

Final Project
Jennifer Hajduk – jhajduk2
IoT 437 Fall 2022

Video Link: <https://youtu.be/1RTfxI2mh4E>

Repo Link: <https://gitlab.engr.illinois.edu/jhajduk21/iot-437-final-project/-/tree/main>

Table of Contents

Motivation	3
Implementation.....	3
Materials	6
Video and Links for working with the SDS011 sensor.....	6
Fritzing Topology	6
SDS011 Sensor Pin Configuration Details	7
Assembled Air Quality Monitor:	9
Securing the Raspberry Pi.....	9
Enabling Raspberry Pi Interfaces	10
Testing the Sensor and installing modules	12
Data Flow	15
Setting Up the Jupyter Notebook on the Raspberry Pi	15
Installing the libraries for Jupyter:	15
Forwarding the local port 8888 to Raspberry Pi 8888:	16
Starting the Jupyter Notebook on the Raspberry Pi	16
Jupyter Notebook from Local Web Browser	16
Coding the Sensor	17
Twilio	18
Continuous Run	19
Device Health	20
Jupyter	24
Results.....	25
Conclusion	28
Overall Project.....	28
Skill Building	28
Innovation	28
Scope	29

Motivation

My motivation for creating this project stems from experience wildfires after moving to the west coast of the US. The air was so thick and dangerous that we could not go outside for extended periods of time. This was the inspiration for getting a top-of-the-line Air Purifier. While we assume that the Air Mega air purifier is keeping our air quality safer, it is always important to experiment and measure. Inhaling dangerous particles, such as those that are measured with the SDS011 sensor, can lead to serious health problems and even premature death. Monitoring the air quality and ensuring that updates are sent throughout the day gives peace of mind that the air is not only safe for us, but our dog as well. Air is essential to life; we cannot choose to abstain from it when the air quality is not good for our health. A realistic solution is to improve the air quality in your home with more or better-quality purifiers. If you do not measure, however, you will not know how to best invest in the safety of the air around you, if at all. Given that we often open the windows and doors throughout the day the air quality can change drastically from hour to hour. While the Air Mega generally indicates that the air quality is in safe ranges, we want to be sure that we take measurements and verify the results with my own device.

This mentality extends beyond our immediate environment. Populations that include senior citizens, young children and pets need to monitor air quality as these populations are particularly susceptible to airborne pathogens. Another population that requires air quality monitoring include those with respiratory diseases and weak immune systems. When we think of all the people that can be affected by air quality, it is easy to see that our friends, family, and even ourselves can be deeply affected by knowing about and taking countermeasures against bad air quality. Therefore, monitoring air quality is so important.

The SDS011 sensor can pick up air particles sizes that are 2.5 microns and 10 microns in size. Particles that are 2.5 microns in size can be particularly harmful as they are easily inhaled into the lungs. Examples of air quality particulates that are 2.5 microns in size include combustion of gasoline, oil, diesel fuel or wood. Examples of particulates that are 10 microns in size include dust from construction sites, landfills and agriculture, and wildfires. It is important to note that these particulates can occur naturally in the atmosphere as well as originating from human activities. Since we live in an urban environment, these gases are ever present to a degree. It is important, particularly in Oregon where I live to ensure that we have good air quality due to the varying number of wildfires that occur during the year. I am hoping that the Air Mega proves to be a great investment in our home by accurately monitoring the air quality. I will use my device to verify that the readings are within good parameters.

Implementation

When designing the code for the air quality monitor, I wanted to keep in mind that the purpose is to compare the air quality with and without the Air Mega air purifier on. The first thing to consider was to leave the AirMega off for a few days and keep the normal use of doors and windows the same to get an accurate environment with all other factors remaining the same. We also continued to cook, clean, and do other normal activities that can influence the air

quality in our home. The air quality monitor that I built should be running continuously during this time to get a good sample size of the air. After running the sensor for two days continuously, I turned the Air Mega on to continuous mode as well to see if the air quality has changed according to the SDS011 sensor.

I am particularly concerned with the 2.5 sensor reading as these particles can be more harmful when inhaled. Here is a chart of how to interpret the result from the PM2.5 particulate sensor readings so that I know what to look out for:

PM _{2.5}	Air Quality Index	PM _{2.5} Health Effects	Precautionary Actions
0 to 12.0	Good 0 to 50	Little to no risk.	None.
12.1 to 35.4	Moderate 51 to 100	Unusually sensitive individuals may experience respiratory symptoms.	Unusually sensitive people should consider reducing prolonged or heavy exertion.
35.5 to 55.4	Unhealthy for Sensitive Groups 101 to 150	Increasing likelihood of respiratory symptoms in sensitive individuals, aggravation of heart or lung disease and premature mortality in persons with cardiopulmonary disease and the elderly.	People with respiratory or heart disease, the elderly and children should limit prolonged exertion.
55.5 to 150.4	Unhealthy 151 to 200	Increased aggravation of heart or lung disease and premature mortality in persons with cardiopulmonary disease and the elderly; increased respiratory effects in general population.	People with respiratory or heart disease, the elderly and children should avoid prolonged exertion; everyone else should limit prolonged exertion.
150.5 to 250.4	Very Unhealthy 201 to 300	Significant aggravation of heart or lung disease and premature mortality in persons with cardiopulmonary disease and the elderly; significant increase in respiratory effects in general population.	People with respiratory or heart disease, the elderly and children should avoid any outdoor activity; everyone else should avoid prolonged exertion.
250.5 to 500.4	Hazardous 301 to 500	Serious aggravation of heart or lung disease and premature mortality in persons with cardiopulmonary disease and the elderly; serious risk of respiratory effects in general population.	Everyone should avoid any outdoor exertion; people with respiratory or heart disease, the elderly and children should remain indoors.

I used a variety of materials and sources to build the air quality monitor and have them listed below. One of the most important things that I considered was to have a sturdy case for the air quality monitor because it was running continuously, and I did not want any elements such as moisture or dust to interfere with its operation. Thinking about how such a device might operate in an open-air environment (in the “field”) I might use some deterrent spray for small animals that like to chew through any wiring. When we first got our dog, we used bitter apple spray to prevent chewing on wires in our home so something similar might be used to prevent wildlife from chewing the device wires.

Materials

- Raspberry Pi Model 4 B Rev 1.5 – (Amazon)
- DEVMO PM Sensor SDS011 High Precision PM2.5 Air Quality Detection Sensor
- Visual Studio Code with Python Extension = (Amazon)
- Flirc Raspberry Pi 4 Case (Silver) – (Amazon)
- STEADYGAMER - 64GB Raspberry Pi Preloaded (RASPBIAN/Raspberry PI OS) SD Card | 4, 3B+ (Plus), 3A+, 3B, 2, Zero Compatible with All Pi Models (64GB) – (Amazon)
- Optional: spray deterrent to prevent interest from animals

