**Parts and Accessories**

Parts

|  |  |  |
| --- | --- | --- |
| Part | Part Number | Price |
| Raspberry Pi Zero W\* | https://www.adafruit.com/product/3400 | $10 |
| Analog to digital Converter | https://www.adafruit.com/product/1085 | $14.95 |
| GPS Module | https://www.adafruit.com/product/746 | $39.95 |
| Temperature Sensor | https://www.adafruit.com/product/1782 | $4.95 |
| Micro SD | Amazon | $8.93 |

\*Any Raspberry Pi will work. A Raspberry Pi 3B with Wifi dongle

may be a better choice, especially if you are planning on using it to drive a driver dashboard.

Tools

A true master can build with any tools. However proper tools make the job much easier. There are many brands, but an investment in good tools is worthwhile. Below is what you will find on my desk.

|  |  |  |
| --- | --- | --- |
| Tool | Part Number | Price |
| Wire Strippers | https://www.adafruit.com/product/527 | $14.95 |
| Flush Cutters | https://www.adafruit.com/product/152 | $7.25 |
| Plier | <https://www.adafruit.com/product/1368> | $11.95 |
| Tweezers | https://www.adafruit.com/product/421 | $3.95 |
| Third Hand | https://www.adafruit.com/product/291 | $6.00 |
| Soldering Iron | Hakko FX888D | $96.72 |
| Magnifier Light | Amazon | $26.95 |

How this project you will also need a selection of hookup wire and connectors. What you need is going to be unique to how you setup your project.

**Step one**

Setup Raspberry Pi

First, we need to install Raspbian to a sim card and boot the Raspberry for the first time.

Then set the default version of Python to Version 3.5

|  |
| --- |
| alias python='/usr/bin/python3.5' |

Setup Raspberry Pi for Circuit Python. This allows us to use Adafruit’s libraries.

<https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi>

Library for Temperature Sensor

<https://learn.adafruit.com/adafruit-mcp9808-precision-i2c-temperature-sensor-guide/python-circuitpython>

Library for ADC

https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/python-circuitpython

Library for GPS.

<https://learn.adafruit.com/adafruit-ultimate-gps/circuitpython-parsing>

https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/uart-serial

|  |
| --- |
| pip install --upgrade pip  sudo apt-get install python-psycopg2 |

I generally do not do work on the Raspberry itself. I use a program called VNC to remotely access the Raspberry. VNC is installed by default on the Raspberry and is available free for almost any other platform. In order to connect you need to be connected the same network and know the Raspberry’s IP address. That is very hard to do if you don’t have a monitor setup. I have found the following script to be extremely helpful. You can set it up to run when the Raspberry boots and it will email you the current IP address.

Like most computers, the Raspberry Pi does not like to be shut off without a proper power down. Doing so risks corrupting the operating system. I highly recommend implementing a pust button shut down. There are good tutorials on the web. This is an example <https://www.hackster.io/glowascii/raspberry-pi-shutdown-restart-button-d5fd07>, however there are many ways to implement this.

import subprocess

import smtplib

from email.mime.text import MIMEText

import datetime

def **connect\_type**(word\_list):

*""" This function takes a list of words, then, depeding which key word, returns the corresponding*

*internet connection type as a string. ie) 'ethernet'.*

*"""*

if *'wlan0'* in word\_list or *'wlan1'* in word\_list:

con\_type = *'wifi'*

elif *'eth0'* in word\_list:

con\_type = *'ethernet'*

else:

con\_type = *'current'*

return con\_type

# Change to your own account information

# Account Information

to = *'your email'* # Email to send to.

gmail\_user = *you email@gmail.com'* # Email to send from. (MUST BE GMAIL)

gmail\_password = *'your password here'* # Gmail password.

smtpserver = smtplib.SMTP(*'smtp.gmail.com'*, 587) # Server to use.

smtpserver.ehlo() # Says 'hello' to the server

smtpserver.starttls() # Start TLS encryption

smtpserver.ehlo()

smtpserver.login(gmail\_user, gmail\_password) # Log in to server

today = datetime.date.today() # Get current time/date

arg=*'ip route list'* # Linux command to retrieve ip addresses.

# Runs 'arg' in a 'hidden terminal'.

p=subprocess.Popen(arg,shell=True,stdout=subprocess.PIPE)

data = p.communicate() # Get data from 'p terminal'.

