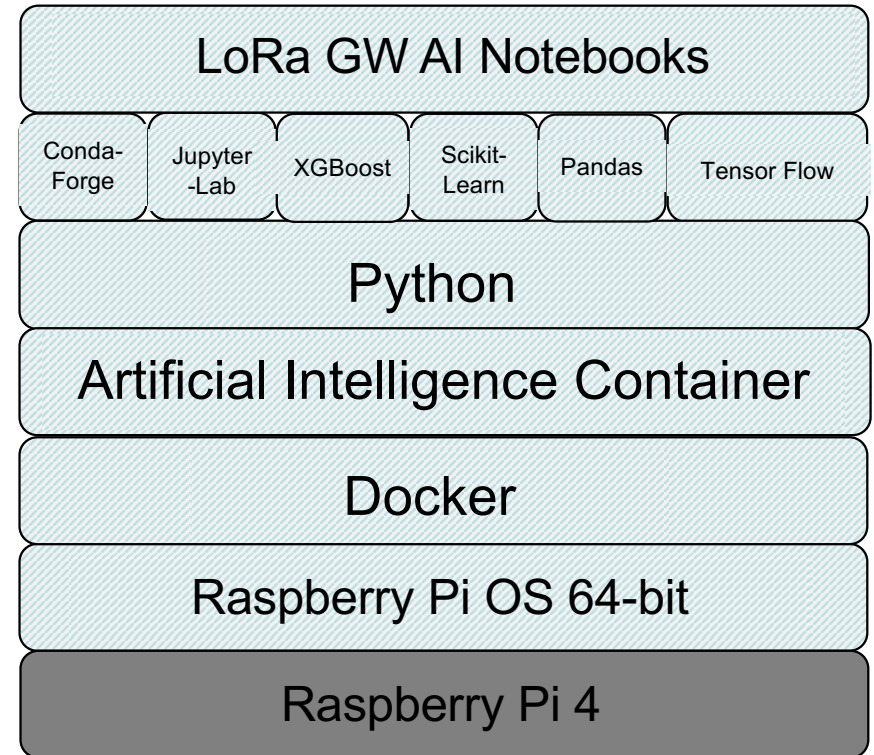
What is it?

- ⦿ This framework is an interface or tool that allows users to build and use Artificial Intelligence models easily on LoRa Gateway, without getting into the depth of the underlying algorithms and runtime environment setup.
- ⦿ Based on a Jupyter-Lab Docker container



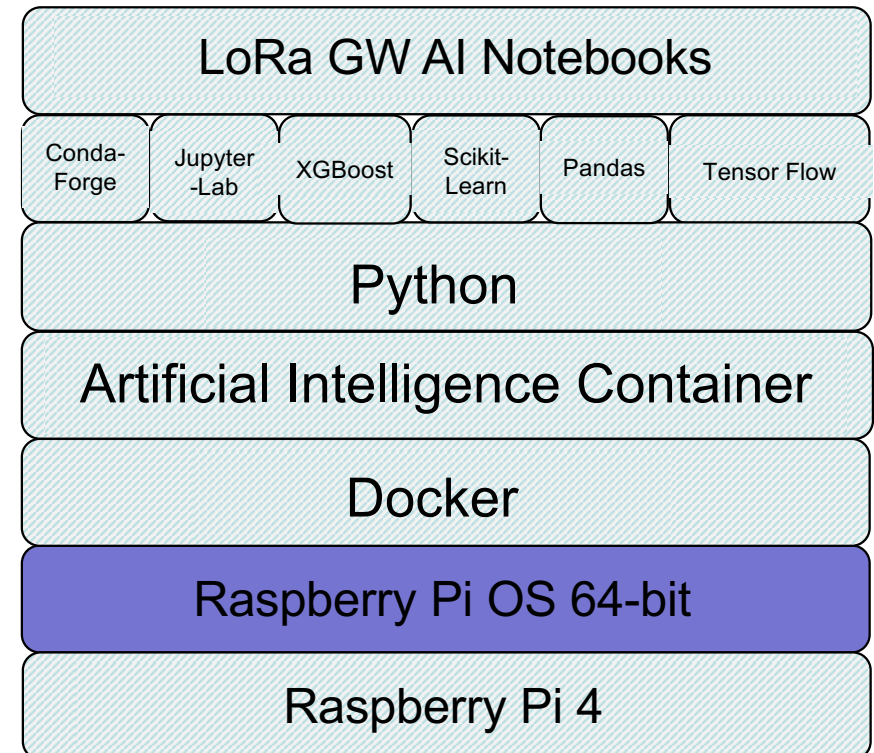
Hardware

- LoRa Gateway is based on Raspberry Pi.
- Lora GW AI Framework upper layers designed for 64-bit CPU.
- Raspberry Pi 4 has a full 64-bit ARM CPU.
- More RAM improves Lora GW AI Framework performance.