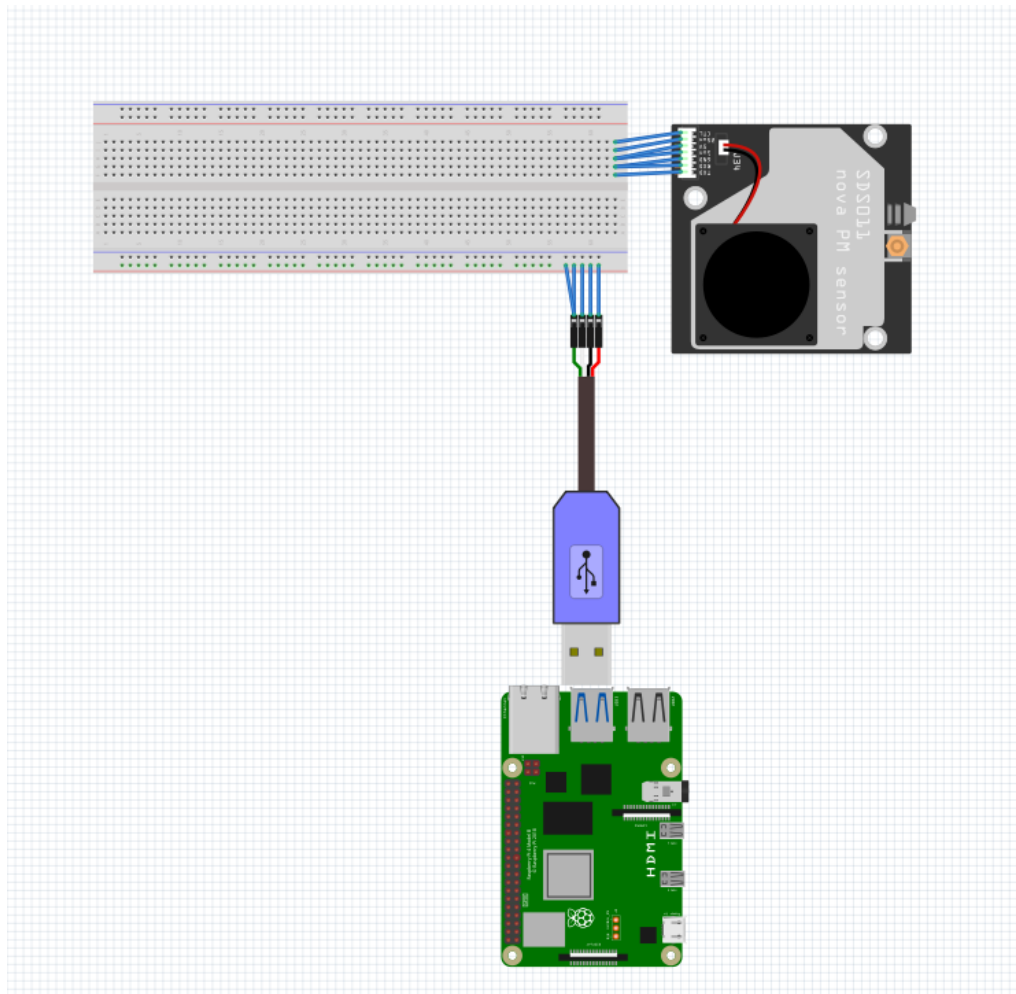
Video and Links for working with the SDS011 sensor

- Using sensors with the Raspberry Pi: <https://www.youtube.com/watch?v=gnE4v-PcYKQ>
- Pyserial: [Python Serial Communication \(Pyserial\) Tutorial 2022 - CodingCompiler](#)
- Reading serial data with Python: <https://www.pythonpool.com/python-serial-read/>
- How to set up Jupyter on Raspberry Pi: <https://www.instructables.com/Jupyter-Notebook-on-Raspberry-Pi/>
- Port forwarding for Jupyter installation on Raspberry Pi: [https://stackoverflow.com/questions/59069494/can-not-access-jupyter-server-on-raspberry-pi - :~:text=To access the notebook server on your Pi,,ssh like this%3A ssh -L 8888%3Alocalhost%3A8888 your.pi.address](https://stackoverflow.com/questions/59069494/can-not-access-jupyter-server-on-raspberry-pi-localhost:8888-ssh-like-this-ssh-L8888-your.pi.address)

Fritzing Topology

I chose to use the SDS011 High precision PM2.5/PM10 air quality sensor for its precision with detecting air quality particles with the range of .3 and 10 microns. The product uses a USB to

TTL connector to communicate with the Raspberry Pi 4 Model B (found the closest match that I could for the serial connector in Fritzing).



[SDS011 Sensor Pin Configuration Details](#)

I took some time to study the pin configuration of the SDS011 sensor and learn what each component does. Here is a graph for it:

No.	Pin Name	Function
1	NC	No connection/Not used
2	1 μ m	Range of PM2.5 value (0-999 μ g /m ³) and provide output in PWM form on this pin
3	5V	Connect 5V to this pin to power dust sensor
4	2.5 μ m	Range of PM10 value (0-999 μ g /m ³) and provide output in PWM form on this pin
5	GND	Connect to ground terminal of power supply and common reference with Arduino or microcontroller
6	Rx	Receiver terminal of UART module and used to receive commands from computer or microcontrollers
7	Tx	Transmitter pin of UART and used to transmit output to the target device such as Arduino or other microcontrollers

Assembled Air Quality Monitor:



Securing the Raspberry Pi

To protect my device “in the field” (my dining room table) I enabled the Uncomplicated Firewall by doing the following:

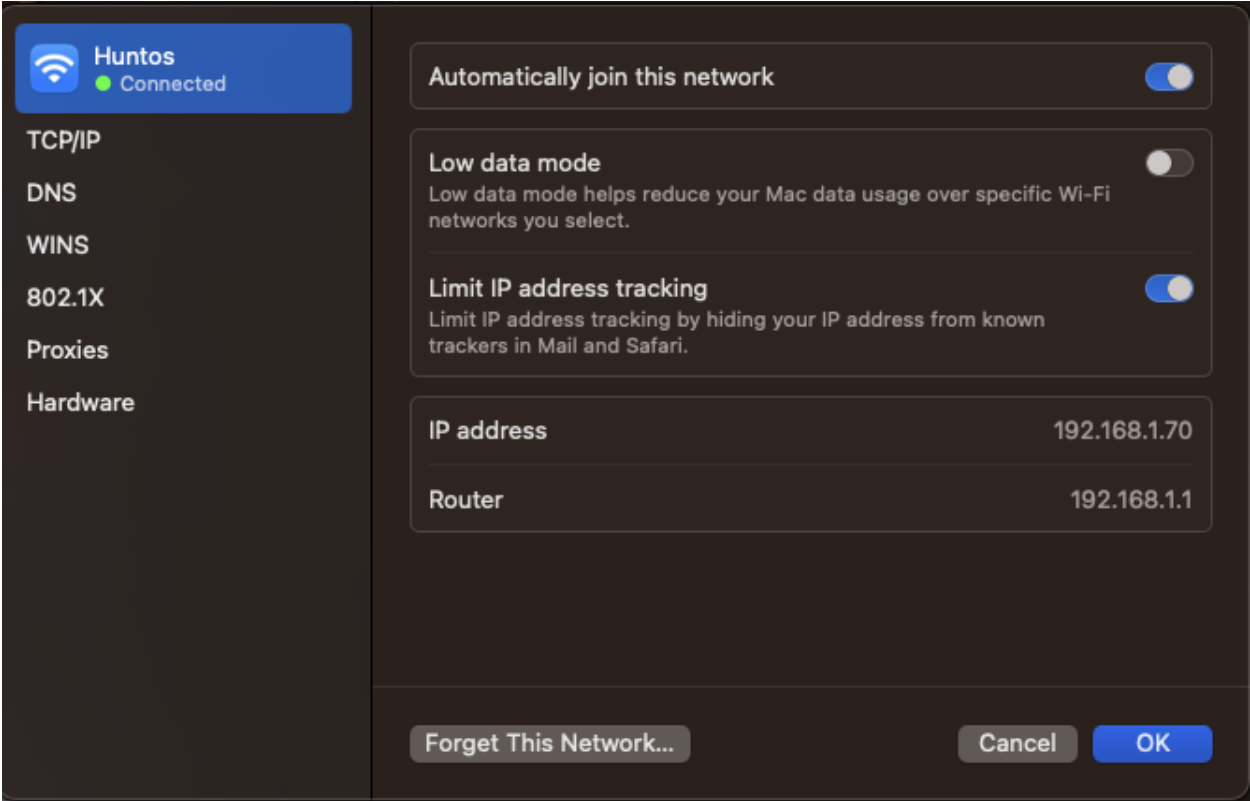
```
● pi@raspberrypi:~ $ sudo ufw status verbose
Status: inactive
● pi@raspberrypi:~ $ sudo ufw allow 22
Rules updated
Rules updated (v6)
● pi@raspberrypi:~ $ sudo ufw allow from 192.168.1.70 to any port 22
Rules updated
● pi@raspberrypi:~ $ sudo ufw enable
Firewall is active and enabled on system startup
○ pi@raspberrypi:~ $
```

```
● pi@raspberrypi:~ $ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To Action From
--
22 ALLOW IN 192.168.1.70

○ pi@raspberrypi:~ $
```