# Split IP text block into three, and divide the two containing IPs into words.

ip\_lines = data[0].splitlines()

split\_line\_a = ip\_lines[1].split()

split\_line\_b = ip\_lines[2].split()

# con\_type variables for the message text. ex) 'ethernet', 'wifi', etc.

ip\_type\_a = connect\_type(split\_line\_a)

ip\_type\_b = connect\_type(split\_line\_b)

*"""Because the text 'src' is always followed by an ip address,*

*we can use the 'index' function to find 'src' and add one to*

*get the index position of our ip.*

*"""*

ipaddr\_a = split\_line\_a[split\_line\_a.index(*'src'*)+1]

ipaddr\_b = split\_line\_b[split\_line\_b.index(*'src'*)+1]

# Creates a sentence for each ip address.

my\_ip\_a = *'Your %s ip is %s'* % (ip\_type\_a, ipaddr\_a)

my\_ip\_b = *'Your %s ip is %s'* % (ip\_type\_b, ipaddr\_b)

# Creates the text, subject, 'from', and 'to' of the message.

msg = MIMEText(my\_ip\_a + *"\n"* + my\_ip\_b)

msg[*'Subject'*] = *'IPs For RaspberryPi on %s'* % today.strftime(*'%b %d %Y'*)

msg[*'From'*] = gmail\_user

msg[*'To'*] = to

# Sends the message

smtpserver.sendmail(gmail\_user, [to], msg.as\_string())

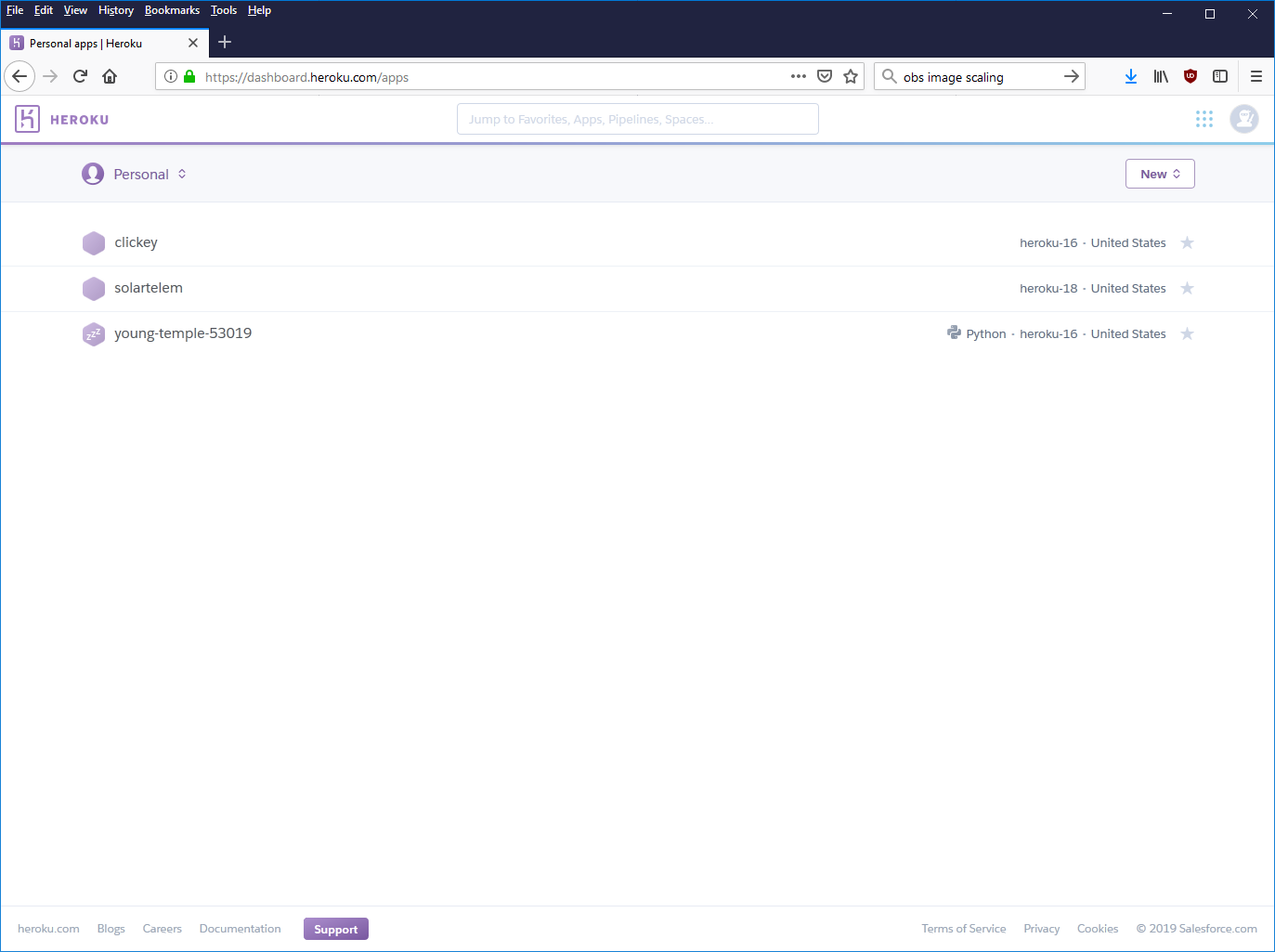
# Closes the smtp server.