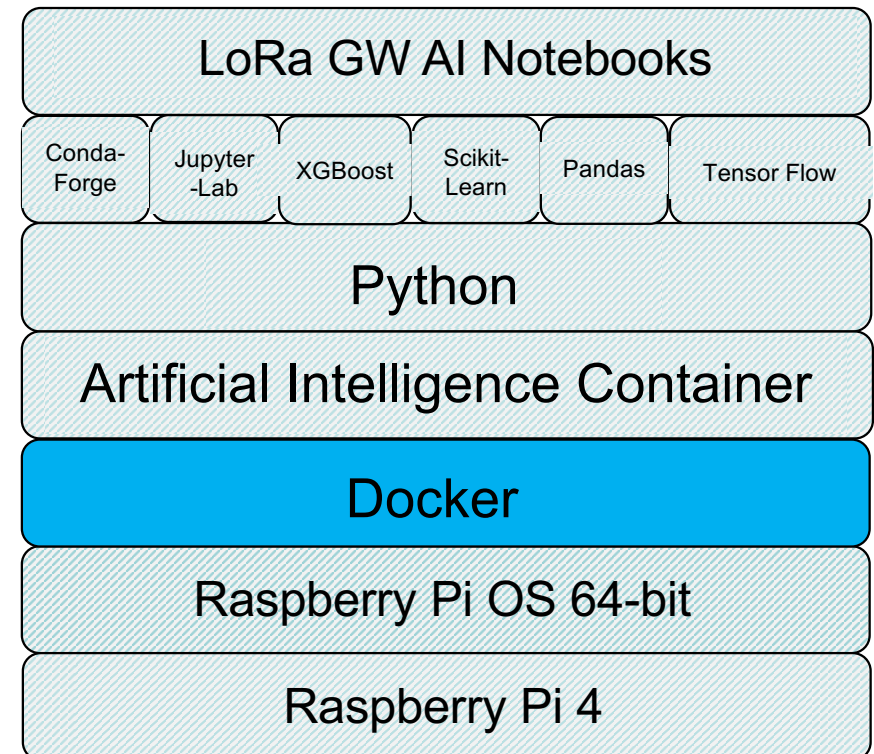
Operating System

- The official Raspberry Pi OS is 32-bit.
- There are Raspberry Pi OS 64-bit releases too.
- Other 64-bit ARM Linux OS are available, e.g. Ubuntu Server, Manajaro, ARMBian
 - Note that the previous don't make the best out of Raspberry Pi hardware (GPIO, PiCam, SPI I²C 1-Wire support, etc.)



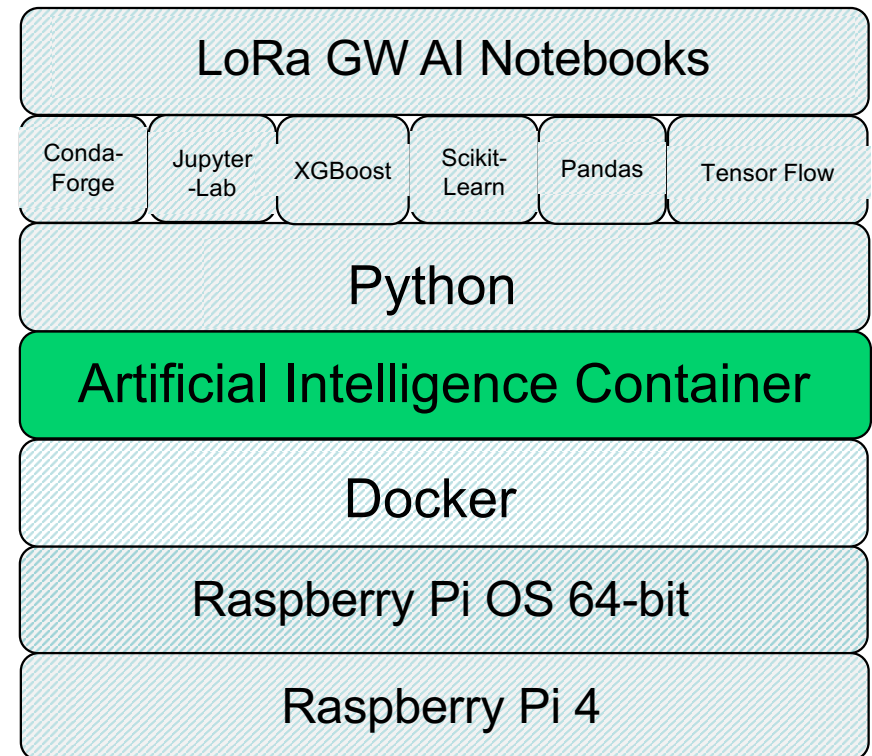
Containerization Platform

- ⦿ Bundles an application and all its dependencies into a single object called a *container*.
- ⦿ A container can be executed with guarantee that execution environment exposed to the application is the same in both, development Raspberry Pi and deployed LoRa Gateway.
- ⦿ Requires 64-bit OS and CPU.
- ⦿ There are 32-bit releases for ARMv7I CPU but no active development community.



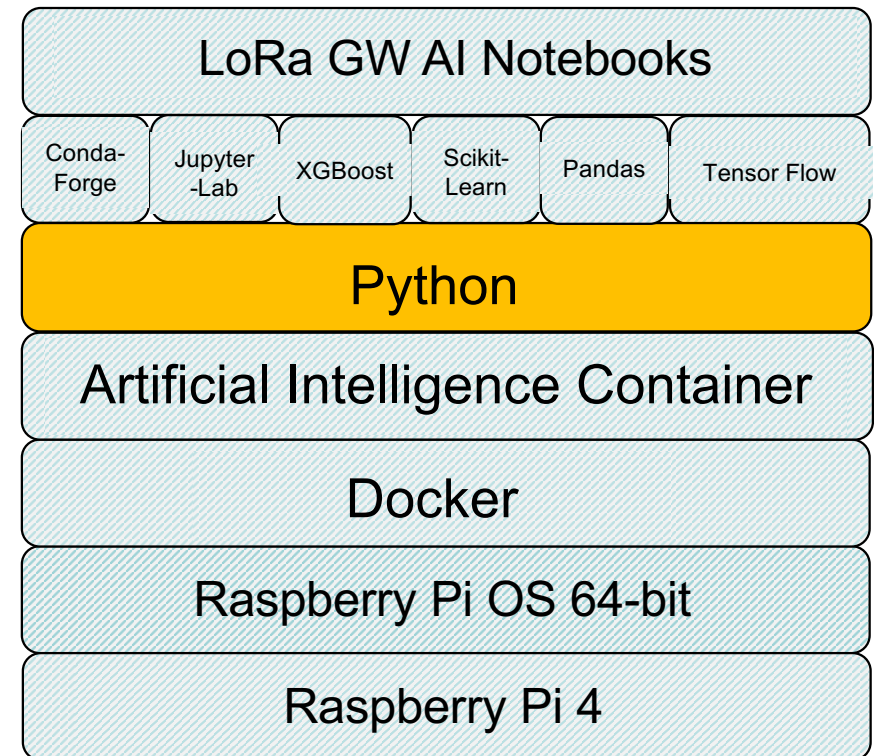
Artificial Intelligence Container

- ⦿ Container with key data science frameworks, libraries, and tools pre-installed.
- ⦿ Specifically designed for 64-bit Raspberry Pi.
- ⦿ AI Regression models from Zindi competition for soil humidity prediction.



