

Software Development for Embedded ML

Michael Stanley

<http://sensip.asu.edu>

Mike.Stanley@ieee.org

Lecture materials can be accessed at
https://github.com/mstanley103/SenSIP_RET_2021

Lecture Mission Statement

Introduce you to embedded software development using the Arduino Nano 33 BLE Sense development board.

During the course of this lecture, we will review embedded code for streaming sensor data to your PC.

Today's Agenda

- Recap prior lecture
- Program & Debug ports
- Arduino Programming Environments
- Demos & Walkthroughs
 - “Blink” code walkthrough / demo
 - Sensor data logger code / walkthrough / demos
- Data collection topics
- Python (if time allows)

Last Time...

- Vocabulary
- Options for Training & Deployment
 - Cloud
 - PC
 - Embedded
- MCU Architecture
- What you need to know about your embedded board
- Introduction to Sensors
 - Scalar sensors: Temperature, Pressure, Humidity, Microphone
 - Vector sensors: Accelerometer, Magnetometer, Gyroscope
 - Vision

Example Hardware: The Arduino Nano 33 BLE Sense

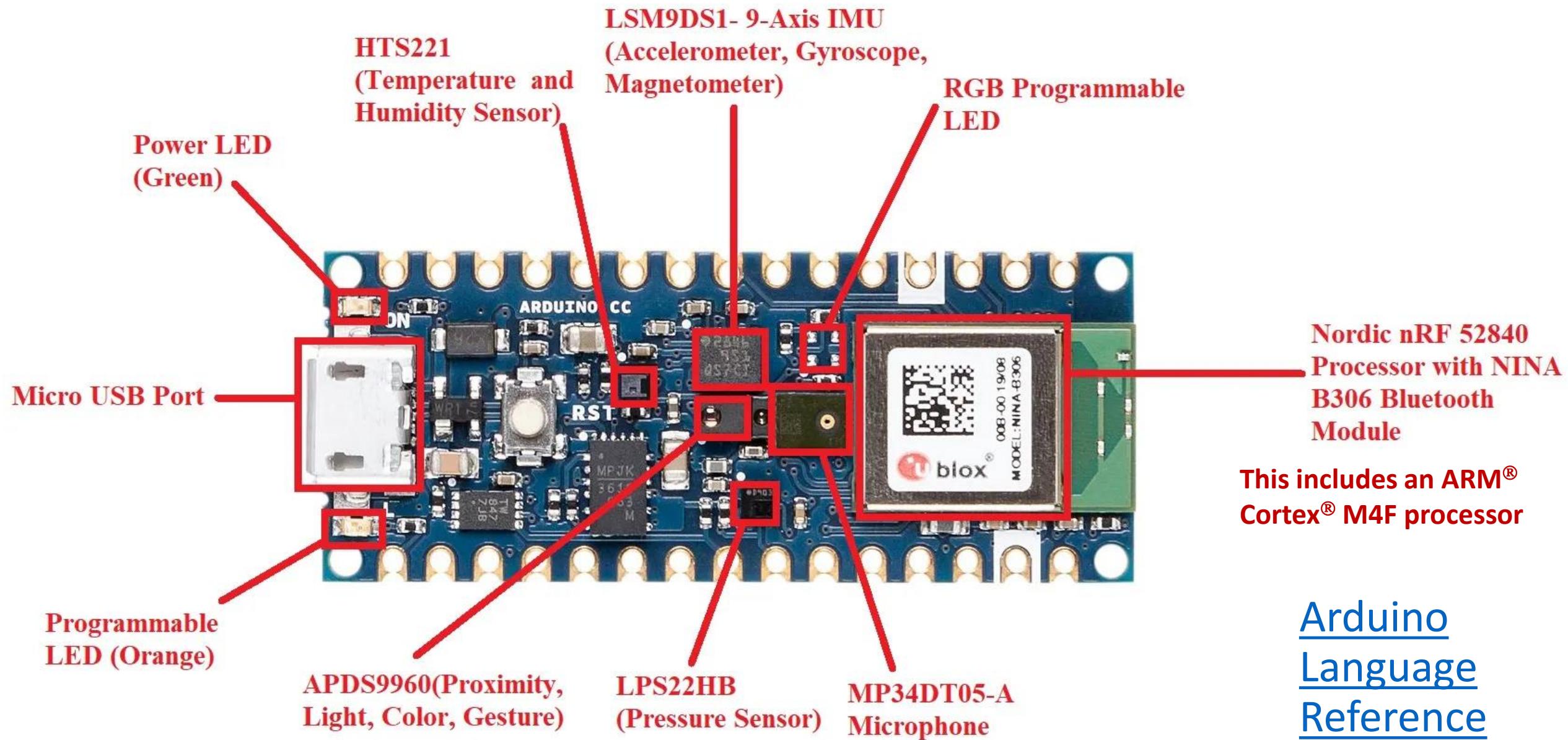
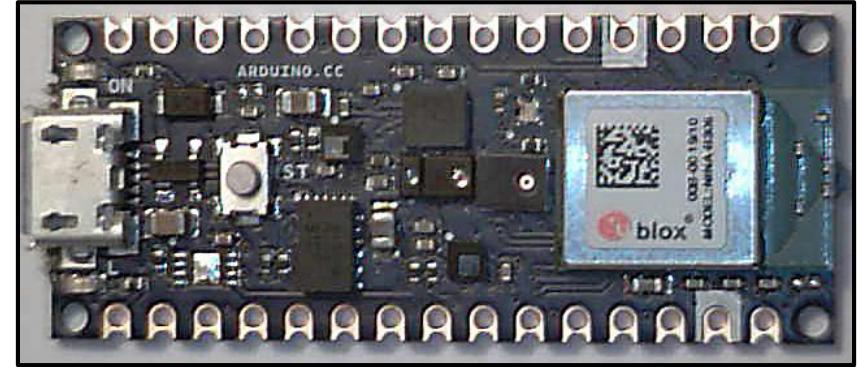


Image source: <https://www.etechnophiles.com/arduino-nano-33-ble-sense-pinout-introduction-specifications/>

Debug & Programming

Most MCUs support dedicated hardware debug ports which offer the ability:

- to start/stop the CPU
- single step through code
- examine code, variables and CPU registers
- set and clear breakpoints
- and much more
- program/erase flash memory



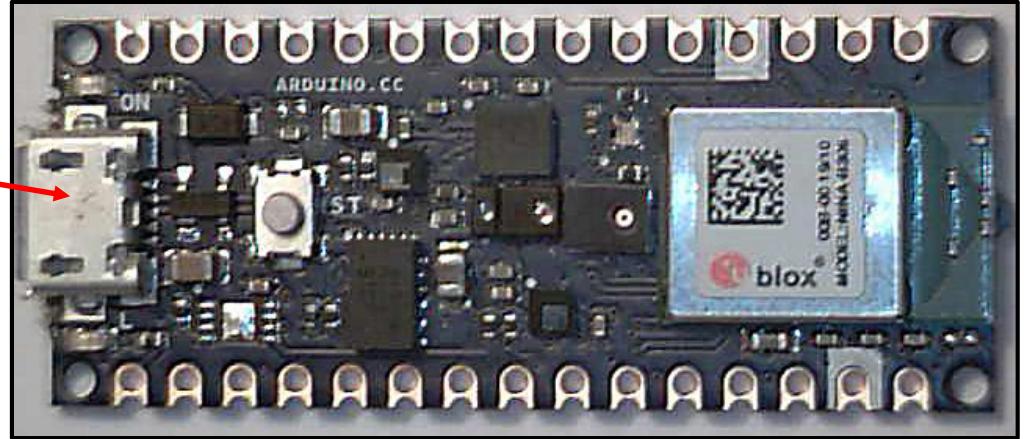
Because these consume pins (which have value and consume board space) and require an external hardware debug module, USB bootloaders have become common features for many development boards. This eliminates the need for an external, hardware-based, debug module.

The Arduino Nano 33 USB-based programming port does NOT support all of the features listed above.

WARNING: Physically, the microUSB connector is the part of the board most likely to fail. Use strain relief if you can.