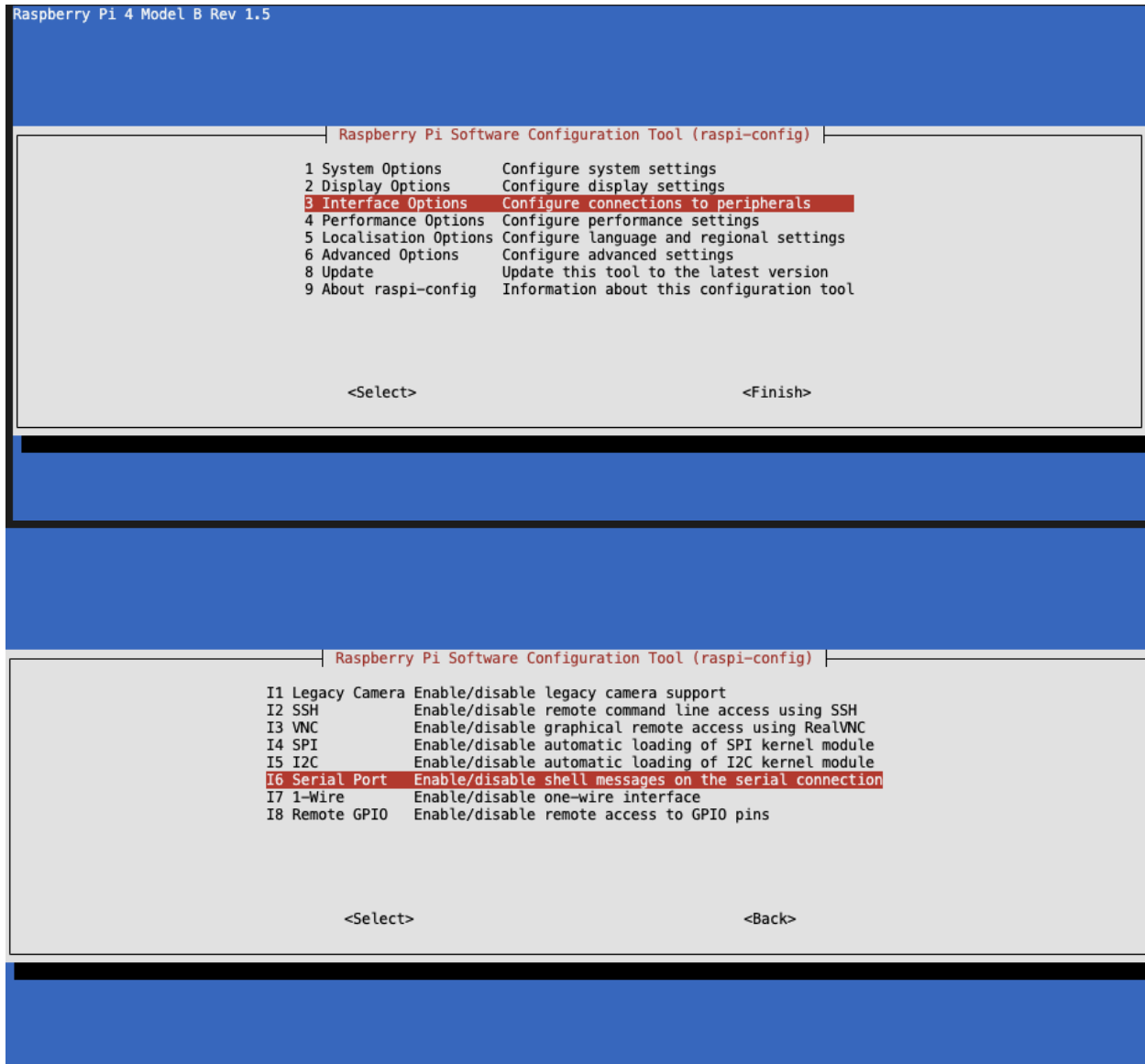
I now allow only my IP Address through on port 22 to the Raspberry Pi. I also set up a static private IP address using my specific home router instructions. I use the Ubiquiti Dream Machine industrial quality switch at home. I generally get the same IP address in any case, but it will relieve possible future headache to ensure that this is set up correctly so that I will not be blocked from my device by the firewall. Here is the IP address from my local machine to show that it is indeed getting the correct IP Address:

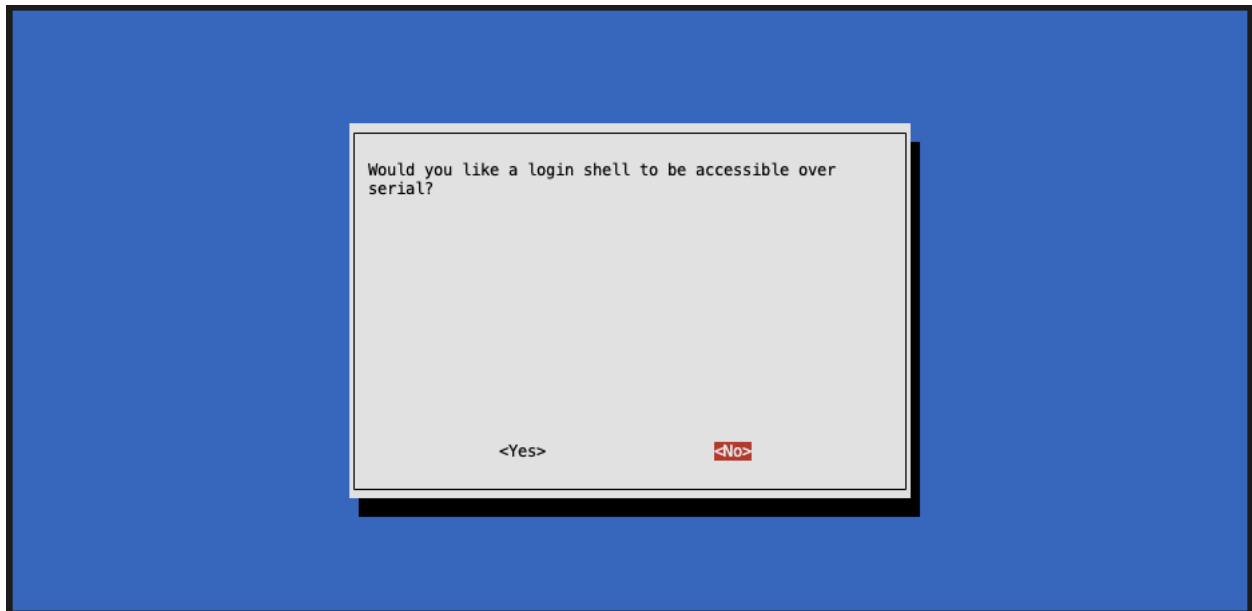


Enabling Raspberry Pi Interfaces

I quickly learned after trying to code the SDS011 sensor unsuccessfully for some time, that the serial port on the Pi had to be enabled. I had to do some research online on how to do this. To allow the air quality sensor to interact with the Raspberry Pi on the serial port, I ensured that

the serial interface was active by going through the raspi-config menu and allowing the communication:





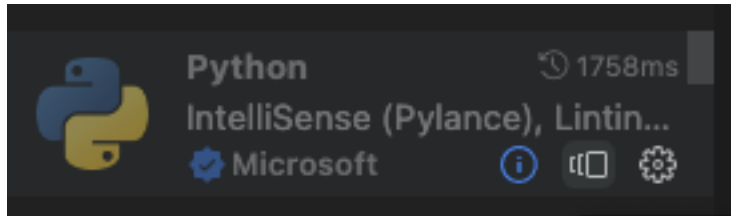
I then rebooted the pi and was able to have my code successfully interact with the SDS011 sensor.