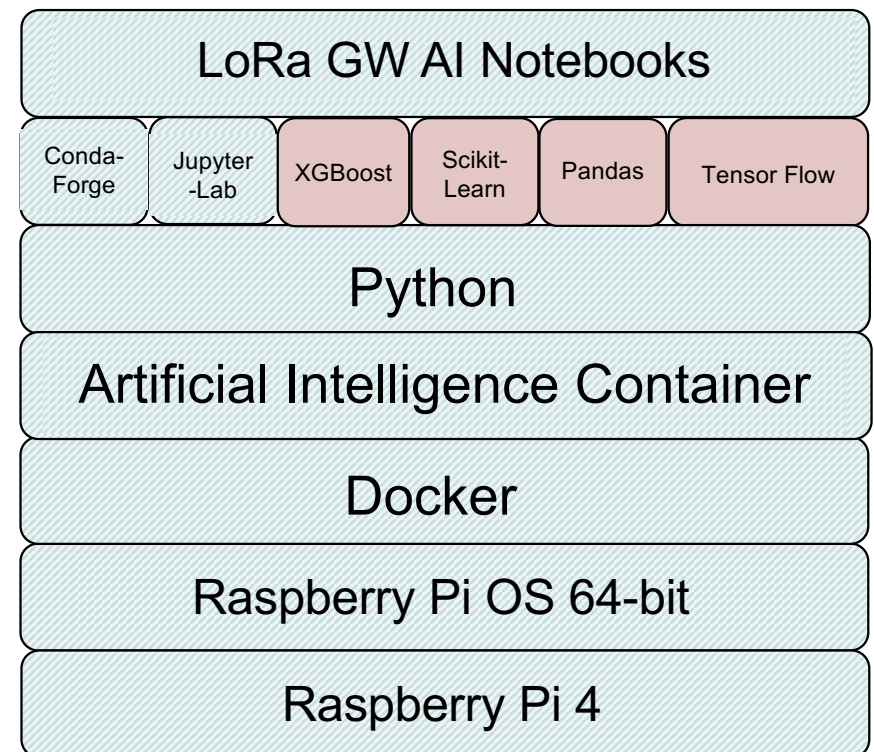
Runtime Engine

- Provides routines and functions that programs require as they performs tasks. For example:
 - Java Virtual Machine (JVM)
 - .NET Common Language Runtime (CLR)
 - Python Interpreter
- Python is fast becoming the language of choice for Artificial Intelligence scientists.



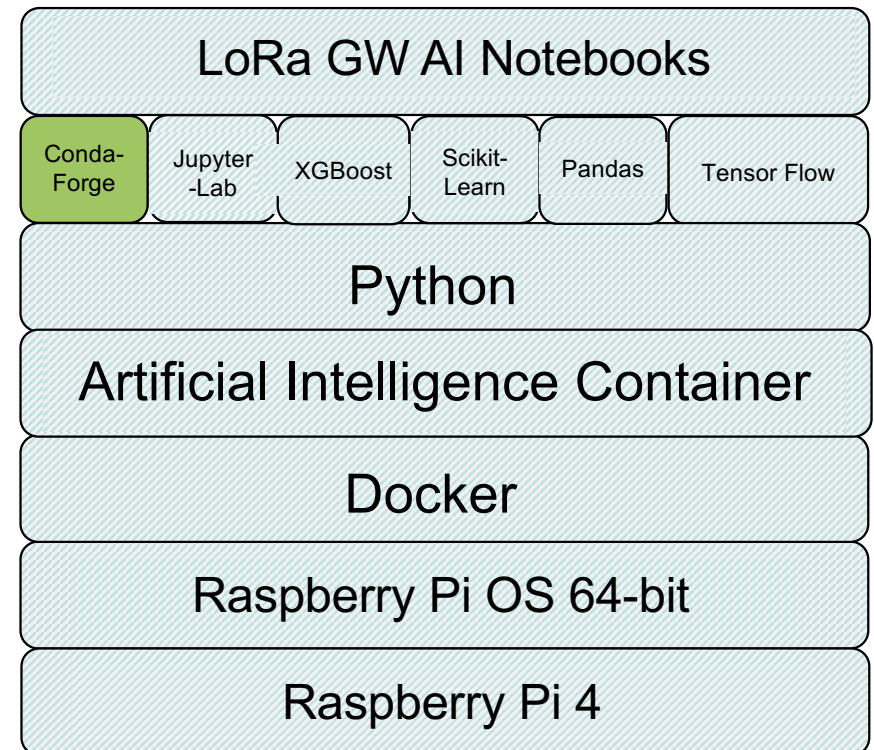
Packages

- ⦿ Packages integrate specific mathematical and data manipulation techniques, functions and procedures to enable Artificial Intelligence (AI) in programs.
- ⦿ Individual packages can be cobbled together building a larger program.
- ⦿ Allow algorithm scaling without getting into the details of the underlying AI algorithms.
- ⦿ Provide reusability of code.