There are several ways to program/erase the Arduino Nano 33 BLE Sense

Attach a microUSB cable from board to your PC



A USB bootloader program can be used to download code to your board as well as implement a virtual serial port. There are two versions:

1. **ArduinoCore Mbed (default)**
2. Adafruit nRF52 (for CircuitPython)

You need a hardware debug module attached to probe points on the bottom of the Nano to switch between them. Alternately you can use this as your primary debug port.



Hardware Debugging

To get the most from a modern software IDE, you will need to attach an external debug module such as a Segger J-link module to your board. You will need a soldering iron and a bit of time. Here are some useful links:

1. [Arduino Nano 33 BLE - SEGGER Wiki](#)
2. [Arduino Nano 33 BLE/IoT Custom Debugging - Hackster.io](#)
3. [SWD Breakout and Breadboard for Nano 33 BLE Sense | Thomas J. Petz, Jr. \(tjpetz.com\)](#)
4. [Arduino Nano 33 IoT Debugging - Arduino Project Hub](#)
5. [J-Link EDU Mini - SEGGER - The Embedded Experts](#)



image source: [4]

Programming Environments for the Arduino Nano 33



> EDITOR

Sketchbook

Examples

Libraries

Monitor

Help

Preferences

Features usage

Arduino IDE for Embedded Development

NEW SKETCH



SEARCH SKETCHBOOK



ORDERING BY LAST MODIFIED

sketch_jun3a

Blink_copy

sketch_jun3a

UPGRADE PLAN



Arduino Nano 33 BLE at CO...

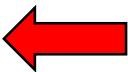
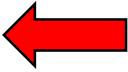


SHARE

```
1 // Arduino_TensorFlowLite -  
2 #include <TensorFlowLite.h>  
3  
4 /*  
5 */  
6  
7 void setup() {  
8 }  
9  
10 void loop() {  
11 }  
12  
13 }  
14  
15 }
```

The Arduino tools

- are “hobbyist” grade
 - Optimized for ease of setup and use
 - Lack sophisticated debug capabilities
- structures applications into “setup” and “loop” functions
- Supported for Windows, MAC or Linux
- <https://www.arduino.cc/en/main/software>

See: [Getting started with the Arduino Nano 33 BLE Sense | Arduino](#)

Libraries you may want to use with the Arduino Nano 33 BLE Sense

- <https://www.arduino.cc/en/Reference/ArduinoLSM9DS1>
- <https://www.arduino.cc/en/Reference/PDM> (appears to be preinstalled for Arduino)
- <https://www.arduino.cc/en/Reference/ArduinoAPDS9960>
- <https://www.arduino.cc/en/Reference/ArduinoLPS22HB>
- <https://www.arduino.cc/en/Reference/ArduinoHTS221>
- https://www.arduino.cc/reference/en/libraries/arduino_tensorflowlite/

See: [Getting started with the Arduino Nano 33 BLE Sense | Arduino](#)

Port Addresses

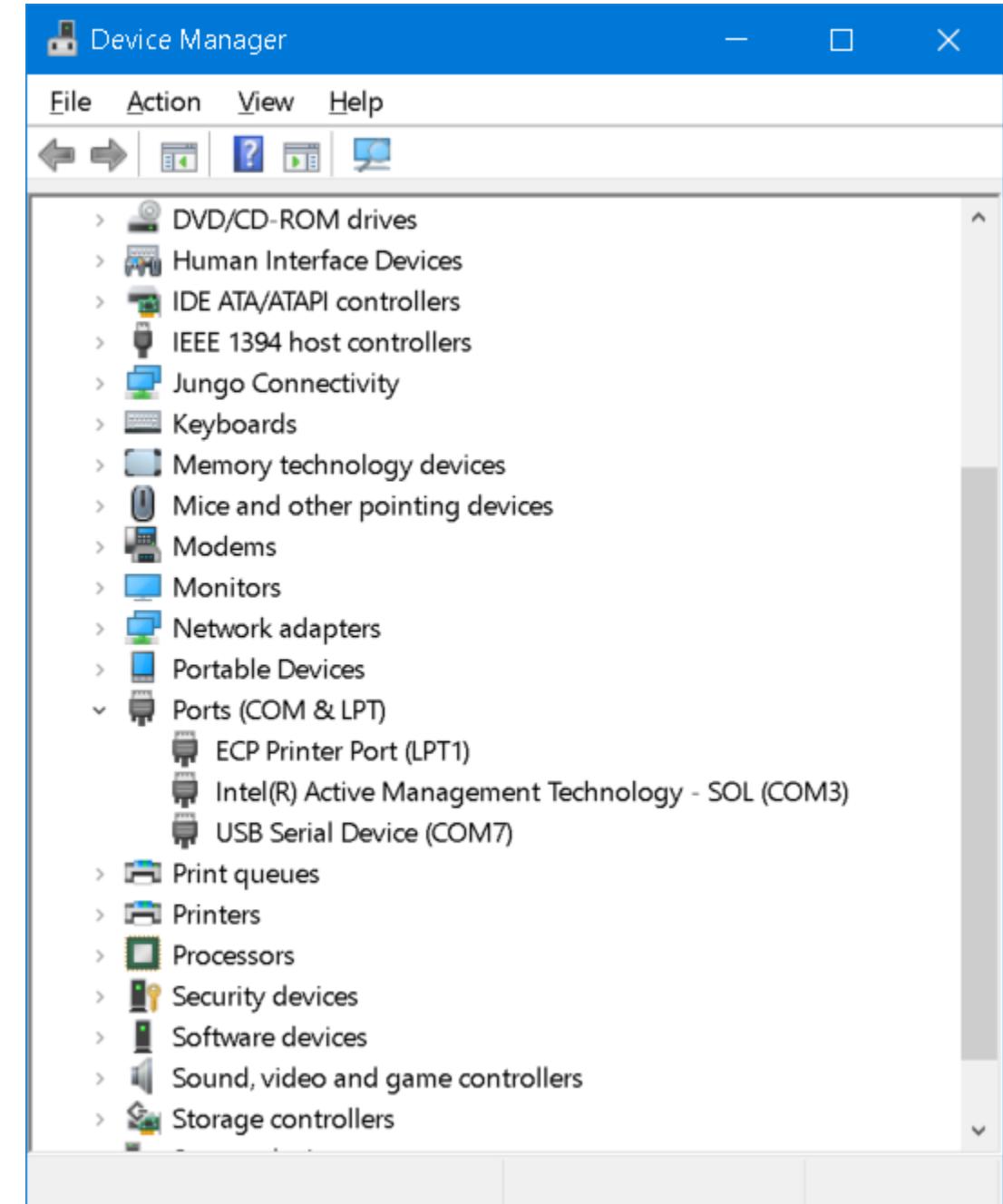
When using the Mbed bootloader, the board will enumerate with one of two serial port addresses:

- address A = address used to program the device
- address A+1 = mission mode port ID

