Select IoT Devices and Sensors: Choose appropriate sensors for measuring air quality parameters such as pollution levels, particulate matter (PM2.5, PM10), temperature, humidity, and more. Common sensors for this purpose include gas sensors, dust sensors, and environmental sensors.

IoT Device Selection: Select IoT devices like Raspberry Pi, Arduino, or ESP8266/ESP32 that can interface with these sensors. Ensure that the chosen device supports Python programming.

Wiring and Connections: Connect the sensors to the IoT device following their respective datasheets and pin configurations.

Data Transmission: Implement a method to transmit the collected data to a data-sharing platform. This could be a cloud service like AWS IoT, Google Cloud IoT, or a custom server. You can use MQTT, HTTP, or other communication protocols for this.

Data Security: Ensure that your data transmission is secure, especially if it involves sensitive data. Implement encryption and authentication mechanisms.

Data Visualization: Consider how you'll visualize the data. You can use platforms like Grafana, ThingSpeak, or custom web applications for this purpose.

Testing: Thoroughly test your IoT device, sensors, and the Python script to ensure they collect and transmit data accurately.

Documentation: Keep detailed documentation of your project, including schematics, code, and setup instructions.

Scaling: If needed, replicate this setup with multiple IoT devices to cover a broader area.

Python Scripting:

```
import time
import serial  # For reading sensor data via serial port

# Replace this with your actual sensor details
SERIAL_PORT = '/dev/ttyUSB0'  # Example serial port for SDS011 sensor

def read_sensor_data():
    ser = serial.Serial(SERIAL_PORT, 9600)
    ser.flushInput()
    data = ser.read(10)
    ser.close()
    return data

def parse_sensor_data(data):
    if len(data) == 10 and data[0] == 170 and data[1] == 192:
        pm25 = data[2] + data[3] * 256
        pm10 = data[4] + data[5] * 256
        return pm25, pm10
    else:
        return None, None

if __name__ == '__main__':
    while True:
        try:
            sensor_data = read_sensor_data()
            pm25, pm10 = parse_sensor_data(sensor_data)
            if pm25 is not None and pm10 is not None:
                print(f"PM2.5: {pm25} µg/m³")
                print(f"PM10: {pm10} µg/m³")
            else:
                print("Invalid data received")

            time.sleep(60)  # Adjust the frequency of data collection
        except Exception as e:
            print(f"Error: {e}")
```