[Testing the Sensor and installing modules](#)

Once I learned to do this, I experimented with the sensor code with some test code to ensure that it was in working order. Testing is of course one of the quintessential steps to building a device as all components need to work together successfully. I installed the **pyserial** library:

```
sudo pip3 install -U pyserial
```

Since I am working in VSCode with the remote extension to interact with the Raspberry Pi, I also had to install the Python extension on the Raspberry Pi while connected to the remote connection:



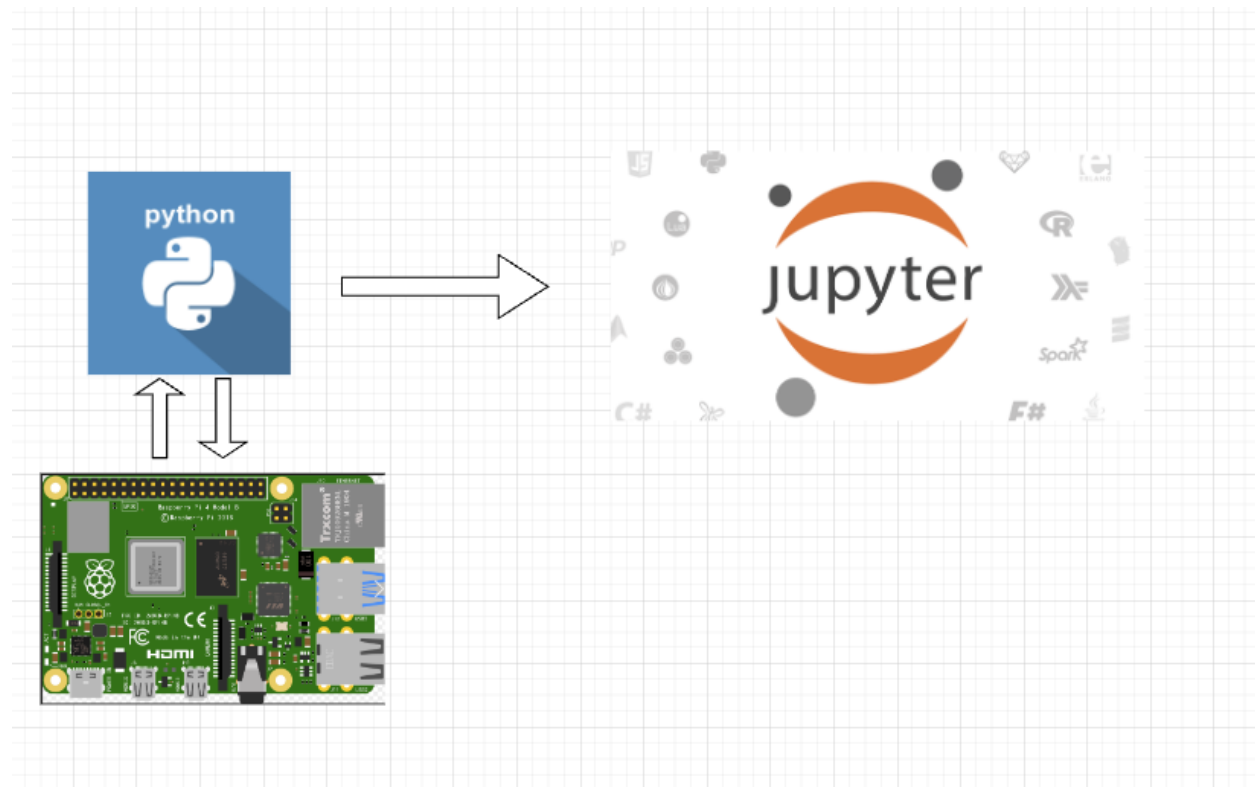
I decided to perform a test using code from the sources above to check if the sensor was working correctly (can be found in the repository: **sensor_test.py**):

```
1 from serial import *
2 import time
3
4 ser = Serial(port='/dev/ttyUSB0', baudrate=9600, parity=PARITY_NONE, stopbits=STOPBITS_ONE, bytesize=EIGHTBITS, timeout=1)
5
6 while True:
7     data = []
8     for index in range(0,10):
9         datum = ser.read()
10        data.append(datum)
11
12    pmtwofive = int.from_bytes(b''.join(data[2:4]), byteorder='little')/10
13    pmten = int.from_bytes(b''.join(data[4:6]), byteorder='little')/10
14
15    print(pmtwofive)
16    print(pmten)
17    time.sleep(10)
```

The output of the code from the 2.5 micrometer and 10 micrometer sensors was:

1.7
4.3
1.7
4.2
1.7
4.3
1.7
4.3
1.7
4.4
1.4
4.2
1.5
4.2
1.5
4.2
1.5
4.2
1.5
4.1
1.5
4.4
1.7
4.4

Data Flow



Setting Up the Jupyter Notebook on the Raspberry Pi

I wanted to be able to visualize the data locally, so I used this tutorial to set up Jupyter Notebook on my Raspberry Pi (found in the links section above). I used this information to forward local port 8888 to my pi's 8888 port.

Installing the libraries for Jupyter:

```
lab-widgets, jupyterlab-pygments, entrypoints, defusedxml, debugpy, bleach, attrs, asttokens, terminado, stack-data, qtpy, matplotlib-inline, jupyter-core, jsonschema, Jinja2, importlib-metadata, cffi, anyio, nbformat, jupyter-client, ipython, argon2-cffi-bindings, nbclient, ipykernel, argon2-cffi, qtconsole, nbconvert, jupyter-console, ipywidgets, jupyter-server, notebook-shim, nbclassic, notebook, jupyter

Attempting uninstall: Send2Trash
Found existing installation: Send2Trash 1.6.0b1
Uninstalling Send2Trash-1.6.0b1:
Successfully uninstalled Send2Trash-1.6.0b1
Attempting uninstall: python-dateutil
Found existing installation: python-dateutil 2.8.1
Uninstalling python-dateutil-2.8.1:
Successfully uninstalled python-dateutil-2.8.1
Attempting uninstall: MarkupSafe
Found existing installation: MarkupSafe 1.1.1
Uninstalling MarkupSafe-1.1.1:
Successfully uninstalled MarkupSafe-1.1.1
Attempting uninstall: asttokens
Found existing installation: asttokens 2.0.4
Uninstalling asttokens-2.0.4:
Successfully uninstalled asttokens-2.0.4
Attempting uninstall: Jinja2
Found existing installation: Jinja2 2.11.3
Uninstalling Jinja2-2.11.3:
Successfully uninstalled Jinja2-2.11.3
Successfully installed Send2Trash-1.8.0 anyio-3.6.2 argon2-cffi-21.3.0 argon2-cffi-bindings-21.2.0 asttokens-2.1.0 attrs-22.1.0 backcall-0.2.0 bleach-5.0.1 cffi-1.15.1 debugpy-1.6.3 defusedxml-0.7.1 entrypoints-0.4 executing-1.2.0 fastjsonschema-2.16.2 importlib-metadata-5.0.0 ipykernel-6.17.1 ipython-8.6.0 ipython-genutils-0.2.0 ipywidgets-8.0.2 Jinja2-3.1.2 jsonschema-4.17.0 jupyter-1.0.0 jupyter-client-7.4.5 jupyter-console-6.4.4 jupyter-core-5.0.0 jupyter-server-1.23.2 jupyterlab-pygments-0.2.2 jupyterlab-widgets-3.0.3 MarkupSafe-2.1.1 matplotlib-inline-0.1.6 mistune-2.0.4 nbclassic-0.4.8 nbclient-0.7.0 nbconvert-7.2.5 nbformat-5.7.0 nest-asyncio-1.5.6 notebook-6.5.2 notebook-shim-0.2.2 packaging-21.3 pandocfilters-1.5.0 pickleshare-0.7.5 platformdirs-2.5.4 prometheus-client-0.15.0 prompt-toolkit-3.0.32 ptyprocess-0.7.0 pure-eval-0.2.2 pycparser-2.21 pyparsing-2.2.1 pyrsistent-0.19.2 python-dateutil-2.8.2 pyzmq-24.0.1 qtconsole-5.4.0 qtpy-2.3.0 sniffio-1.3.0 stack-data-0.6.1 terminado-0.17.0 tinycss2-1.2.1 tornado-6.2 traitlets-5.5.0 wcwidth-0.2.5 websocket-client-1.4.2 widgetsnbextension-4.0.3 zipp-3.10.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/en/latest/faq/#using-pip-as-root
pi@raspberrypi:~/cs437net Final Project $
```