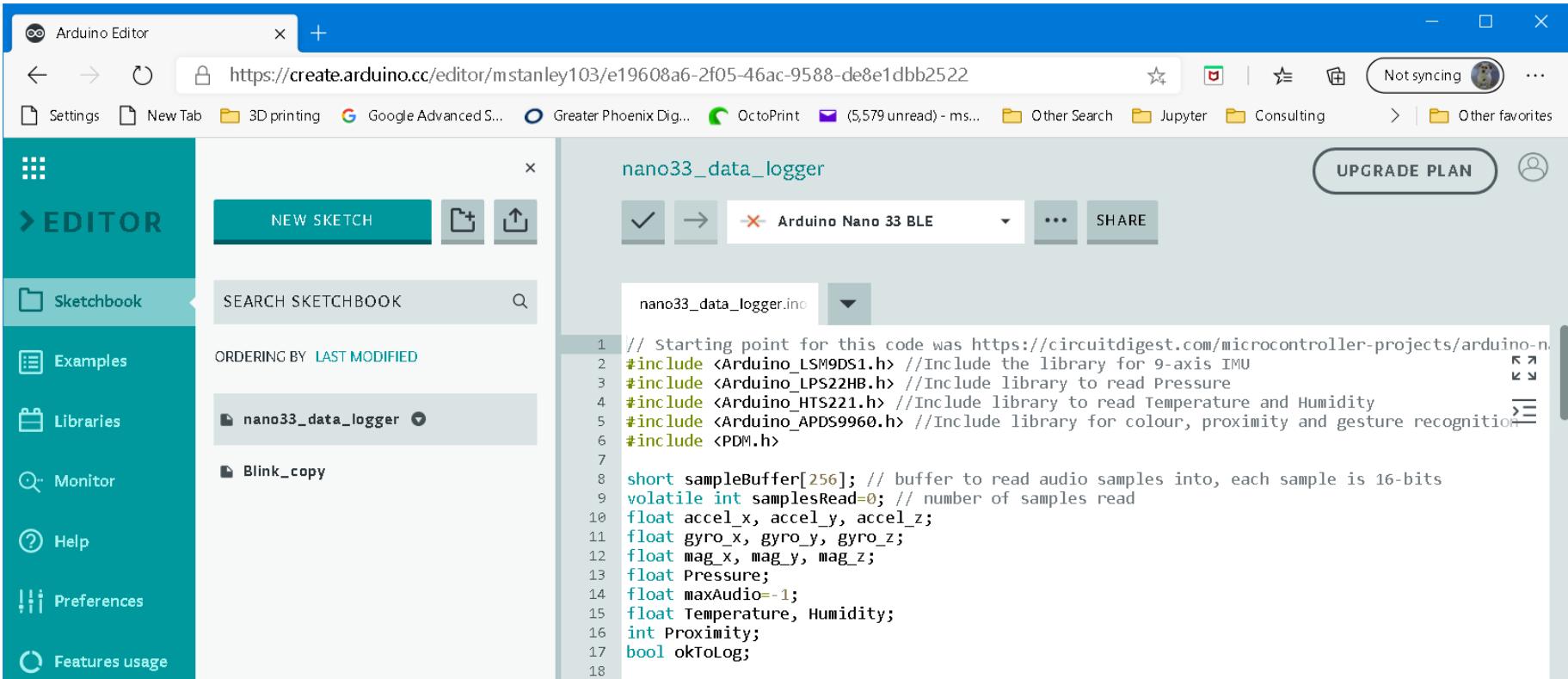
You will need to select the proper port address in the Arduino IDE.

This two address scheme is somewhat unique to the Arduino Nano 33 family. Other ARM devices running Mbed or OpenSDA interfaces typically only consume one address.

Clicking the reset button twice in fast succession will put you into bootloader mode. Once should get you mission mode.

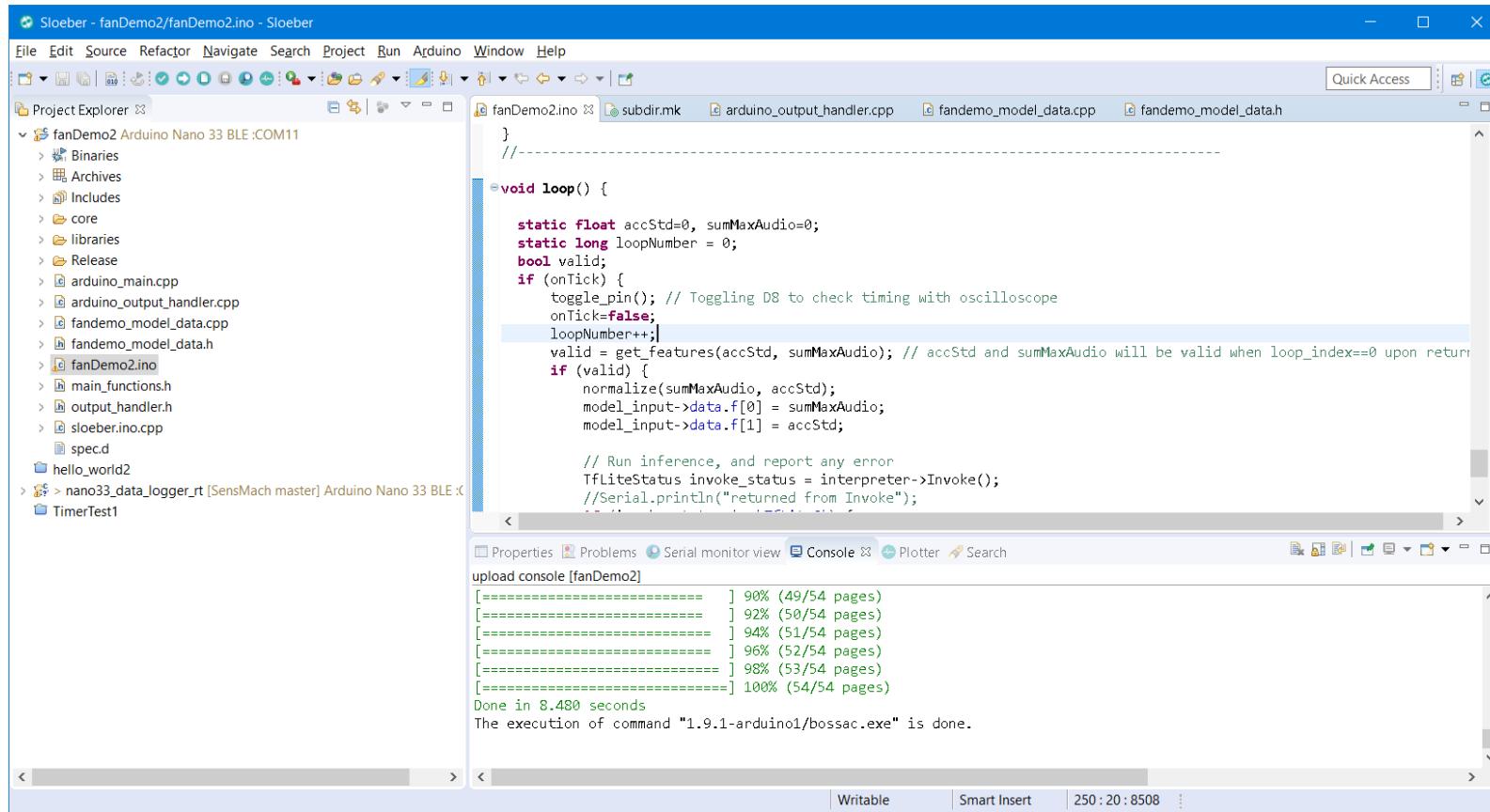


A Cloud-based Version is also Available



You will need to install the [Arduino Create Plugin](#) on your PC to communicate with your board

More Sophisticated Coders Will Prefer Eclipse



The Sloeber plugin for Eclipse provides:

- Full C/C++ language support
 - Faster build times
 - Code tracing
 - Compatibility with the standard Arduino libraries
 - Built in serial port monitor
 - Hardware debug support (may require an external debug module and board modification)
-
- <https://eclipse.baeyens.it/>

A relatively new option is the Visual Micro extension to Visual Studio

([VisualMicro - Arduino IDE For Visual Studio](#))

The screenshot shows the Visual Studio IDE interface with the Visual Micro extension installed. The main window displays an Arduino sketch named 'ColorSensor.ino' in the Global Scope. The code implements the setup() and loop() functions for an APDS9960 color sensor, including initializing the serial port and turning on white LED light sources. The Solution Explorer panel shows the project structure for 'ColorSensor' with files like 'ColorSensor.vsarduino.h', 'ColorSensor.ino', and 'Header Files'. The Properties panel is also visible. The bottom status bar indicates the project is 'ColorSensor' on board 'Arduino...', with line 25, column 27, character 27, and mode INS.

```
16 void setup() {
17     Serial.begin(9600);
18     while (!Serial);
19
20     if (!APDS.begin()) {
21         Serial.println("Error initializing APDS9960 sensor.");
22     }
23     // Turn on White LED light source
24     digitalWrite(LED_R, LOW);
25     digitalWrite(LED_G, LOW);
26     digitalWrite(LED_B, LOW);
27
28 }
29
30 void loop() {
31
32     // check if a color reading is available
33     while (! APDS.colorAvailable()) {
```

The best of the lot appears to be PlatformIO

The screenshot shows the PlatformIO Home interface within the Visual Studio Code environment. The interface is dark-themed with orange and white highlights.