smtpserver.quit()

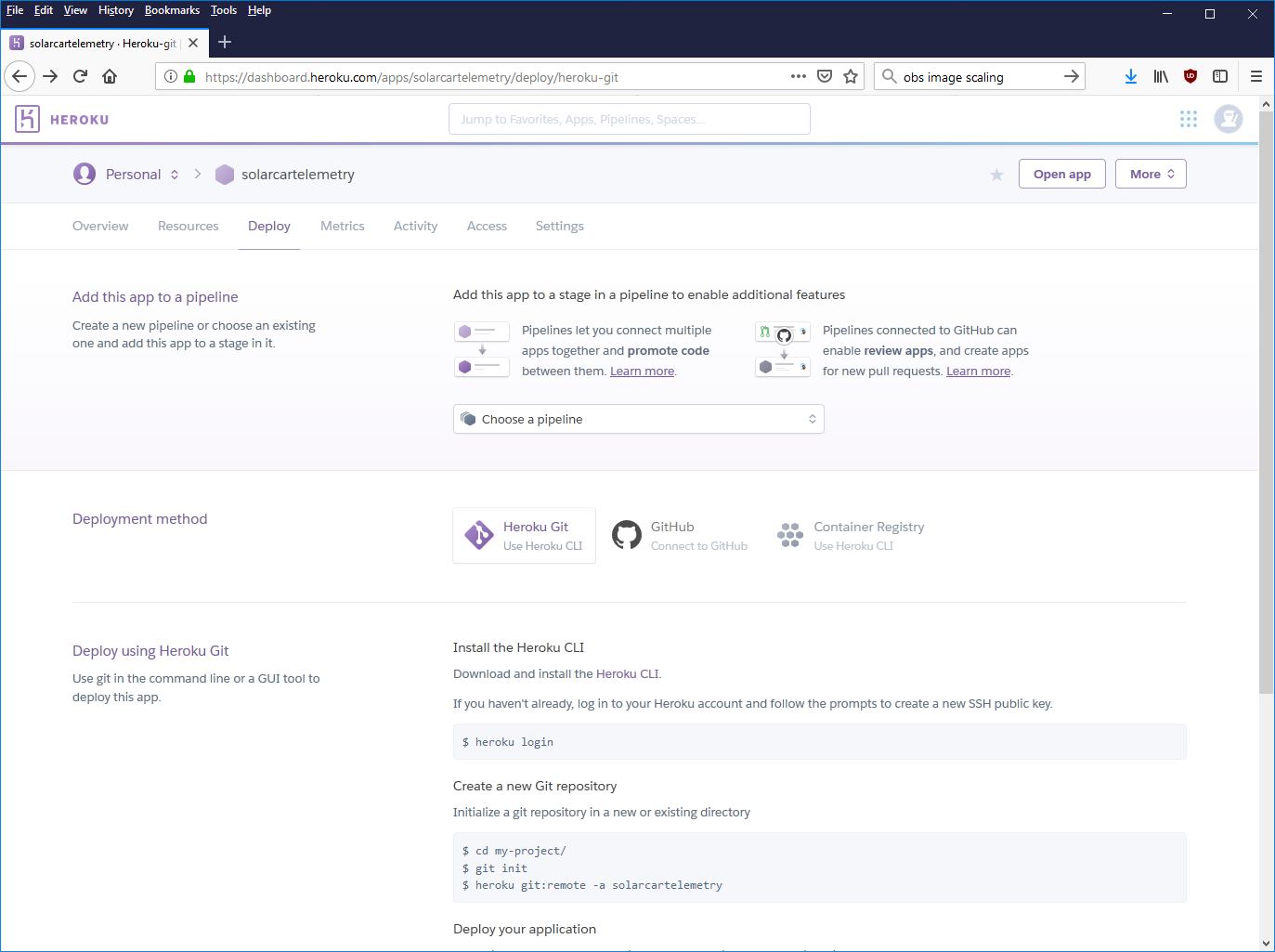
**Step Two**

Setup Your Database

I am going to walk through setting up Heroku <https://www.heroku.com/> as it is what I use. There are other free and paid services out there and your school may be able to host a database for you, but that is beyond the scope of this project. It is a free service that simplifies the setup of an Amazon Web Service server. Once you create an account you will come to the home page and click New bitton and select create a new app.