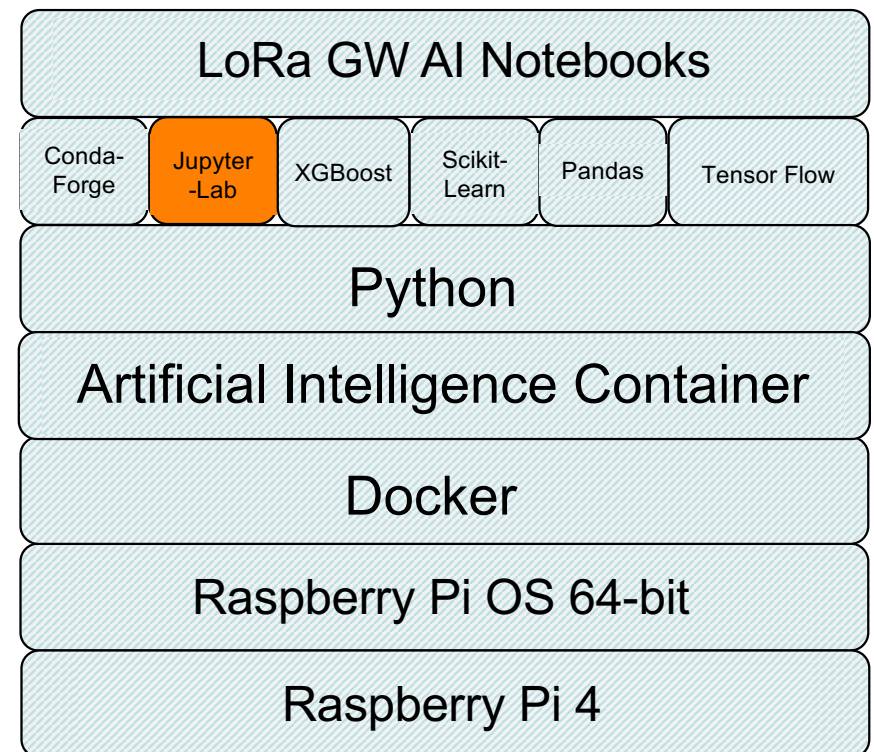


Package Manager

- ⦿ Software that finds, installs, updates and remove packages and their dependencies.
- ⦿ Conda-Forge is a package manager linked to a very active community effort, hosted in Github, that provides packages for a wide range of scientific software.

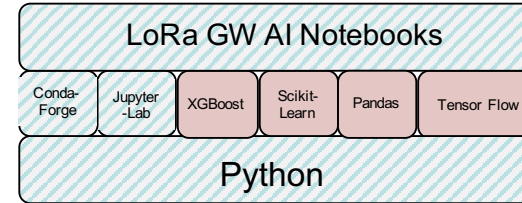


- Software for building programs combining common developer tools into a single graphical user interface (GUI).
- Jupyter-Lab is a web-based Interactive Development Environment (IDE) for Jupyter Notebooks, code, and data.
- Flexible: configure and arrange user interface to support workflows in data science, scientific computing, and Artificial Intelligence.



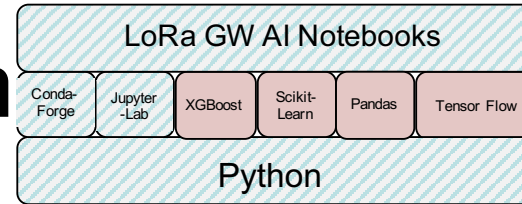


Package: pandas

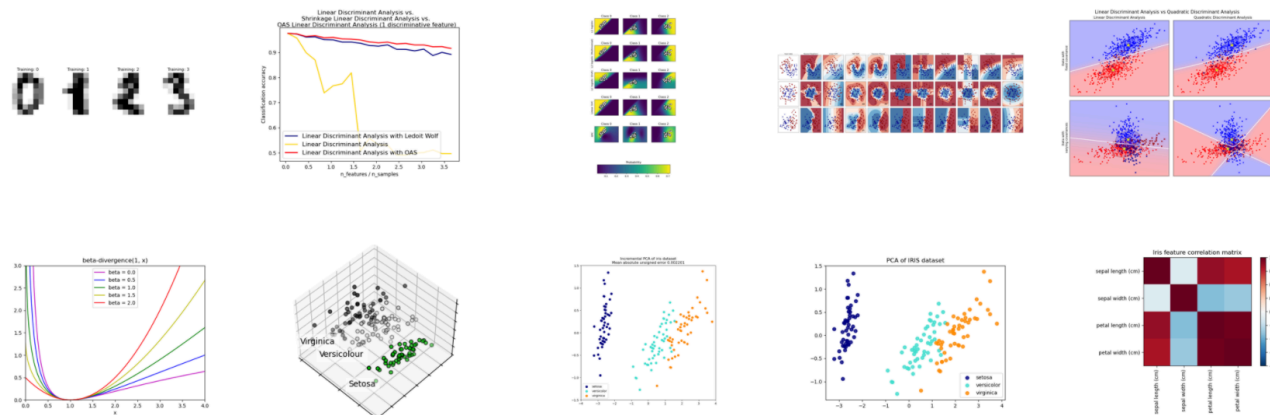


- ⦿ Python Data Analysis Library
- ⦿ pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language
- ⦿ pandas provides fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive
- ⦿ it aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.
- ⦿ https://youtu.be/_T8LGqJtuGc

Package: Scikit-Learn

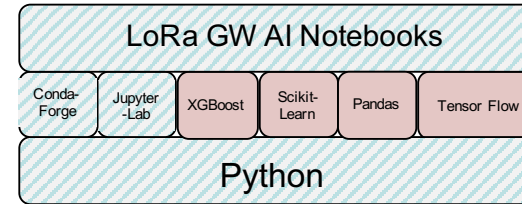


- Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning
- It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities
- https://scikit-learn.org/stable/auto_examples/index.html



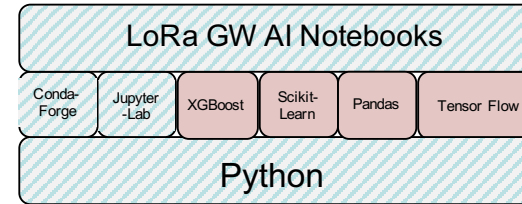


Package: XGBoost



- ⦿ eXtreme Gradient Boosting is an optimized open source implementation of the gradient boosting trees algorithm
- ⦿ Gradient Boosting is a supervised learning algorithm whose principle is to combine the results of a set of data and weaker models in order to provide a better prediction
- ⦿ XGBoost includes a large number of hyperparameters which can be modified and tuned for improvement
- ⦿ XGBoost is not part of Scikit-Learn but works perfectly with it
- ⦿ XGBoost behaves remarkably in machine learning competitions!
- ⦿ Source: XGBoost: The super star of algorithms in ML competition: <http://aishelf.org/xgboost/>