Left Sidebar (EXPLORER):

- OPEN EDITORS:** PIO Home
- WORKSPACE (WORKSPACE):**
 - > test
 - > Blink
 - > nano33_data_logger
 - > .pio
 - > vscode
 - > include
 - > lib
 - > src
 - nano33_data_logger.ino
 - > test
 - .gitignore
 - platformio.ini

Center Content:

 - Welcome to PlatformIO** with a large orange alien logo.
 - Core [5.1.1] · Home [3.3.4]
 - Quick Access:**
 - + New Project
 - Import Arduino Project
 - Open Project
 - Project Examples
 - Recent News:**
 - June 03 PlatformIO: PlatformIO Open Source May Updates are here! 🎉
 - May 31 PlatformIO: #LearnEmbedded 🎓 Explore the benefits of Memory Protection Units for robust and reliable firmware in a great
 - May 27 PlatformIO: Meet the new Blynk IoT platform!

Bottom Status Bar:

0 △ 4 ⌂ ✓ → ⌂ Default (Blink)

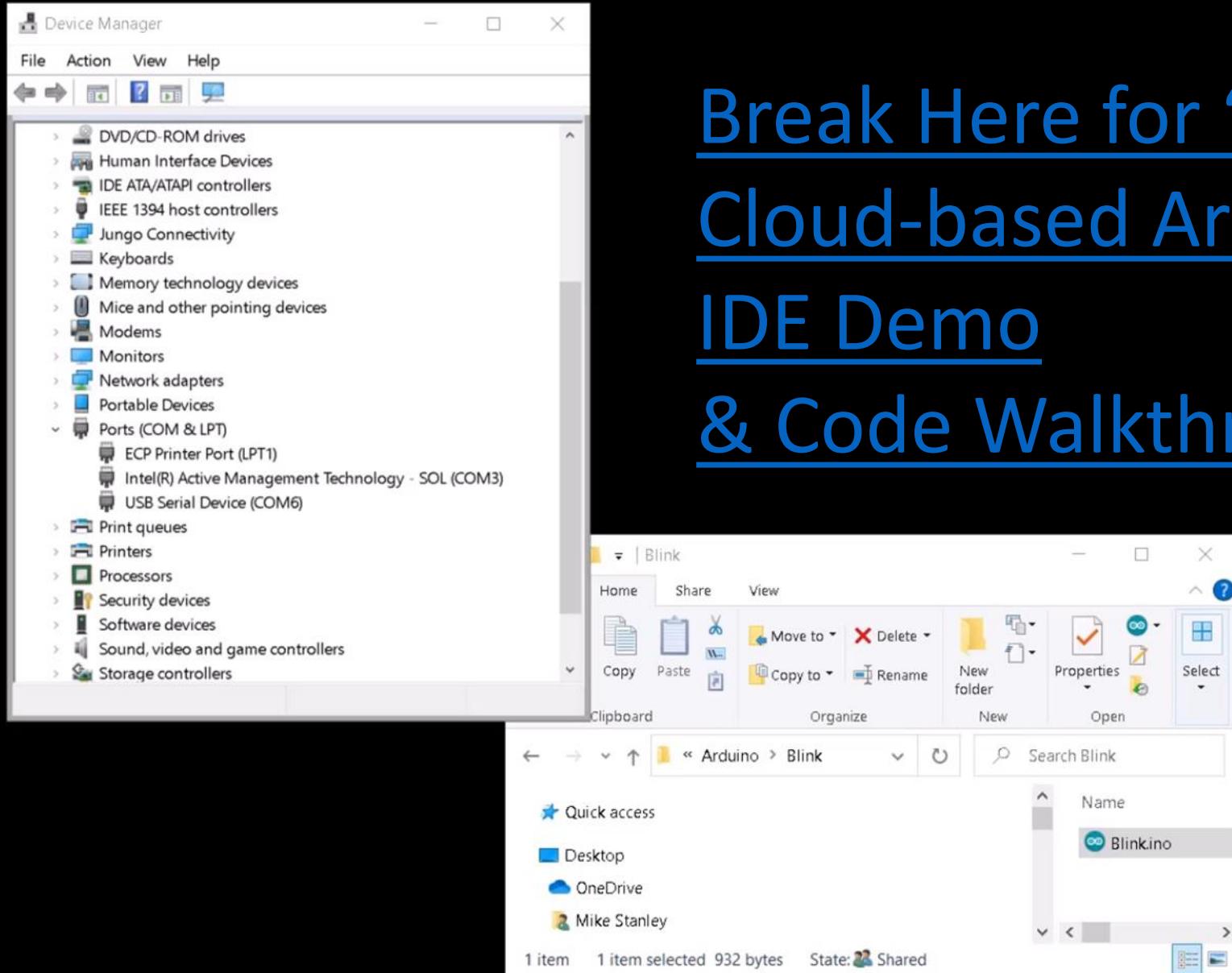
PlatformIO

- PlatformIO (<https://platformio.org/>) is available on a wide variety of hardware platforms
- As shown here, it is a plugin run within Microsoft Visual Studio Code
- PlatformIO is written in (and requires) pure Python
- It has code autocomplete, function cross-referencing, fast build and download and memory profiling features
- Look for your projects under Documents\PlatformIO\Projects

```
[env:stable]
platform = nordicnrf52
board = nano33ble
framework = arduino
monitor_speed = 115200
; change MCU frequency
board_build.f_cpu = 64000000L
; change microcontroller
board_build.mcu = nrf52840
```

You will need the settings above included in your projects platformio.ini file for proper operation with your Arduino Nano 33. Note that the monitor_speed setting must match the baud rate you initialize your serial port to.

IDE Demos



Break Here for “Blink”

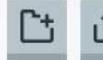
Cloud-based Arduino

IDE Demo

& Code Walkthrough

► EDITOR

NEW SKETCH



SEARCH SKETCHBOOK



ORDERING BY LAST MODIFIED

> ColorSensor (1)