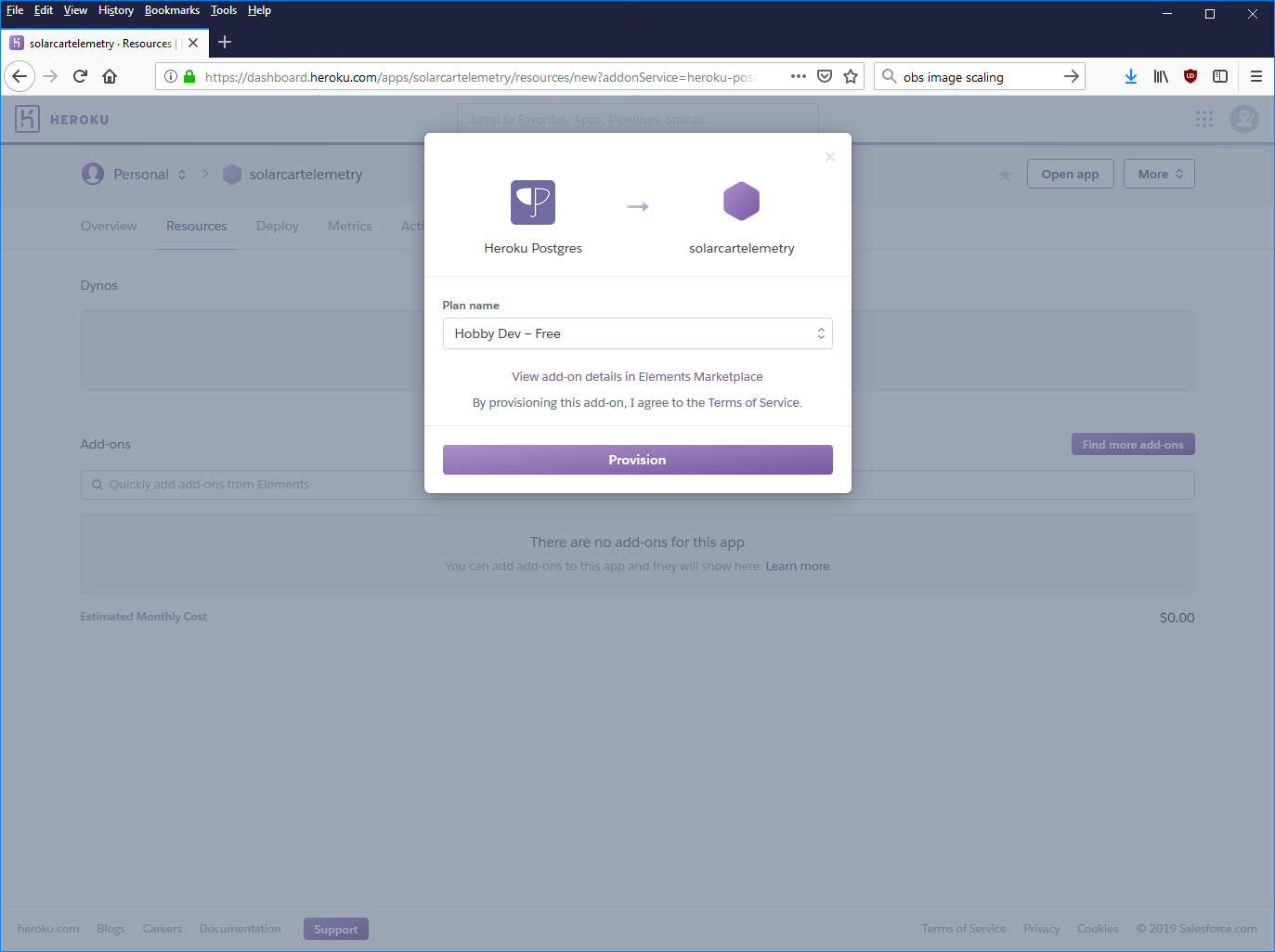
Once you give you App a name, you will come to the home page for the app. You will want to add a resource