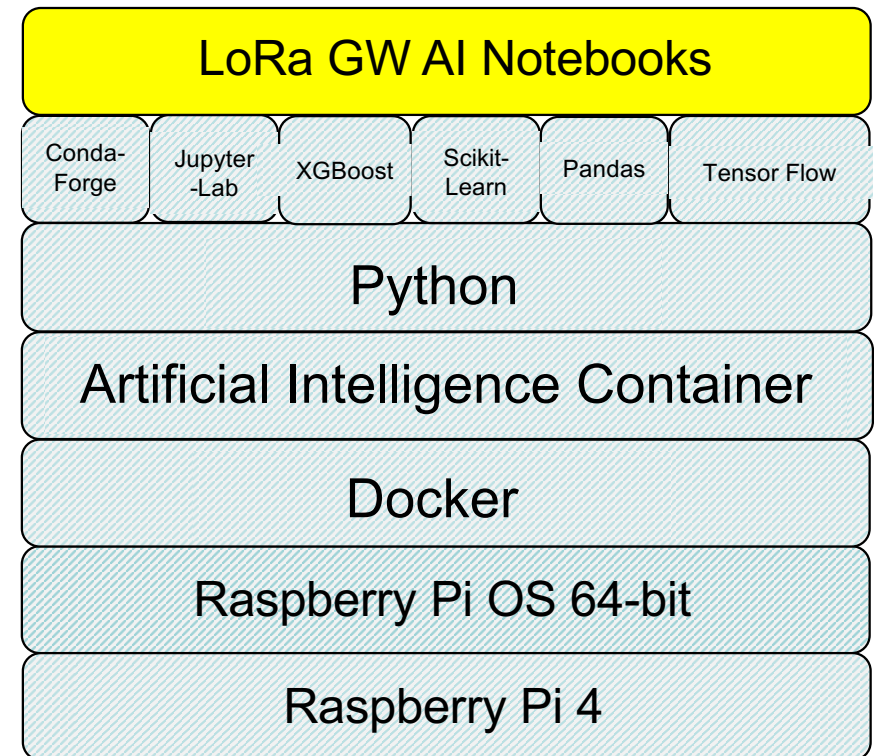
Package: TensorFlow



- ⦿ TensorFlow is an end-to-end open source platform for machine learning and deep learning
- ⦿ It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications
- ⦿ "TensorFlow is more of a low-level library whereas Scikit-Learn comes with off-the-shelf algorithms, e.g., algorithms for classification such as SVMs, Random Forests, Logistic Regression, ..." [https://stackoverflow.com/questions/61233004]
- ⦿ <https://www.tensorflow.org/>
- ⦿ <https://www.tensorflow.org/overview>

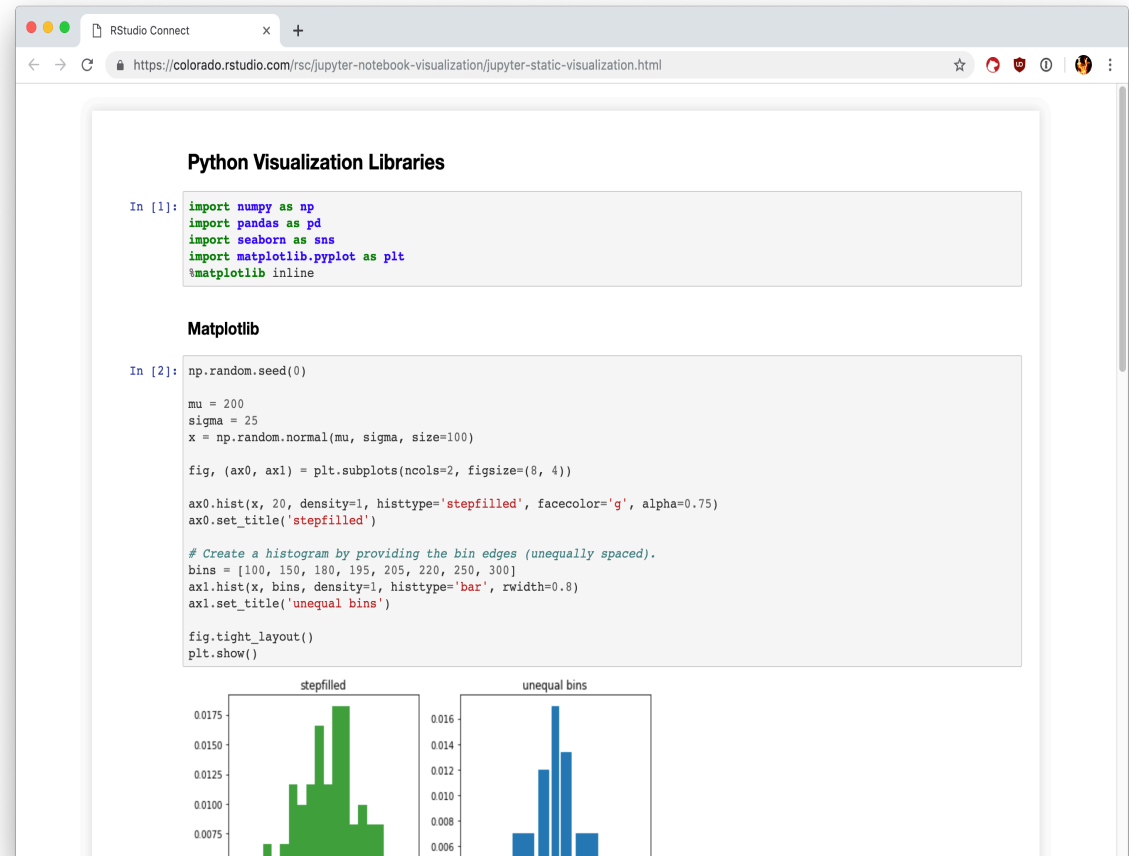
LoRa Gateway AI Notebooks

- ⦿ Front-end interface that allows the user to interact with Artificial Intelligence programs.
- ⦿ Based on Jupyter Notebook, a multimedia document that contains working code, figures and text.



Web application that allows to create and share documents that contain live code, equations, visualizations and narrative text.