► nano33_data_logger ◉

► nano33_data_logger_r

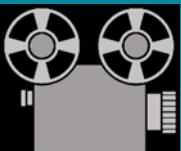
Q Monitor

Reference

Help

Preferences

Features usage



Double-Click Nano 33 reset pin before starting

A A

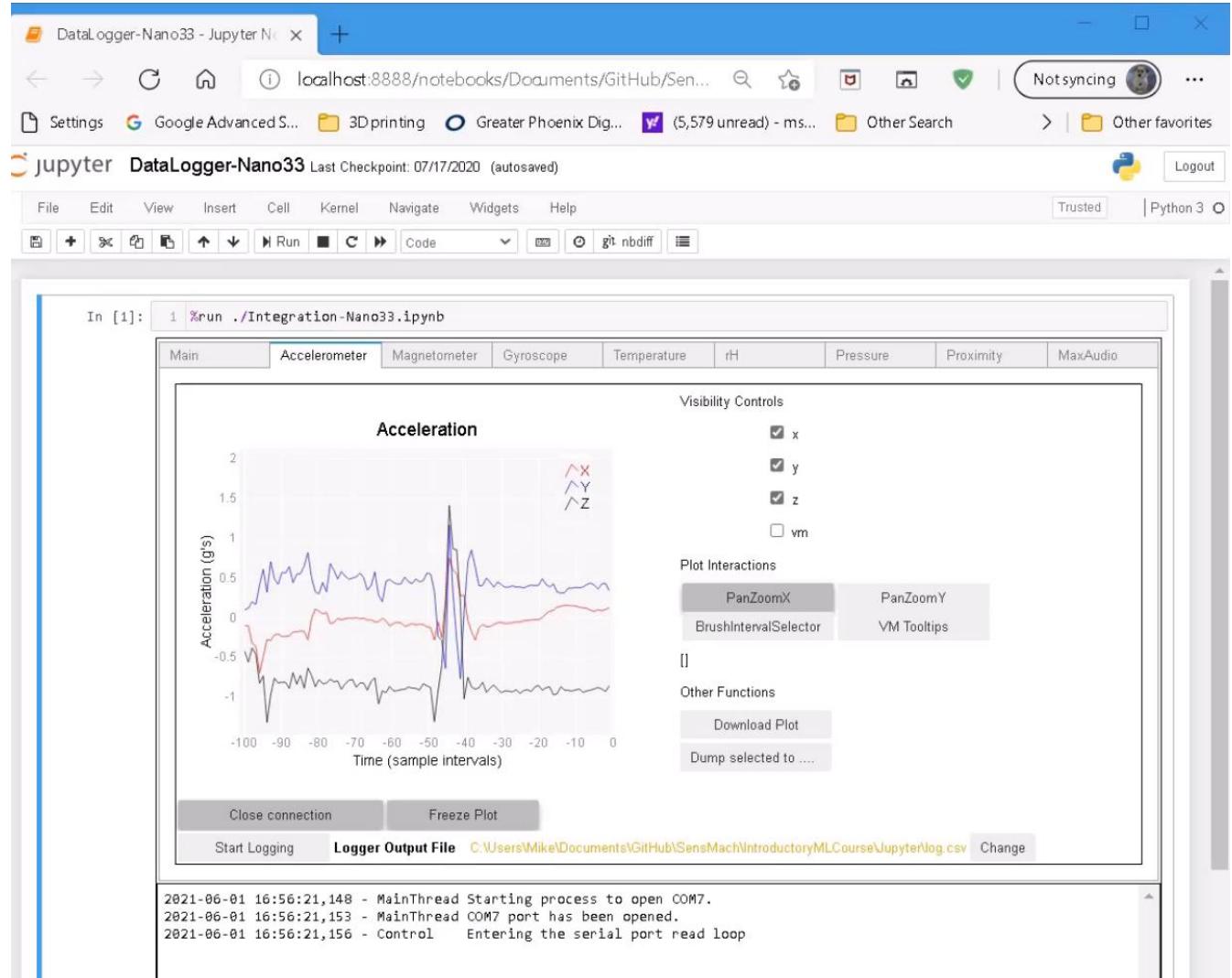
Break here for PlatformIO Demo & Walkthrough (nano33_data_logger)

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Editor Title:** main.cpp - nano33_data_logger - Visual Studio Code
- Explorer Sidebar:** Shows the project structure:
 - OPEN EDITORS: PIO Home, main.cpp
 - NANO33_DATA_LOGGER: .pio, .vscode (selected), c_cpp_properties.json, extensions.json, launch.json, include, lib, src (main.cpp), test, .gitignore, platformio.ini
- Editor Content:** The main.cpp file contains C++ code for initializing sensors and logging data to Serial. The code includes sections for IMU, BARO, HTS, and a loop to read audio samples.
- Terminal Output:** The terminal window displays sensor data in a CSV-like format, starting with:

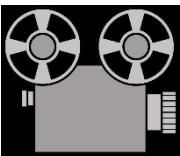
```
-0.18, -0.02, 0.97, 4.21, 1.10, -0.12, -8.97, 21.45, 17.30, 96.27, 31.99, 29.13, 247, 16.00
-0.18, -0.02, 0.97, 3.97, 1.04, -0.12, -8.90, 21.03, 18.01, 96.26, 31.97, 29.14, 250, -1.00
-0.18, -0.02, 0.97, 4.27, 1.10, -0.24, -8.73, 20.52, 17.58, 96.27, 31.94, 29.10, 248, 90.00
-0.18, -0.02, 0.97, 3.91, 1.04, -0.12, -9.11, 21.04, 17.15, 96.27, 31.97, 29.08, 247, 185.00
-0.18, -0.02, 0.96, 4.21, 1.22, -0.12, -9.47, 21.25, 17.53, 96.27, 31.97, 29.25, 247, 28.00
-0.18, -0.02, 0.97, 4.33, 0.98, -0.12, -9.35, 20.50, 17.53, 96.26, 32.01, 29.04, 249, 6.00
-0.18, -0.02, 0.97, 4.27, 1.10, -0.18, -8.95, 21.02, 17.42, 96.28, 32.01, 29.19, 248, 32.00
```
- Status Bar:** Includes icons for file operations, a progress bar, and text indicating the current file is Default (nano33_data_logger). It also shows the line and column numbers (Ln 41, Col 35), spaces used (Spaces: 2), and encoding (UTF-8).

Collecting Training Data



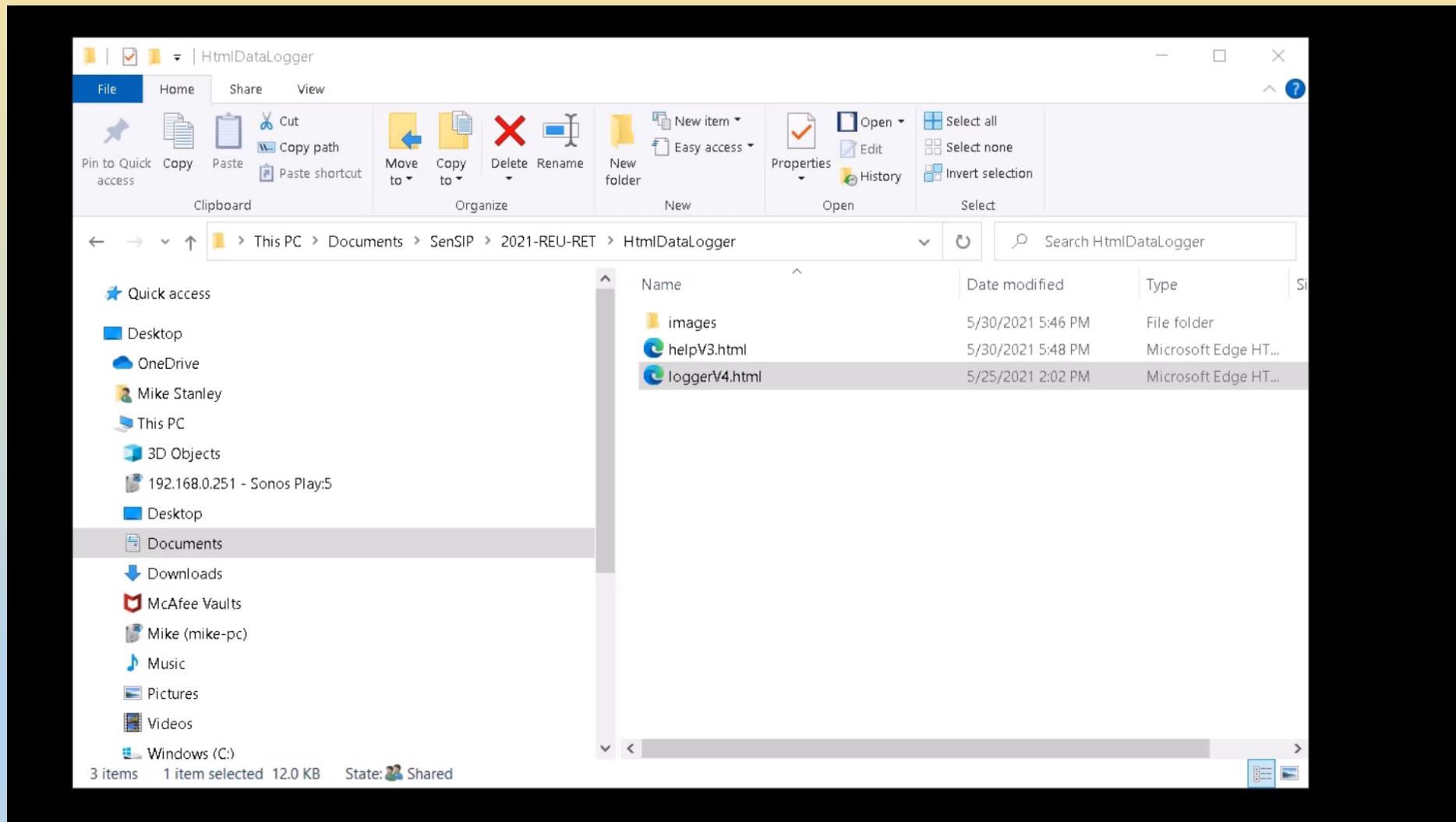
Data loggers can:

- give you the ability to visually monitor signals and look for patterns.
- run timed data logging sessions
- easily record session metadata right in the log files
- easily create graphics for reports
- upload data to cloud servers
- manage your data sets
- monitor MCU CPU & memory utilization
- manage network security and handshaking



This version is pure HTML/Javascript

- No graphics
- Depends upon experimental feature in Microsoft Edge Browser
- Requires only 2 HTML files.
- Files can be locally or server hosted



Data Collection Topics

- Sensors can be noisy, and they can be impacted by other PCB components
- Ideally you would like to collect sensor samples as fast as possible
- ... and at a regular (fixed) interval
- If you have multiple sensors, you would like to sample them simultaneously
- All of the above are complicated by the fact that there are huge sampling speed differences between sensor types. For example:
 - A high resolution pressure sensor may take 1 second per conversion
 - A high speed microphone might be sampling at 44KHz or higher
 - If you have more than one microphone with beamforming and filters running, things get more complicated.
- You also need to consider MCU and communications bandwidths.

Inspire the future generation of



If time allows...

Anaconda

The screenshot shows a web browser window displaying the Anaconda Individual Edition product page. The URL in the address bar is <https://www.anaconda.com/products/individual>. The page features a blue header with the Anaconda logo and navigation links for Settings, Products (which is underlined in green), Pricing, Solutions, Resources, Blog, and Company. A 'Get Started' button is visible in the top right. The main content area is titled 'Your data science toolkit' and describes the Individual Edition as the easiest way to perform Python/R data science and machine learning. It mentions over 20 million users worldwide and the availability of thousands of open-source packages and libraries. A 'Download' button is located at the bottom left. The browser interface includes standard controls like back, forward, and search.

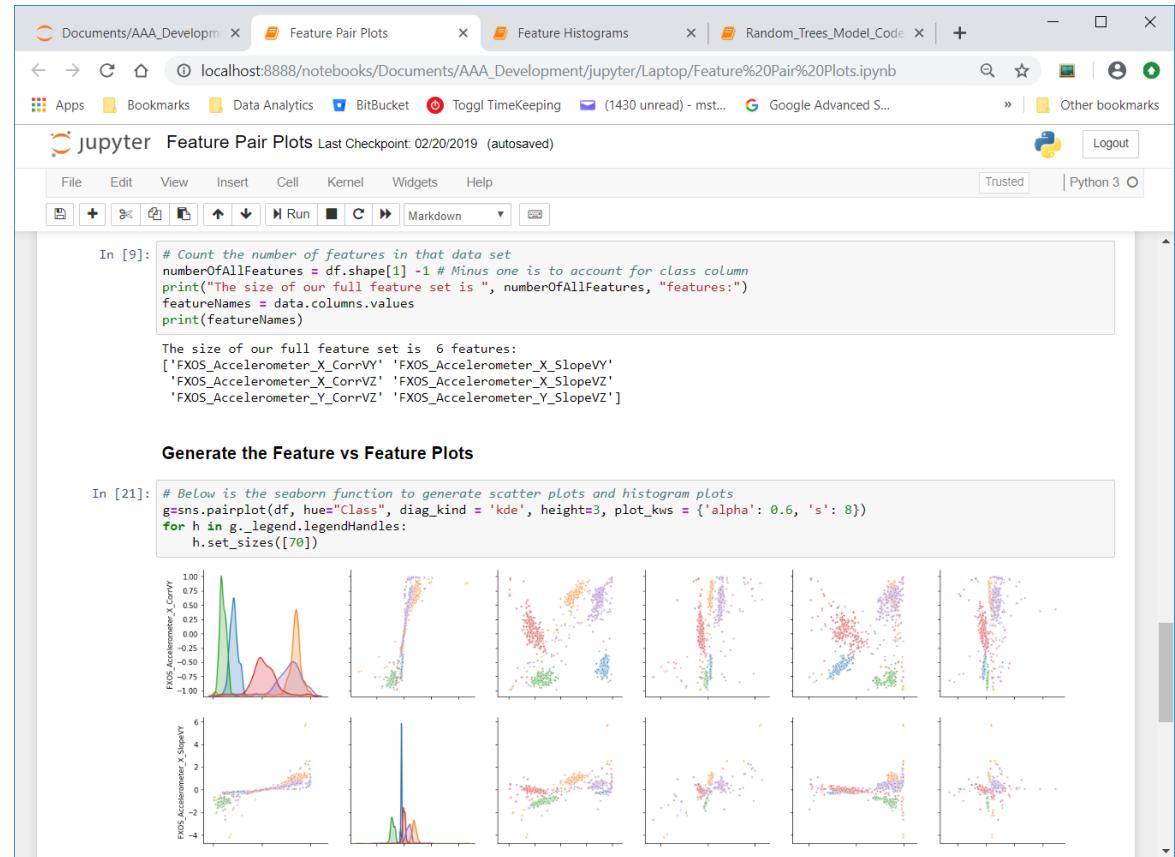
- Anaconda provides a solid framework for managing your Python, R and Jupyter environments
- Anaconda is free and open source
- <https://www.anaconda.com/products/individual>
- Click the “Download” button and follow installation instructions. Windows users will then find an “Anaconda3” entry in their start menu.
- BOTH scikit-Learn and Tensorflow/Keras can be run under this environment.

Anaconda

- Anaconda includes many Python packages in its default installation.
- Anaconda gives you the ability to setup multiple environments with different combinations (and versions) of libraries. You can switch between environments with a single command.
- The easiest way to tell if a 3rd party Python script or notebook will run is to try it. Python will give you clear messages indicating packages that need to be installed.
- In that case, a “`pip install <package_name>`” from the Python command line or a cell in a Jupyter notebook will probably do the trick. Best practice is simply to Google the package name to look up the recommended installation command.

Open Source Tools - Jupyter Notebooks

- Jupyter gives us the ability to execute R and Python scripts in a web browser environment
- Documentation, code and results are all part of a notebook
- Notebooks can easily be shared
- The Jupyter ecosystem is still being actively expanded to include additional capabilities, including an extensive set of graphical widgets.
- Python support is available for almost all non-graphical machine learning platforms



Recommendation:
<https://www.anaconda.com/distribution/>

Google Colab

- Execute your notebooks in the cloud.
- Google provides free access to GPUs and an easy way to share.
- Downside: No direct access to your embedded hardware

The screenshot shows a web browser window for 'Welcome To Colaboratory - Colab'. The URL in the address bar is <https://colab.research.google.com/notebooks/intro.ipynb>. The page content includes a 'Table of contents' sidebar with links to 'Getting started', 'Data science', 'Machine learning', 'More Resources', and 'Machine Learning Examples'. The main content area features a section titled 'What is Colaboratory?' which describes it as an environment for writing and executing Python code. It lists three benefits: 'Zero configuration required', 'Free access to GPUs', and 'Easy sharing'. Below this, a 'Getting started' section explains that the document is an interactive Colab notebook and provides an example of a code cell.

<https://colab.research.google.com/>

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!