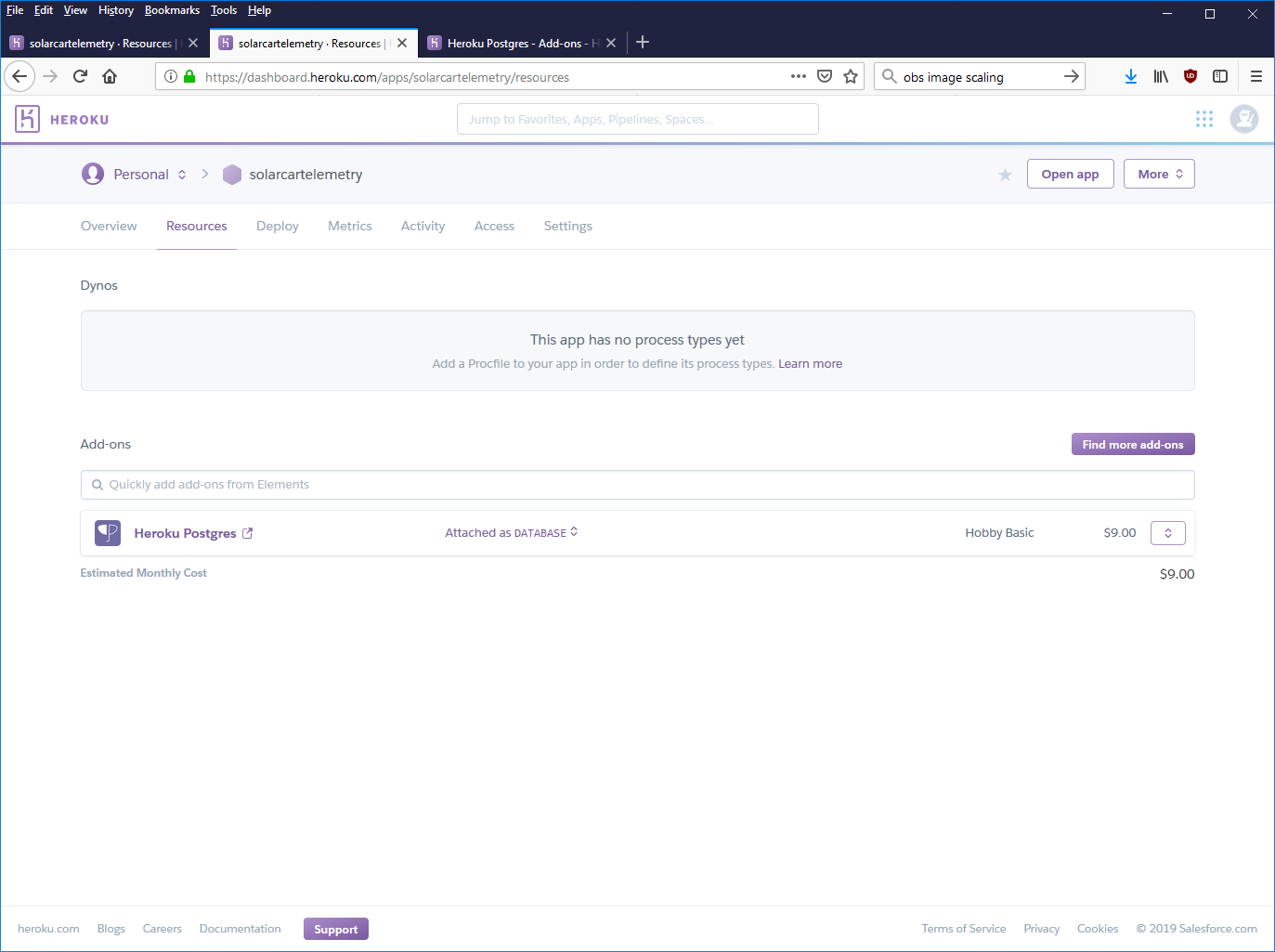
Type Postgres into search field and select “Heroku Postgres”