Forwarding the local port 8888 to Raspberry Pi 8888:

```
jenniferhajduk@Jennifers-Air ~ % ssh -L 8888:localhost:8888 192.168.1.53
pi@192.168.1.53's password:
Linux raspberrypi 5.15.61-v7l+ #1579 SMP Fri Aug 26 11:13:03 BST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

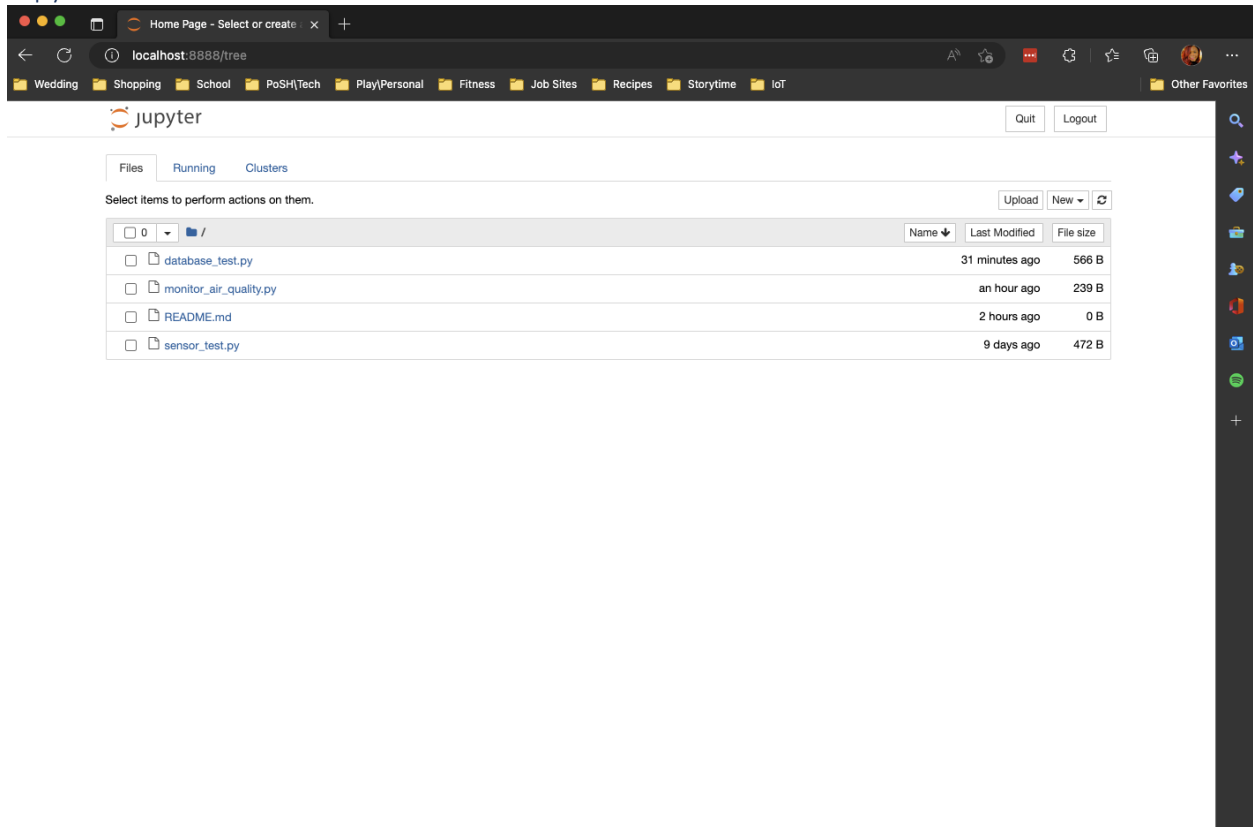
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Nov 14 22:26:12 2022
```

Starting the Jupyter Notebook on the Raspberry Pi

```
/pip.pypa.io/warnings/venv
pi@raspberrypi:~/cs437iot_Final_Project $ jupyter-notebook
[I 22:38:45.877 NotebookApp] Writing notebook server cookie secret to /home/pi/.local/share/jupyter/runtime/notebook_cookie_secret
[I 22:38:46.596 NotebookApp] Serving notebooks from local directory: /home/pi/.local/share/jupyter/runtime/notebook_cookie_secret
[I 22:38:46.596 NotebookApp] Jupyter Notebook 6.5.2 is running at:
[I 22:38:46.597 NotebookApp] https://localhost:8888/?token=cd97047c9be2ebdfb010b459e43429d9fbc96fbdecdfc572
[I 22:38:46.597 NotebookApp] or http://127.0.0.1:8888/?token=cd97047c9be2ebdfb010b459e43429d9fbc96fbdecdfc572
[I 22:38:46.597 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 22:38:46.610 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/pi/.local/share/jupyter/runtime/nbserver-2200-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=cd97047c9be2ebdfb010b459e43429d9fbc96fbdecdfc572
or http://127.0.0.1:8888/?token=cd97047c9be2ebdfb010b459e43429d9fbc96fbdecdfc572
[I 22:39:11.176 NotebookApp] 302 GET /?token=cd97047c9be2ebdfb010b459e43429d9fbc96fbdecdfc572 (:::1) 4.530000ms
```

Jupyter Notebook from Local Web Browser



The screenshot shows a web browser window with the address bar set to `localhost:8888/tree`. The browser's tab bar shows "Home Page - Select or create". The Jupyter Notebook interface is displayed, featuring a "Quit" and "Logout" button in the top right. Below the Jupyter logo, there are tabs for "Files", "Running", and "Clusters". The "Files" tab is active, showing a file browser interface. At the top of the file browser, it says "Select items to perform actions on them." and includes "Upload" and "New" buttons. A table lists the files in the current directory:

	Name	Last Modified	File size
<input type="checkbox"/>	/		
<input type="checkbox"/>	database_test.py	31 minutes ago	566 B
<input type="checkbox"/>	monitor_air_quality.py	an hour ago	239 B
<input type="checkbox"/>	README.md	2 hours ago	0 B
<input type="checkbox"/>	sensor_test.py	9 days ago	472 B

Coding the Sensor

In the below code, after the imports of the libraries, we see the line that initiates the serial connection to the sensor with the port information, the baudrate to send at 9600 bits per second, the parity of NONE to select no parity bit to be sent, stop bits set to one to check for baudrate or byte length mismatch, byte size set to eight bits to set the total number of bits to be read and a timeout of 1 to set the number of commands to accept before timing out.

I then fill out the sensitive data of my account sid and auth token from my Twilio account (not pictured) to be able to authenticate to my Twilio account and send text messages. I also input my purchased Twilio number and my cell number into the `sendTextMessage` function (not pictured)

The **takeMeasurement** function that you see will take measurements and format the results to be used by the other functions. I retrieve the current time and use the boilerplate test code to take a reading from the SDS011 sensor. I capture the readings in variables, and I then create the message that will be sent via text that includes a more verbose and human friendly output of the sensor results. Thinking of how to store the data in different ways can be helpful depending on how and who will access the data. I create three variables (`pmtwofive` and `pmten` and `today`) that will be sent as-is to a csv file. I will be using this format of the data in my Jupyter notebook.

The **saveResults** function that will save the results of the readings and the date in a csv file every time a reading is performed.