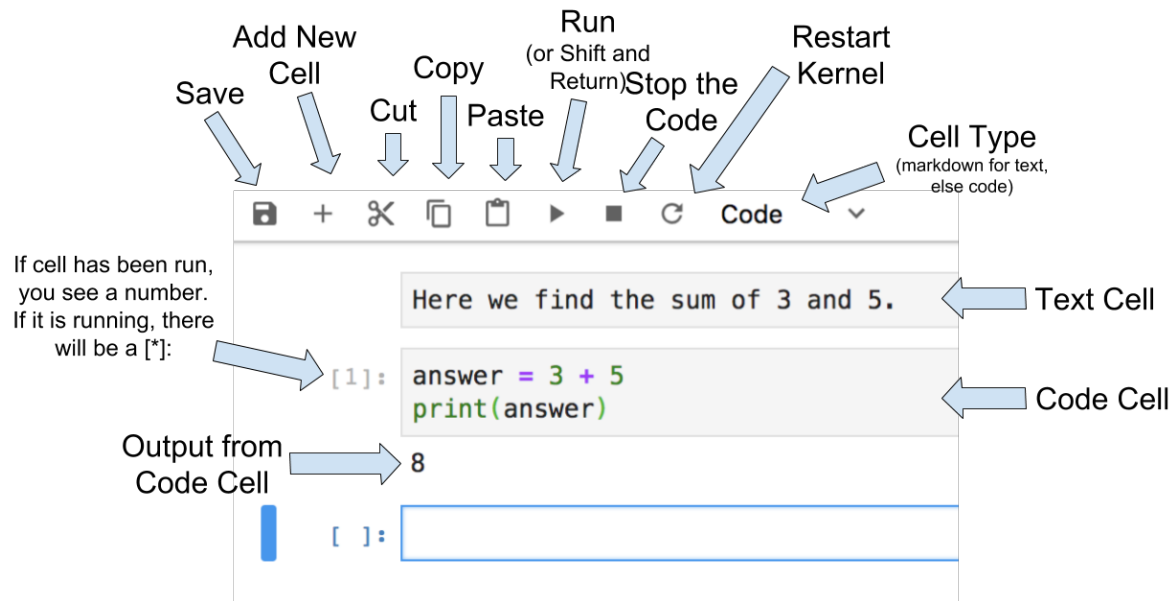
- ⦿ Data cleaning and transformation
- ⦿ Numerical simulation
- ⦿ Statistical modeling
- ⦿ Data visualization
- ⦿ Machine learning, and more.



Features

- ⦿ In-browser editing for code, with automatic syntax highlighting, indentation, and tab completion/introspection.
- ⦿ Execute code from the browser, with results of computations attached to the code which generated them.
- ⦿ Displaying result of computation using rich media representations, such as HTML, LaTeX, PNG, SVG, etc.
- ⦿ In-browser editing using Markdown markup language, which can provide commentary for the code, is not limited to plain text.
- ⦿ The ability to easily include mathematical notation within Markdown language using LaTeX.

Notebook Interface

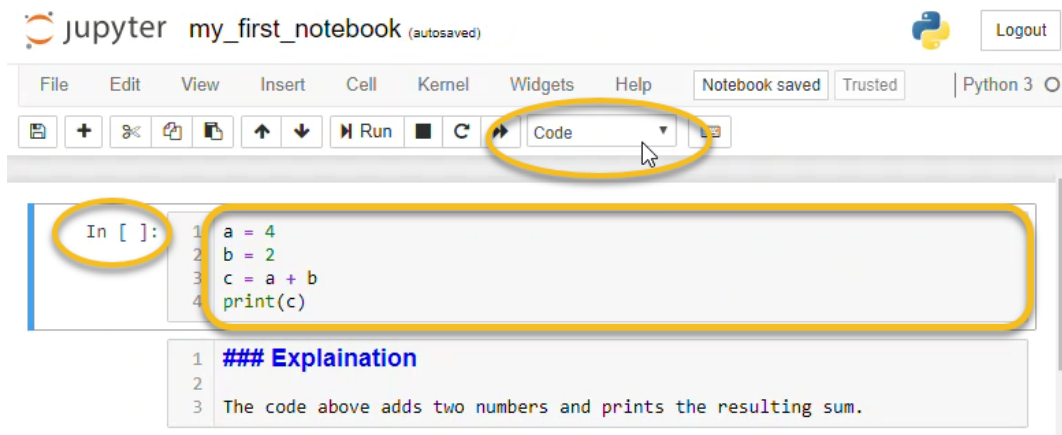


Jupyter notebook is comprised of a bunch of cells which are arrayed one after another in boxes below the menu items and buttons. There are 3 main types of cells:

- ⦿ Code cells
- ⦿ Output cells
- ⦿ Markdown cells

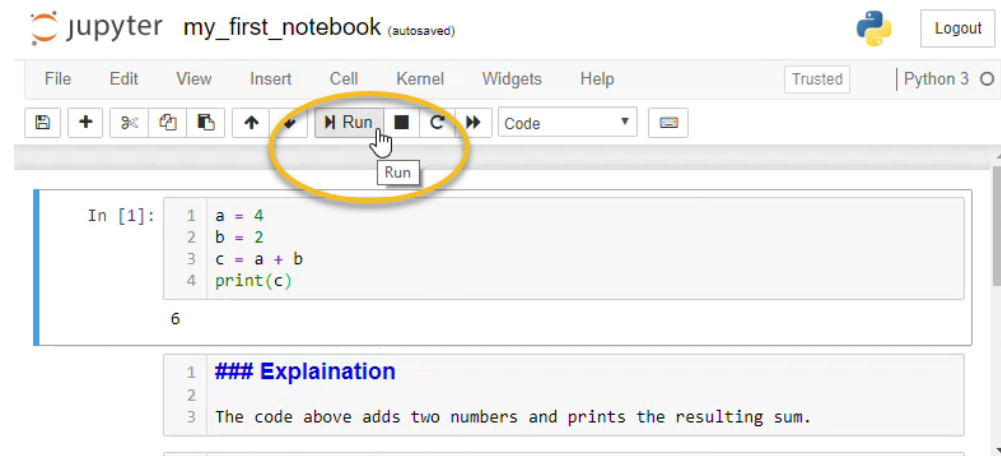
Code Cells

In code cells, write Python code, then execute the Python code and see the resulting output.