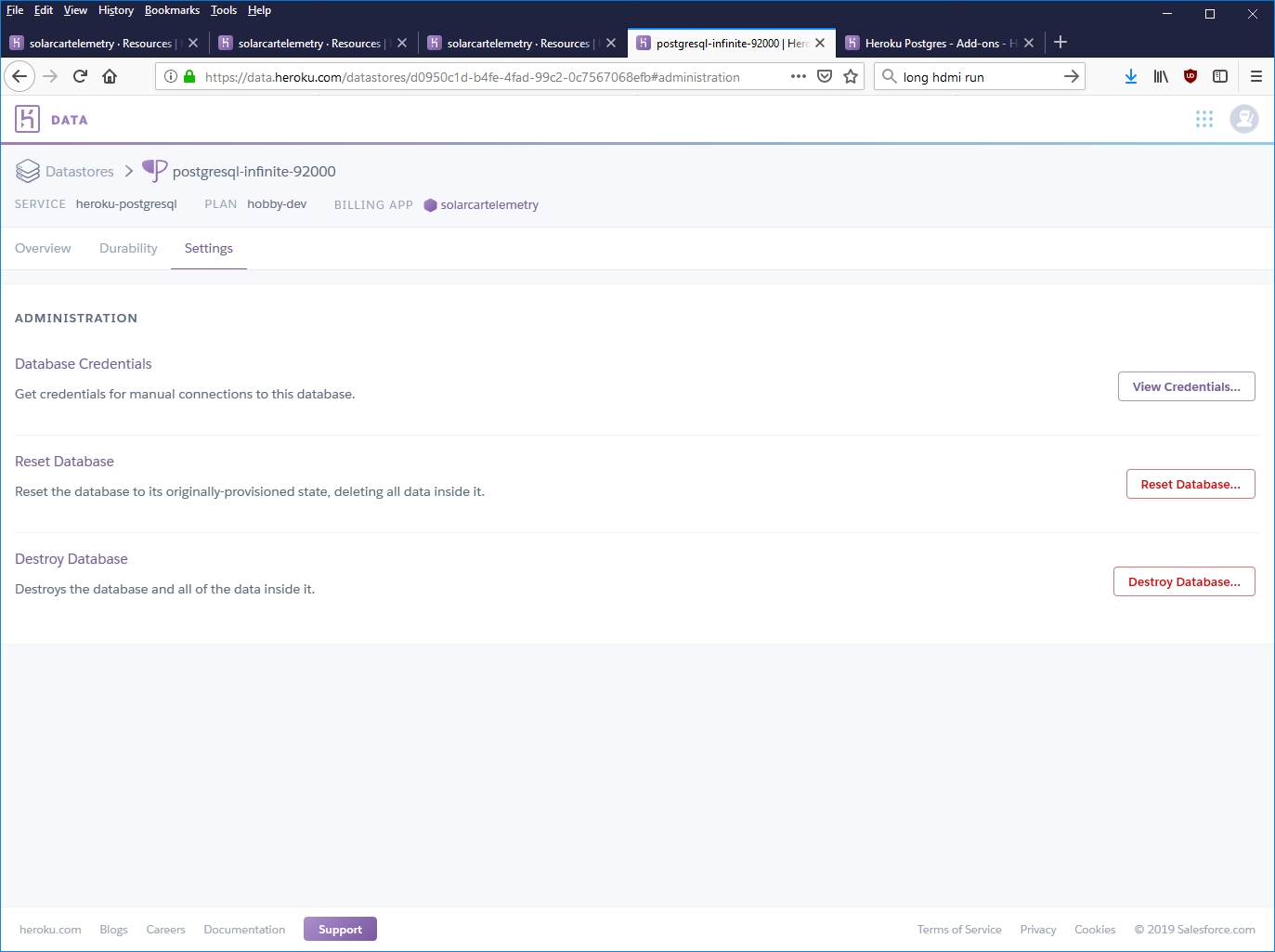
Select Hobby Dev -Free. This gets you a 10k line database. That is enough to store a little over 24 hour of data if updating every 10 seconds. Or just under three hours if updating every second. Heroku is a pay as you go service so. If 10k lines is to limiting, 10 million lines is only nine dollars a month and you can upgrade any time.



Click on your database



Click Setting and then View Credentials. This will give you your log in information. You will need it later. Heroku occasionally changes servers on the free tier. If your system stops working check back to make sure your log in information is still current.



**Step Three**

Provision Your Database

A database is just a series of tables. There are multiple ways to create tables fields in postgress. One of the simplest is to use a program like pgAdmin to access the database and use the GUI to add tables and fields in a manner similar to Excel.

A more powerful method is to use SQL statements. While SQL is beyond the scope of this walk through, a good introduction can be found here: <https://www.w3schools.com/sql/sql_create_table.asp>

One way to use SQL is a simple program that calls an SQL statement in the database. Below is a an example of a simple python script to create the database. You can change command to any SQL statement to modify or delete you table. Running this script after the database is created will destroy any data store.

import psycopg2

try:

conn = psycopg2.connect(

dbname=*'you db name '*,

user=*'your user name'*,

password=*'your password here'*,

host=*'your host here.*

*com'*,

port=*'5432'*)

print(*"connected"*)

cur = conn.cursor()

except:

print(*"I am unable to connect to the database."*)

commands= *"DROP TABLE solardata"*

cur.execute(commands)