Some Python Data Types

```
In [1]: my_data = [1, 2.0, 'a', True, [1,2], (1,2)]
```

```
print('my_data\t=', str(type(my_data)))
for v in my_data:
    print(v, '\t=', str(type(v)))
```

```
my_data = <class 'list'>
1      = <class 'int'>
2.0    = <class 'float'>
a      = <class 'str'>
True   = <class 'bool'>
[1, 2] = <class 'list'>
(1, 2) = <class 'tuple'>
```

```
In [2]: for i in range(len(my_data)):
```

```
    print(my_data[i], '\t=', str(type(my_data[i])))
```

```
1      = <class 'int'>
2.0    = <class 'float'>
a      = <class 'str'>
True   = <class 'bool'>
[1, 2] = <class 'list'>
(1, 2) = <class 'tuple'>
```

- Lists are denoted with square brackets []
- Lists can contain any other data type
- Unlike C and C++, indentation is used to denote scope, not curly braces.
White space IS significant.
- Python is designed to be easy to read.

```
In [4]: for i in range(4):
```

```
    print(i)
```

```
0  
1  
2  
3
```

Python Dictionaries

Python dictionaries are:

- Unordered
- Changeable
- Indexed
- Written with curly brackets
- Are composed of key/value pairs

```
In [26]: thisdict = {  
    'First': 'Tom',  
    'Last': 'Jones',  
    'DOB': 1964,  
    'MaritalStatus': 'single'  
}  
thisdict['NetWorth'] = 0.05  
  
print(thisdict)  
print(thisdict['First'])
```

{'First': 'Tom', 'Last': 'Jones', 'DOB': 1964, 'MaritalStatus': 'single', 'NetWorth': 0.05}
Tom

Python Functions

- Because Python is loosely typed, a single function declaration can often handle multiple variable types.
- Functions can return more than one value via the use of “tuples”

In [44]:

```
# add two variables
def myAdd(x, y):
    return(x+y)

print(myAdd(2,1))
print(myAdd(2,1.0))
print(myAdd('Hello ', 'There'))
print(myAdd([2,3,4], [8,9]))
```

```
3
3.0
Hello There
[2, 3, 4, 8, 9]
```

In [35]:

```
# Function for adding 2 wide tuples
def myAddTuple(x, y):
    return x[0]+y[0], x[1]+y[1]

x, y = myAddTuple((2,3), (4,4))
print(x, y)
```

```
6 7
```

Python is Object-Oriented

```
In [28]: class object():
```

```
    def __init__(self, name):  
        self.name = name;  
  
    def identity(self):  
        return(self.name)
```

```
obj = object('Friend')  
obj.identity()
```

```
out[28]: 'Friend'
```



You could derive the object class from another by specifying the parent class here

The class name is “object”

This is the constructor:

- ‘self’ is a pointer to the object
- ‘name’ is a parameter to the constructor
- Self.name is a local variable used to store the name of this object

‘identity’ is a just a function we defined.
It always has the ‘self’ pointer as an implied first parameter.

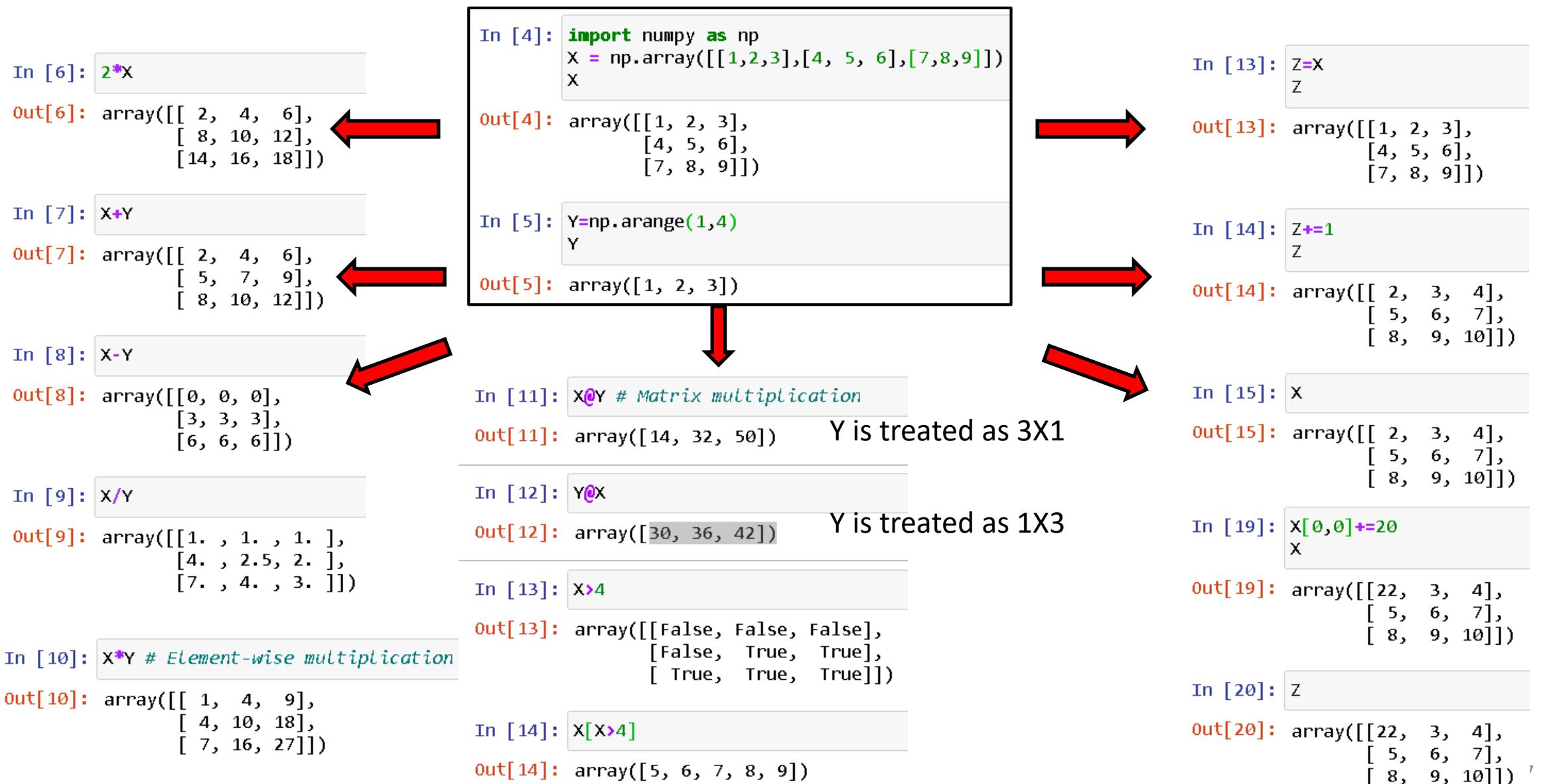
Python Libraries

- Much of the functionality in Python comes from the wealth of function libraries available.
- There is a two step process to use them:
 - Install the library in your local environment via pip or conda commands
 - Import the functions you need at the top of your program

1 Standard Module Imports

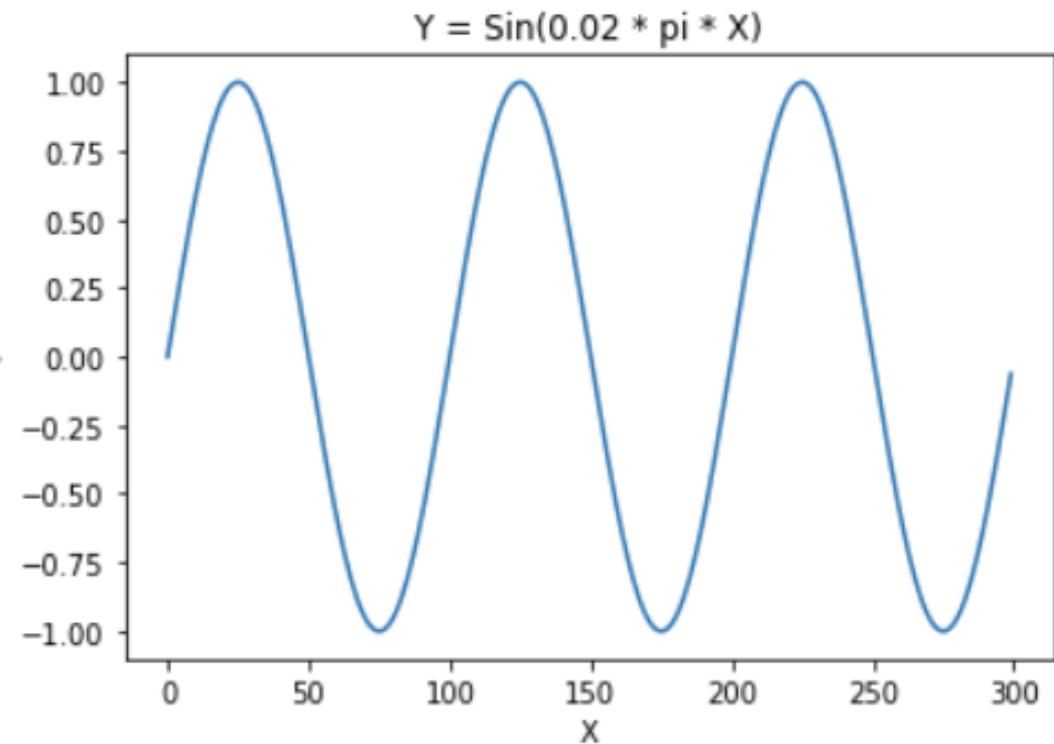
```
In [1]: from sklearn import svm, mixture, neighbors, tree, gaussian_process, ensemble, neural_network, naive_bayes, \
discriminant_analysis, metrics, preprocessing
import numpy as np
import pandas as pd
import bqplot as bq
import ipywidgets as widgets
from skimage import measure
import scipy.stats as stats
import math
import inspect
import matplotlib.pyplot as plt
from sklearn.gaussian_process.kernels import RBF
import traceback
```