The **sendTextMessage** function takes the text message variable and uses it to send to my phone for each reading done

In the main body of the function, I call the previous functions and insert a sleep timer so that measurements will be taken every hour.

```
from serial import *
from sds011 import SDS011
import time
from datetime import datetime, date
import csv
from twilio.rest import Client
```

```

ser = Serial(port='/dev/ttyUSB0', baudrate=9600, parity=PARITY_NONE, stopbits=STOPBITS_ONE, bytesize=EIGHTBITS, timeout=1)
account_sid = ''
auth_token = ''
client = Client(account_sid, auth_token)

#Take Measurement
def takeMeasurement():

    today = date.today()
    current_time = datetime.now().strftime("%H:%M")

    data = []
    for index in range(0,10):
        datum = ser.read()
        data.append(datum)

    pmtwofive = int.from_bytes(b''.join(data[2:4]), byteorder='little')/10
    pmten = int.from_bytes(b''.join(data[4:6]), byteorder='little')/10

    text_message = "PM10 reading is: " + str(pmten) + " PM2.5 reading is: " + str(pmtwofive) + " This measurement was taken at: " + str(current_time)
    return(pmtwofive, pmten, text_message, today)

def saveResults(twofive, ten, day):

    with open('./results.csv', 'a', newline='\n') as r:
        writer = csv.writer(r)
        writer.writerow([twofive, ten, day])
        return

def sendTextMessage(text):
    message = client.messages \
        .create(
            body=text,
            from_ = ,
            to =
        )
    return([message.sid])

def main():

    while True:
        twopointfive, ten, text, day = takeMeasurement()
        saveResults(twopointfive, ten, day)
        sendTextMessage(text)
        time.sleep(60 * 60)

if __name__ == "__main__":
    main()

```

Twilio

The Air Mega air purifier changes colors to indicate the quality of the air. This is nice but I wanted to have detailed data about the quality of the air to study later. There are ranges, as indicated by the chart above, of good, bad, and hazardous air quality. I wanted to know the exact parameters of the air and the particulate matter so that I could be sure that the air around us was healthy. A convenient way to do this is to have text message sent to me periodically. To have the script send me text messages, I employed the Twilio platform. Twilio allows developers to use pre-built libraries to send communications from their code, including text messages. It is a low-cost option for developers to test their code while not breaking the bank. I signed up for a Twilio account and purchase a phone number through them. I then used their pre-built library functions to send the messages in the code:

<https://www.twilio.com/docs/sms/quickstart/python>

Continuous Run

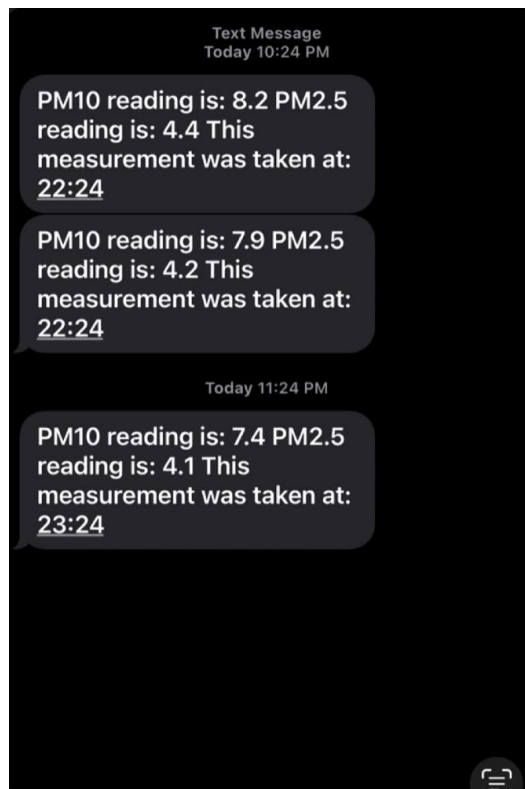
I wanted to ensure that the program ran continuously, and I found that there were several ways to do this by modifying different system files. While I did try other methods such as editing the cron file on the Pi, none of those options seemed to be running the code. I found the tmux package from this site:

<https://www.tomshardware.com/how-to/run-long-running-scripts-raspberry-pi>

```
0 upgraded, 2 newly installed, 0 to remove and 55 not upgraded.
Need to get 304 kB of archives.
After this operation, 707 kB of additional disk space will be used.
Get:1 http://raspbian.mirror.axinja.net/raspbian bullseye/main armhf libutempter0 armhf 1.2.1-2 [8,596 B]
Get:2 http://raspbian.raspberrypi.org/raspbian bullseye/main armhf tmux armhf 3.1c-1+deb11u1 [296 kB]
Fetched 304 kB in 2s (164 kB/s)
Selecting previously unselected package libutempter0:armhf.
(Reading database ... 108970 files and directories currently installed.)
Preparing to unpack .../libutempter0_1.2.1-2_armhf.deb ...
Unpacking libutempter0:armhf (1.2.1-2) ...
Selecting previously unselected package tmux.
Preparing to unpack .../tmux_3.1c-1+deb11u1_armhf.deb ...
Unpacking tmux (3.1c-1+deb11u1) ...
Setting up libutempter0:armhf (1.2.1-2) ...
Setting up tmux (3.1c-1+deb11u1) ...
Processing triggers for man-db (2.9.4-2) ...
Processing triggers for libc-bin (2.31-13+rpt2+rpil+deb11u5) ...
pi@raspberrypi:~/cs437iot_Final_Project $ tmux new -s test_session_air_quality
[detached (from session test_session_air_quality)]
pi@raspberrypi:~/cs437iot_Final_Project $ tmux list-sessions
test_session_air_quality: 1 windows (created Wed Nov 16 02:00:26 2022)
pi@raspberrypi:~/cs437iot_Final_Project $
```

What this package allows you to do is to run a session that will run the script continuously and allow you observability into the session information. Once I ran the python script with a tmux session, I received my first text message and knew that the script was running continuously. I was then able to log off from the Pi and continue to receive messages periodically

I awoke to the text messages from the Pi informing me of the air quality while I slept:



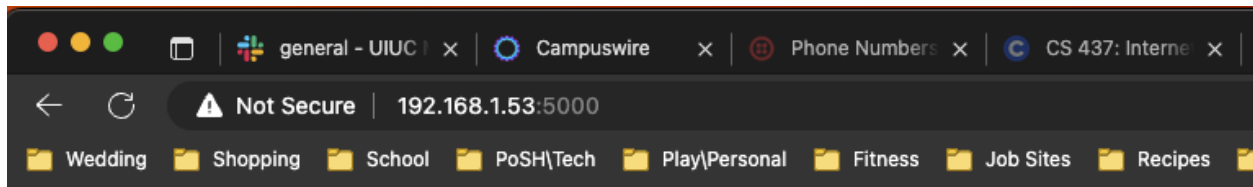
Device Health

Lab 2 was a struggle for me using web development for the first time. This struggle inspired me to try to do a simple web app to check the device information from the Pi. While this is great to be able to get information from the Pi sent to my phone, I also want to check the health of the Pi whenever I choose. I included a web page that I can access from my home network that will give me insights to how the air quality monitor is doing. I found a nice tutorial on getting flask setup on the Raspberry Pi here: <https://towardsdatascience.com/python-webserver-with-flask-and-raspberry-pi-398423cc6f5d>

When I set up the Flask server test, I used the sample code to get familiar with it:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
    return 'Hello world'
if __name__ == '__main__':
    app.run(debug=True, port=5000, host='0.0.0.0')
```

I used a port above 1024 to access and had to allow that port through the firewall with `sudo ufw allow 5000` on the Pi. When I accessed the web page, here is the initial result before customization:



Hello world

I proceeded to design the web page and write the application code to collect information from the Raspberry Pi and ensure that the information is accessible over a web page. I created the `app.py` program as shown below:

```

from flask import Flask
import datetime
import psutil
import platform
import os
from flask import Flask, render_template

app = Flask(__name__)
@app.route('/')
def index():
    svmem = psutil.virtual_memory()
    load1, load5, load15 = psutil.getloadavg()
    cpu_temp = os.popen("vcgencmd measure_temp").readline()
    cpu_temp = cpu_temp.replace("temp=", "")
    sys_data = {}
    templateData = {}
    sys_data['Time'] = datetime.datetime.now().strftime("%d-%b-%Y , %I : %M : %S %p")
    sys_data['System'] = platform.system()
    sys_data['Release'] = platform.release()
    sys_data['version'] = platform.version()
    sys_data['machine'] = platform.machine()
    sys_data['processor'] = platform.processor()
    sys_data['Total Cores'] = psutil.cpu_count(logical=True)
    sys_data['CPU Usage'] = (load15/os.cpu_count()) * 100
    sys_data['Total Memory'] = svmem.total
    sys_data['RAM Used GB'] = psutil.virtual_memory()[3]
    sys_data['Used Memory'] = svmem.used
    sys_data['CPU Temp'] = cpu_temp

    templateData = {
        'time' : sys_data['Time'],
        'system' : sys_data['System'],
        'release' : sys_data['Release'],
        'version' : sys_data['version'],
        'machine' : sys_data['machine'],
        'processor' : sys_data['processor'],
        'total_cores' : sys_data['Total Cores'],
        'cpu_usage' : sys_data['CPU Usage'],
        'cpu_temp' : sys_data['CPU Temp'],
        'ram_usage' : sys_data['RAM Used GB'],
        'total_memory' : sys_data['Total Memory'],
    }
    return render_template('index.html', **templateData)
if __name__ == '__main__':
    app.run(debug=True, port=5000, host='0.0.0.0')

```

Not wanting to re-invent the wheel, I am using **psutil** and **platform** Python libraries to easily retrieve data about the air quality monitor with Python. I input these details into a dictionary and then create the **templateData** object to access the data in the index.html

I created a simple index.html page to clearly display the information:

```

1  <!DOCTYPE html>
2  <head>
3  |   <title>Raspberry Pi Air Quality Monitor Device Details</title>
4  </head>
5  <body>
6  |   <h1>Current Time:</h1>
7  |   <h2>{{ time }}</h2>
8  |   <h1>System Information:</h1>
9  |   <h2>The operating system is: {{ system }}</h2>
10 |   <h2>The release of the OS is: {{ release }}</h2>
11 |   <h2>The version of the OS is: {{ version }}</h2>
12 |   <h2>The architecture is: {{ machine }}</h2>
13 |   <h1>CPU Information:</h1>
14 |   <h2>There temperature of the CPU is: {{ cpu_temp }}</h2>
15 |   <h2>There are {{ total_cores }} total cores</h2>
16 |   <h2>The current CPU usage is: {{ cpu_usage }}</h2>
17 |   <h1>Memory Information:</h1>
18 |   <h2>The total memory is: {{ total_memory }}</h2>
19 |   <h2>The used memory is: {{ ram_usage }}</h2>
20 </body>
21 </html>

```

When I run the web app, I am shown the following device information:

Current Time:

16-Nov-2022 , 10 : 58 : 23 PM

System Information:

The operating system is: Linux

The release of the OS is: 5.15.61-v7l+

The version of the OS is: #1579 SMP Fri Aug 26 11:13:03 BST 2022

The architecture is: armv7l

CPU Information:

There temperature of the CPU is: 50.1'C

There are 4 total cores

The current CPU usage % is: 3.8

Memory Information:

The total memory is: 4025401344

The used memory is: 251973632

In order to run the site (app.py) continuously, I again used a tmux session:

```
pi@raspberrypi:~/cs437iot_Final_Project/Device_Health_Server $ tmux list-sessions
monitor_airs: 1 windows (created Wed Nov 16 14:05:25 2022)
run_flask: 1 windows (created Wed Nov 16 14:06:52 2022)
pi@raspberrypi:~/cs437iot_Final_Project/Device_Health_Server $
```

I can now access the web page anytime that I choose to see device health information from any device on my home network. I can use the data to ensure that the Pi is not overloaded and is not overheating.

Jupyter

Now that I have my device set up and running, I want to start visualizing the data so that I can recognize any trends. I installed Jupyter Notebook on my Raspberry Pi using the tutorial linked above in the Links section.

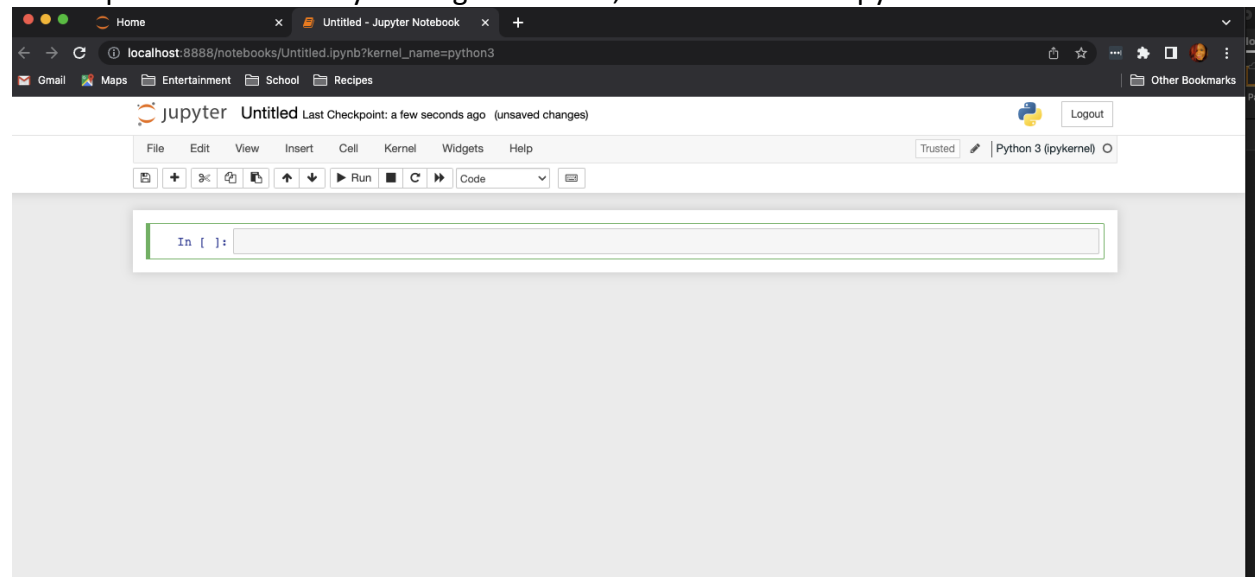
When starting the Jupyter Notebook, I grab the URL and port from the command output:

```
pi@raspberrypi:~/cs437iot_Final_Project/Jupyter $ jupyter-notebook
[I 20:56:07.870 NotebookApp] Serving notebooks from local directory: /home/pi/cs437iot_Final_Project/Jupyter
[I 20:56:07.870 NotebookApp] Jupyter Notebook 6.5.2 is running at:
[I 20:56:07.870 NotebookApp] http://localhost:8888/?token=a5e196312cf5c230f69e84b919385e817383e0ee37d9b013
[I 20:56:07.870 NotebookApp] or http://127.0.0.1:8888/?token=a5e196312cf5c230f69e84b919385e817383e0ee37d9b013
[I 20:56:07.870 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).

[C 20:56:07.885 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/pi/.local/share/jupyter/runtime/nbserver-13303-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=a5e196312cf5c230f69e84b919385e817383e0ee37d9b013
or http://127.0.0.1:8888/?token=a5e196312cf5c230f69e84b919385e817383e0ee37d9b013
```

Again, I must remember to allow port 8888 through the firewall on the Raspberry Pi using the Uncomplicated Firewall. By clicking on the link, I am taken to a Jupyter session:



When the page opens to show the notebook, I want to import the necessary libraries that I will use as well as the dataset:


```
In [1]: from matplotlib import pyplot as plt
import pandas as pd
df = pd.read_csv(r'../results.csv')
print(df)
```

The csv file is in the directory in which I am storing the csv results on the Pi. Here is the successfully imported data:

	twofive	ten	day
0	4.4	8.2	2022-11-16
1	4.2	7.9	2022-11-16
2	4.1	7.4	2022-11-16
3	4.2	7.4	2022-11-17
4	4.2	7.4	2022-11-17
..
69	4.8	7.1	2022-11-19
70	4.8	7.1	2022-11-19
71	4.6	7.1	2022-11-19
72	4.7	7.2	2022-11-19
73	4.7	7.3	2022-11-19

[74 rows x 3 columns]

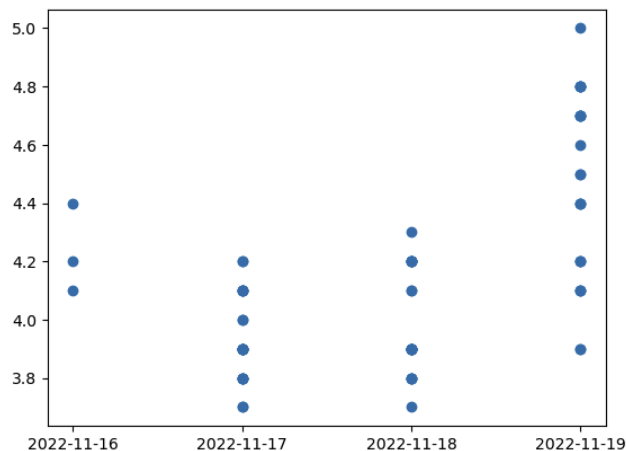
Now that I have my data imported, I can create some informative visualizations. For reference, I ran the air quality monitor without the air purifier on the 16th and 17th and with the air purifier on the 18th and 19th. We will see if there is a dramatic change in the data. There were some very interesting results.

Results

The first results that I want to study are the readings from the PM2.5 sensor over the time that the SDS011 was running:

```
In [11]: twofive = df['twofive']
day = df['day']
plt.scatter(day, twofive)
plt.show
```

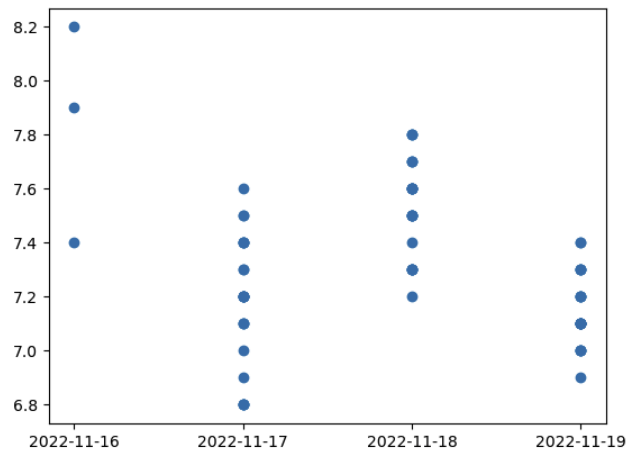
```
Out[11]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Next, I want to see the results of the PM10 sensor:

```
In [13]: ten = df['ten']  
day = df['day']  
plt.scatter(day, ten)  
plt.show
```

```
Out[13]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Combined, here is the comparison of readings:

```

In [59]: barWidth = .5
fig = plt.subplots(figsize =(12, 8))

# set height of bar
PMtwofive = df['twofive']
PMten = df['ten']
Date = df['day']

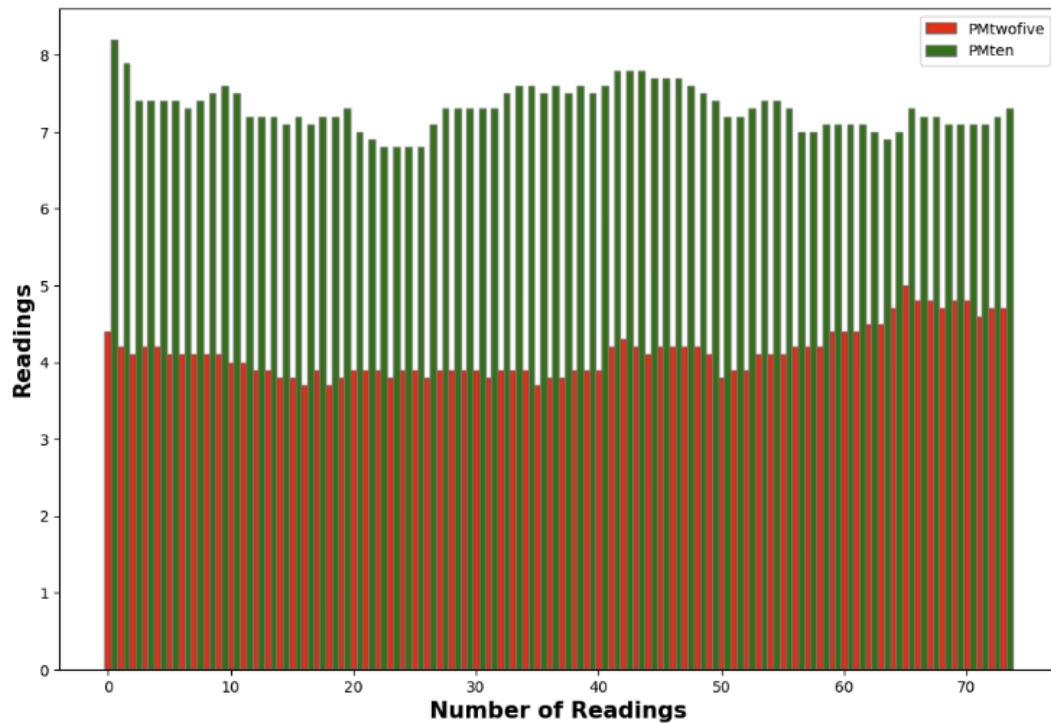
# Set position of bar on X axis
br1 = np.arange(len(PMtwofive))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, PMtwofive, color ='r', width = barWidth,
        edgecolor ='grey', label ='PMtwofive')
plt.bar(br2, PMten, color ='g', width = barWidth,
        edgecolor ='grey', label ='PMten')

# Adding Xticks
plt.xlabel('Number of Readings', fontweight ='bold', fontsize = 15)
plt.ylabel('Readings', fontweight ='bold', fontsize = 15)

plt.legend()
plt.show()

```



Conclusion

I think that my results were quite interesting. As we can see from the diagrams, the PM2.5 sensor experience a slight *increase* of particulates over time and the PM10 sensor experience a slight *decrease*. How can this be? I have a theory for the increase. Since the air quality monitor is placed in the same room as the air purifier, the particles that are being sucked into the air purifier may be passing through the air quality monitor at an increased rate. This is just a theory but does not explain the decrease of the PM10 particulates. That may be a result of changing air quality from outside as the rainy season here in Oregon freshens the air a bit and takes out some of the particulates. This can be due to a variety of factors, however. Since I wanted to ensure that all other factors remained the same, including normal use of doors and windows, this is a reasonable conclusion. Overall, the most important factor is that, according to the diagram of health quality that is pasted earlier in the document, all the readings are within a healthy and normal range, and I am confident that we are breathing healthy air. I think to enhance the project, a subject matter expert on air quality can help to identify better locations in our home to take measurements and indicated some educated reasons for the increase and decrease.

Overall Project

Skill Building

I did build some valuable skills while building this project that I believe will be valuable to other IoT endeavors. I learned that different sensors may have different ways of reading data and the it is important to test any sample code given before moving on to the final program. I learned that Flask web development was not as difficult as I thought it was. Web applications are transforming and becoming increasingly popular and knowing the basics gave me confidence to keep practicing. Always use libraries where possible to cut down on programming time. I learned about local Jupyter notebooks. I do not work with Jupyter notebooks in my daily tasks on the job, but I had always thought that they were online only. Knowing that they can be local is helpful in situations where internet access is non-existent, such as the Savannah! You can still make great visualizations that can be saved for later. You must, again, ensure that you have all the right libraries to visualize the data.

Innovation

As a model of air quality monitoring, I used the air purifier that we purchased. The model that we purchased has LED light to indicate air quality, but no measurements and there is no way to check the health of it and no way to visualize the air quality over time. We must trust that it is doing what it says it does. With the Raspberry Pi air quality monitor I built, you have much more opportunity to intervene if air quality becomes unhealthy through alerts sent to your phone, device health monitoring, and local tools for visualizing data. I think that if these were incorporated into the AirMega product, it would increase buyer satisfaction through increasing buyer confidence.

Scope

This project was, first and foremost, fun to work on. While, originally, I just wanted to measure air quality over time, I also addressed some valuable features as I went to ensure a more well-rounded project. I also tackled web development, which was new to me and improved on the skills I could have gained in Lab 2. I also took pause when considering incorporating some of the thought process from previous weeks, such as the security of the device and how the device would communicate with users “in the field”, or perhaps even “in the wild”. I thought about what would be necessary for network communication as well. Challenges included finding out how the SDS011 sensor sends data to the Pi and how to parse the bits, how to decide on what type of messaging should be used to communicate to the user and implementing long running scripts on the Pi. I thought about all the pieces since the project proposal and continued to test until I got it right. Testing this type of device involves investing a lot of time because of the nature of a monitoring device. Sometimes this meant testing for a day only to find out that some data was missing or misconfigured. I am proud of what I accomplished on my own, tackling new ideas. I hope you enjoy reading this project paper.