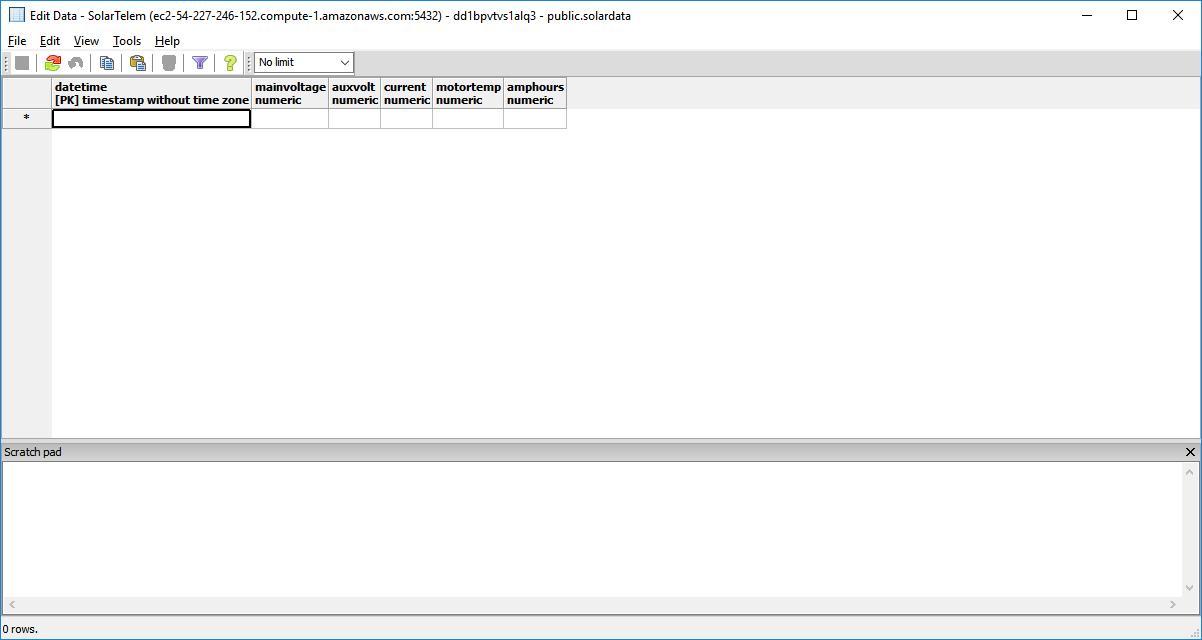
commands= *"CREATE TABLE solardata (datetime TIMESTAMP PRIMARY KEY, latitude DECIMAL, Longitude DECIMAL, mainvoltage DECIMAL, auxvolt DECIMAL, current DECIMAL, motortemp DECIMAL, amphours DECIMAL)"*

cur.execute(commands)

conn.commit()

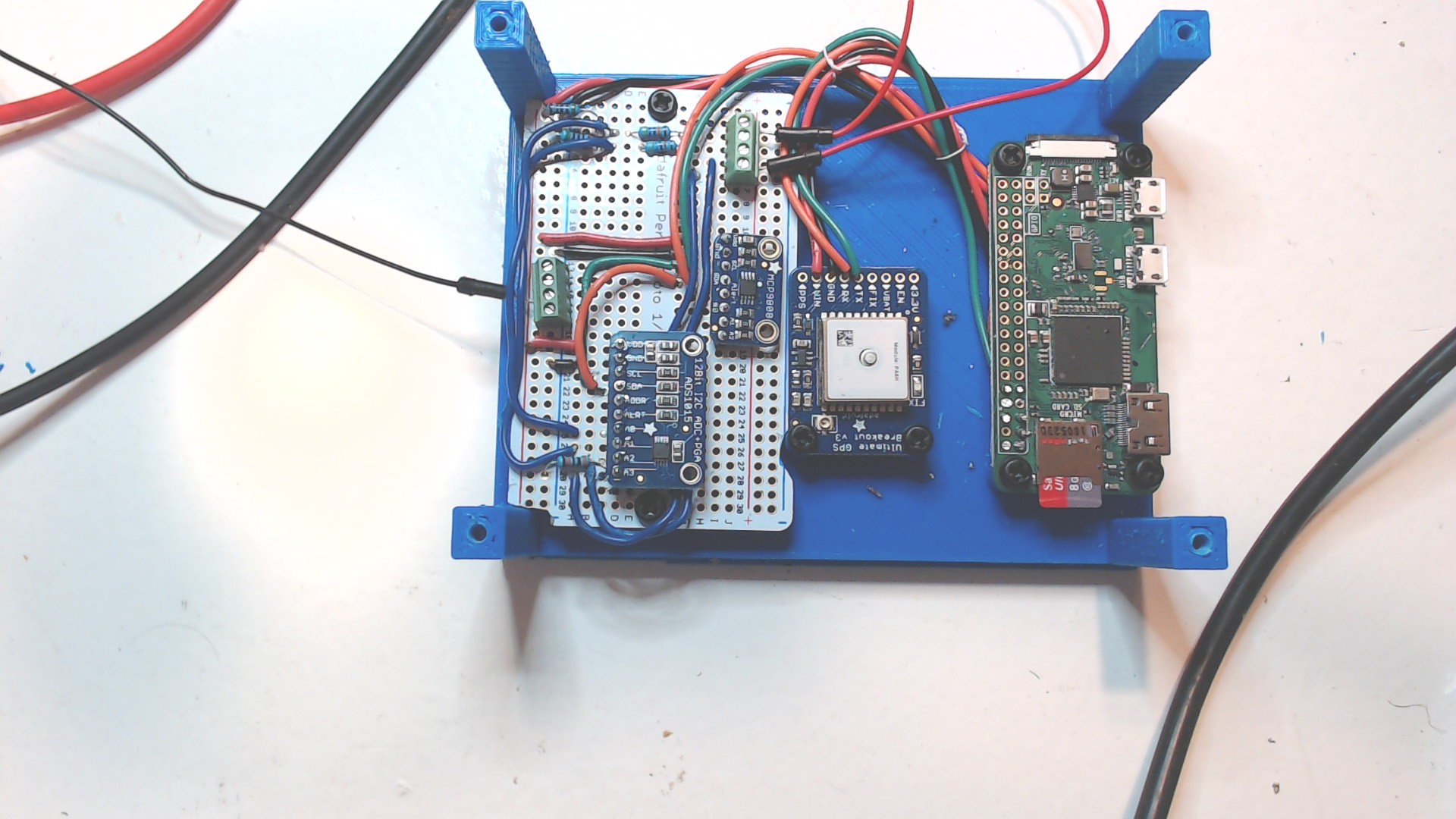
conn.close()