Use the NumPy library for matrix math



Matplotlib is the most popular plotting library

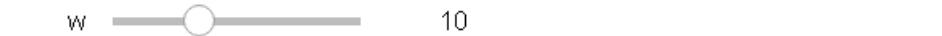
```
In [39]: import matplotlib.pyplot as plt
import numpy as np
import math
X = np.arange(0, 300)
Y = np.sin(0.02 * math.pi * X)
plt.plot(X, Y)
plt.title('Y = Sin(0.02 * pi * X)')
plt.xlabel('X')
plt.ylabel('Y')
```



'bqplot' is a great 2-D plotting library with hooks for developing interactive graphics.

Build Graphical Interfaces with the IP Widgets Library

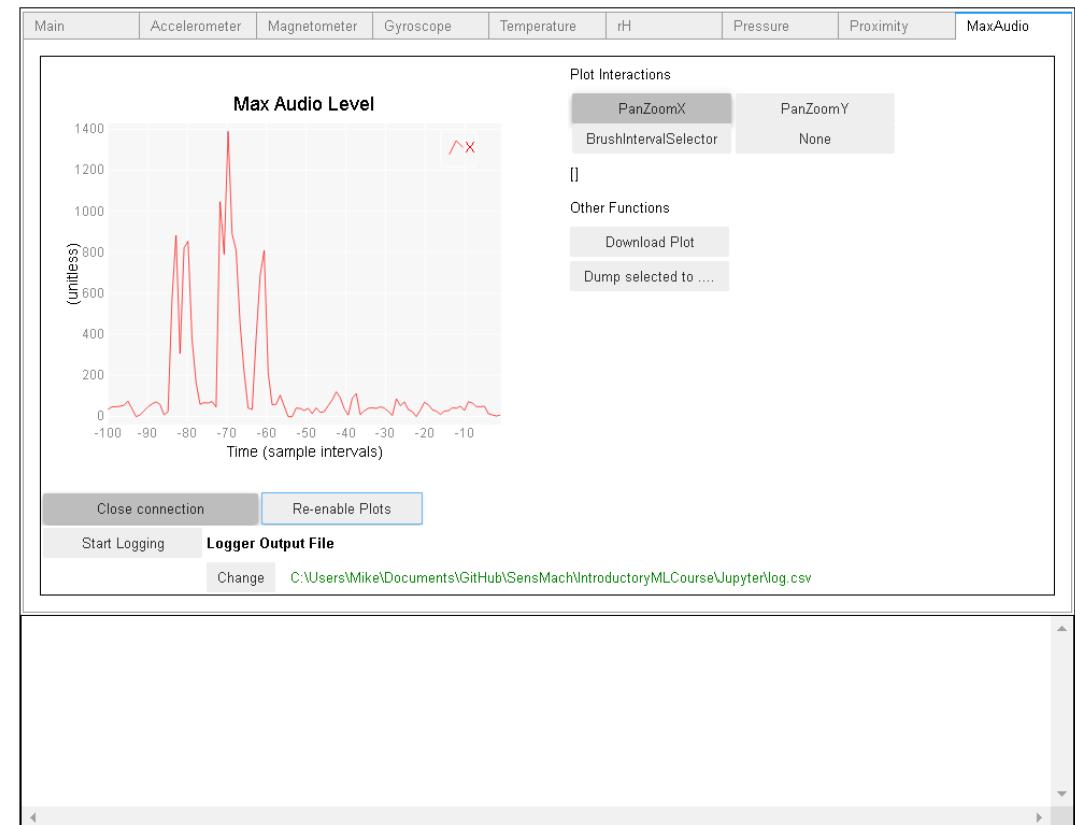
```
In [1]: import ipywidgets as widgets
from IPython.display import display
w = widgets.IntSlider(description='W', min=-10, max=50, step=1)
display(w)
```



```
In [2]: w.value
```

```
Out[2]: 10
```

```
In [4]: widgets.ToggleButton(
    value=False,
    description='Toggle Button',
    disabled=False,
    button_style='',
    tooltip='my button',
    icon='check'
)
```



Classical ML

- **scikit-learn** is easily the most accessible and well-documented toolkit available for classical machine learning.
- It can be used in a standard Jupyter environment.
- scikit-learn.org contains a huge amount of huge collection of tutorials and examples. The latter are easily downloaded and run on your laptop.

scikit learn

Install User Guide API Examples More ▾

scikit-learn

Machine Learning in Python

Getting Started Release Highlights for 0.23 GitHub

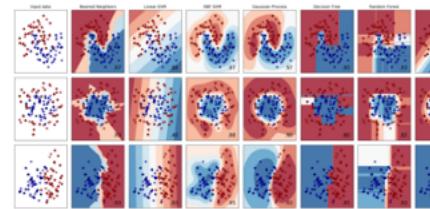
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...



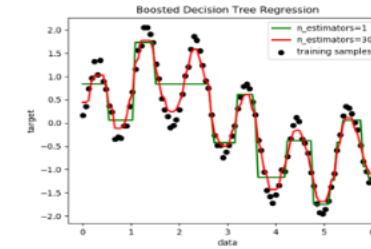
Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, nearest neighbors, random forest, and more...



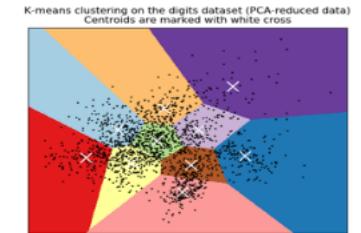
Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...



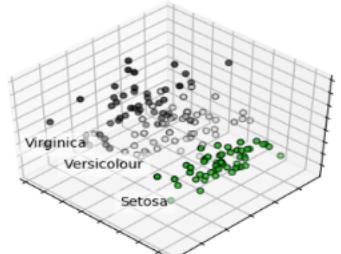
Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...



classification/plot_classifier_comparison.html

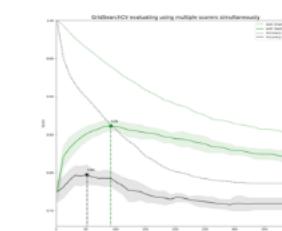
Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning

Algorithms: grid search, cross validation, metrics, and more...



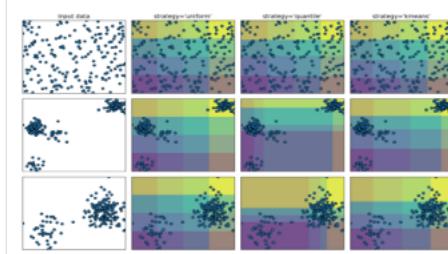
Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.

Algorithms: preprocessing, feature extraction, and more...



Examples

Summary/Wrapup

- We've learned how to code a simple C++ program and embedded it into an Arduino environment.
- We've looked at both Windows and Cloud-based environments
- We've seen how it's possible for Jupyter/Python code to interact with your hardware
- We've learned a bit about Python coding

Next time: Detailed design review of a fruit classification system