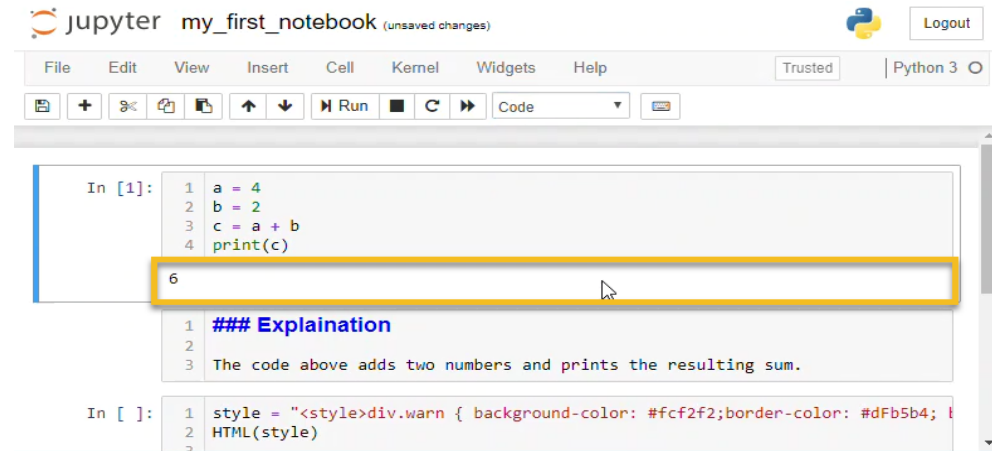
You can tell you are typing in a code cell because In []: is shown to the left of the cell and the cell-type drop-down menu shows **Code**.

To run the Python code in a code cell push the [Run] button.

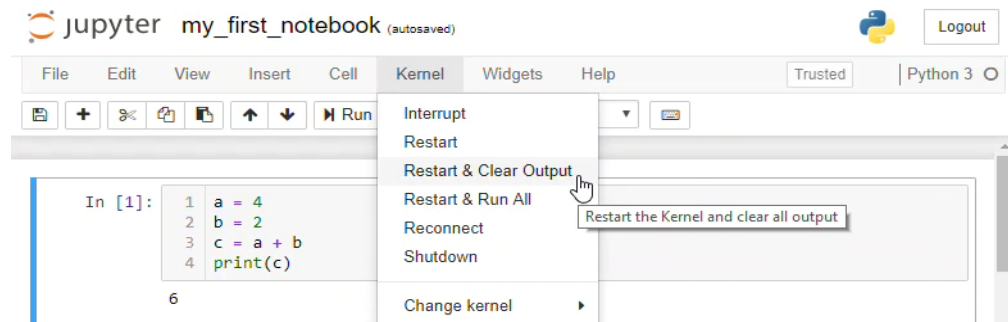


Output Cells

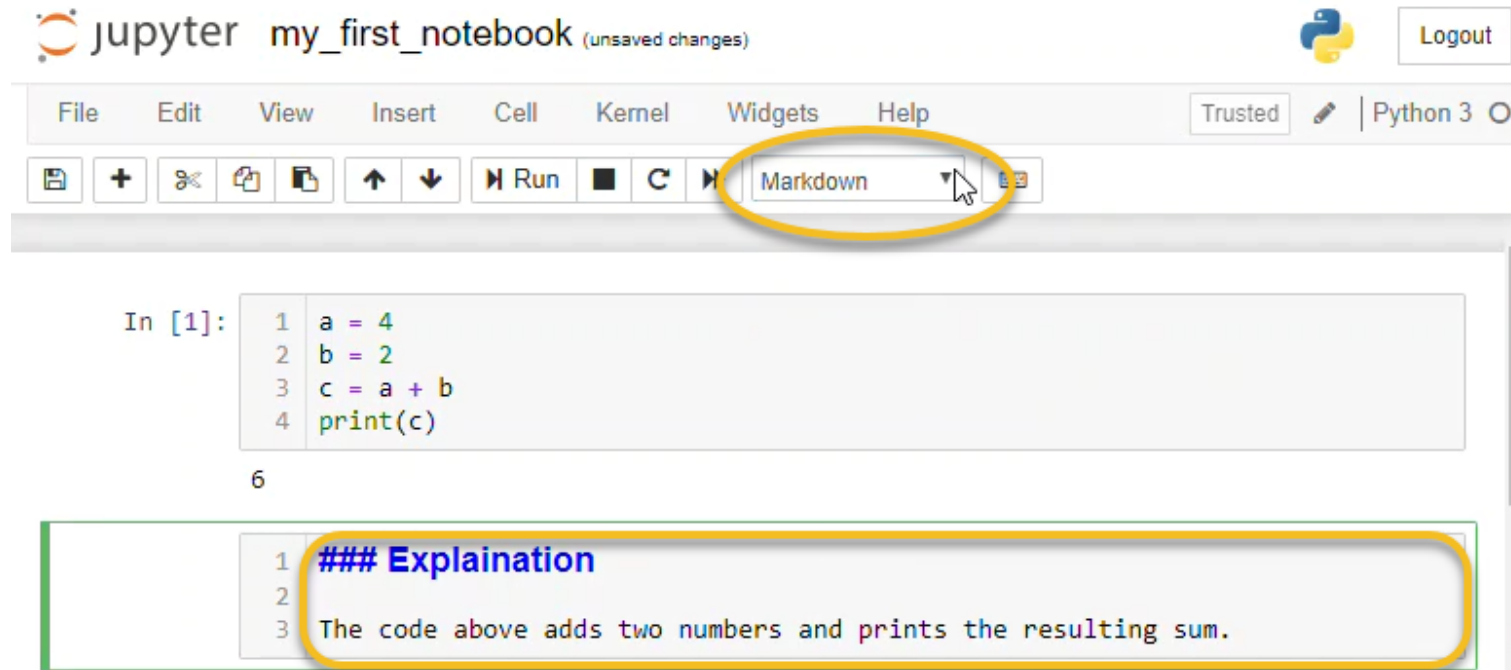
After a code cell is run, an output cell can be produced below the code cell. The output cell contains the output from the code cell above it. Not all code produces output, so not all code cells produce output cells. The results in output cells can't be edited. If a code cell produces plots, charts or images, these outputs are shown in output cells.



You can clear all the output cells and re-run code cells by selecting **[Kernel] --> [Restart Kernel and Clear Output]**.



Markdown Cells



The screenshot shows a Jupyter Notebook interface. At the top, there is a header with the Jupyter logo, the text "jupyter my_first_notebook (unsaved changes)", a Python logo, and a "Logout" button. Below the header is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A toolbar below the menu bar contains icons for file operations, a "Run" button, a "Markdown" button (circled in yellow), and other utility icons. The main area contains two cells. The first cell is a code cell with the following Python code:

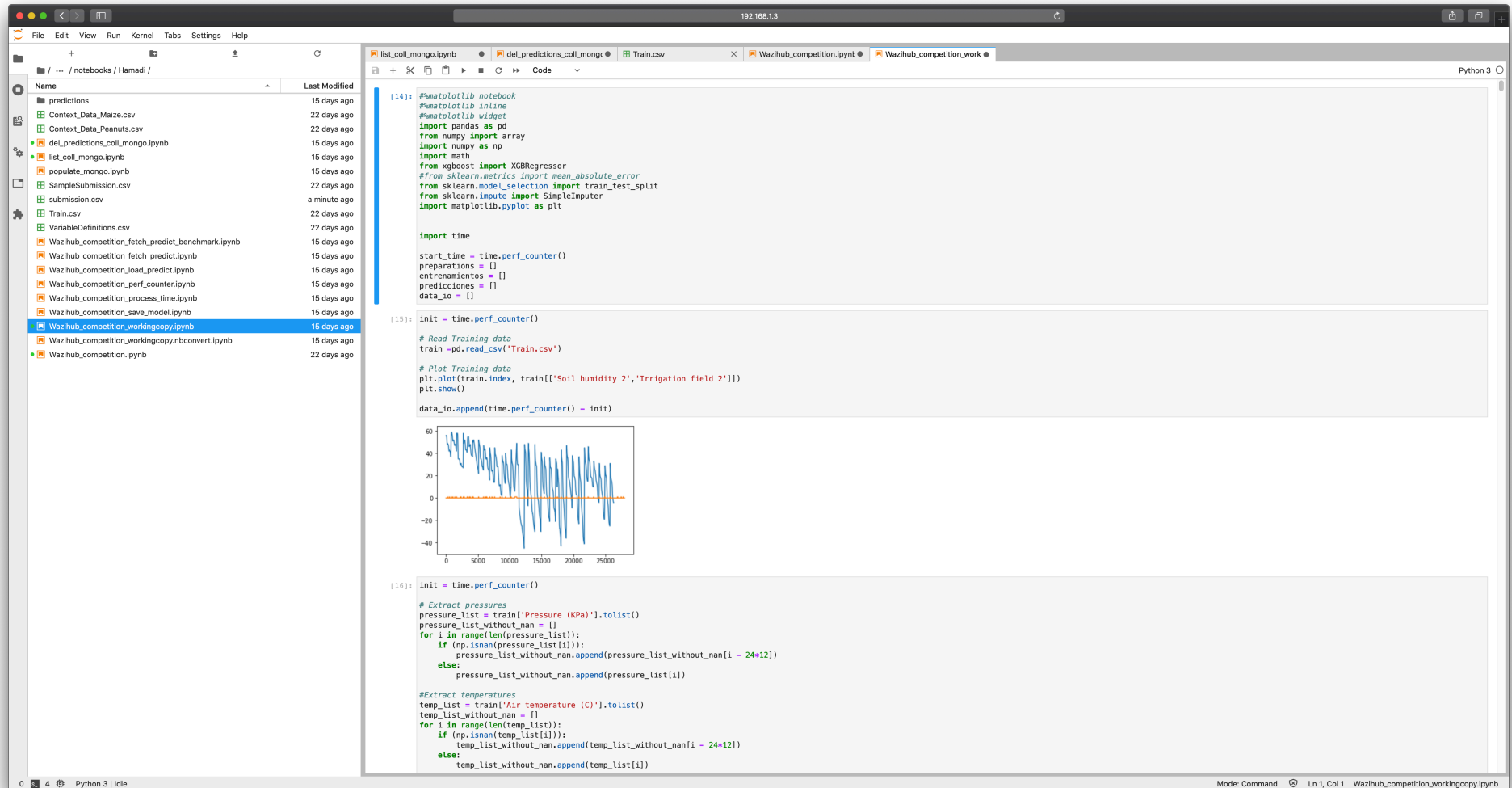
```
In [1]: 1 a = 4
        2 b = 2
        3 c = a + b
        4 print(c)
```

The output of the code cell is the number "6". The second cell is a markdown cell, outlined in green, containing the following text:

```
1 ### Explanation
2
3 The code above adds two numbers and prints the resulting sum.
```

Markdown cells don't contain Python code. Markdown cells contain text written in Markdown format. Text in markdown cells can be formatted to show bold or italic text. Tables, images, and lists can also be included in markdown cells.

Example



The screenshot shows a Jupyter Notebook environment with the following code and output:

```
[14]: #matplotlib notebook
#matplotlib inline
#matplotlib widget
import pandas as pd
from numpy import array
import numpy as np
import math
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt

import time

start_time = time.perf_counter()
preparations = []
entrenamientos = []
predicciones = []
data_io = []

[15]: init = time.perf_counter()

# Read Training data
train = pd.read_csv('Train.csv')

# Plot Training data
plt.plot(train.index, train[['Soil humidity 2', 'Irrigation field 2']])
plt.show()

data_io.append(time.perf_counter() - init)

[16]: init = time.perf_counter()

# Extract pressures
pressure_list = train['Pressure (KPa)'].tolist()
pressure_list_without_nan = []
for i in range(len(pressure_list)):
    if np.isnan(pressure_list[i]):
        pressure_list_without_nan.append(pressure_list[i - 24*12])
    else:
        pressure_list_without_nan.append(pressure_list[i])

#Extract temperatures
temp_list = train['Air temperature (C)'].tolist()
temp_list_without_nan = []
for i in range(len(temp_list)):
    if np.isnan(temp_list[i]):
        temp_list_without_nan.append(temp_list[i - 24*12])
    else:
        temp_list_without_nan.append(temp_list[i])
```

The plot output for [15] shows a time series plot with two data series: 'Soil humidity 2' (blue line) and 'Irrigation field 2' (orange line). The x-axis represents time from 0 to 25,000, and the y-axis represents values from -40 to 60. The blue line shows high-frequency oscillations between approximately 0 and 60, while the orange line is a constant horizontal line at 0.