You table will look something like below. This table is missing the latitude and longitude columns but is close. A final version is available on the GitHub site for this project.



**Step Four**

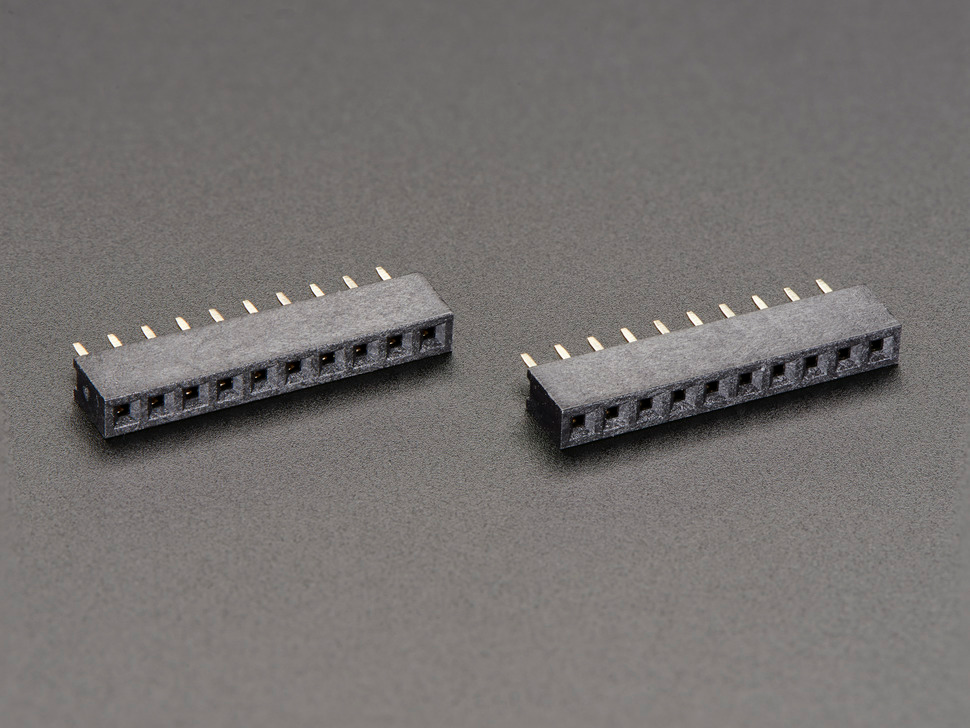
Wiring



Some notes before we start on the wiring. This is an arena where neatness counts. Things will go wrong you will save yourself an inordinate amount of time, energy, and suffering by being meticulous in this realm. I highly recommend using multiple colors and labeling your wiring. There is little more frustrating then having everything but one sensor working and trying to identify it’s wire among a sea of red wiring. For this project I used red for +5v black for ground, blue for data, green for SCL, and orange for SCA. There is nothing sacred about that color scheme, but write it down and be consistent.

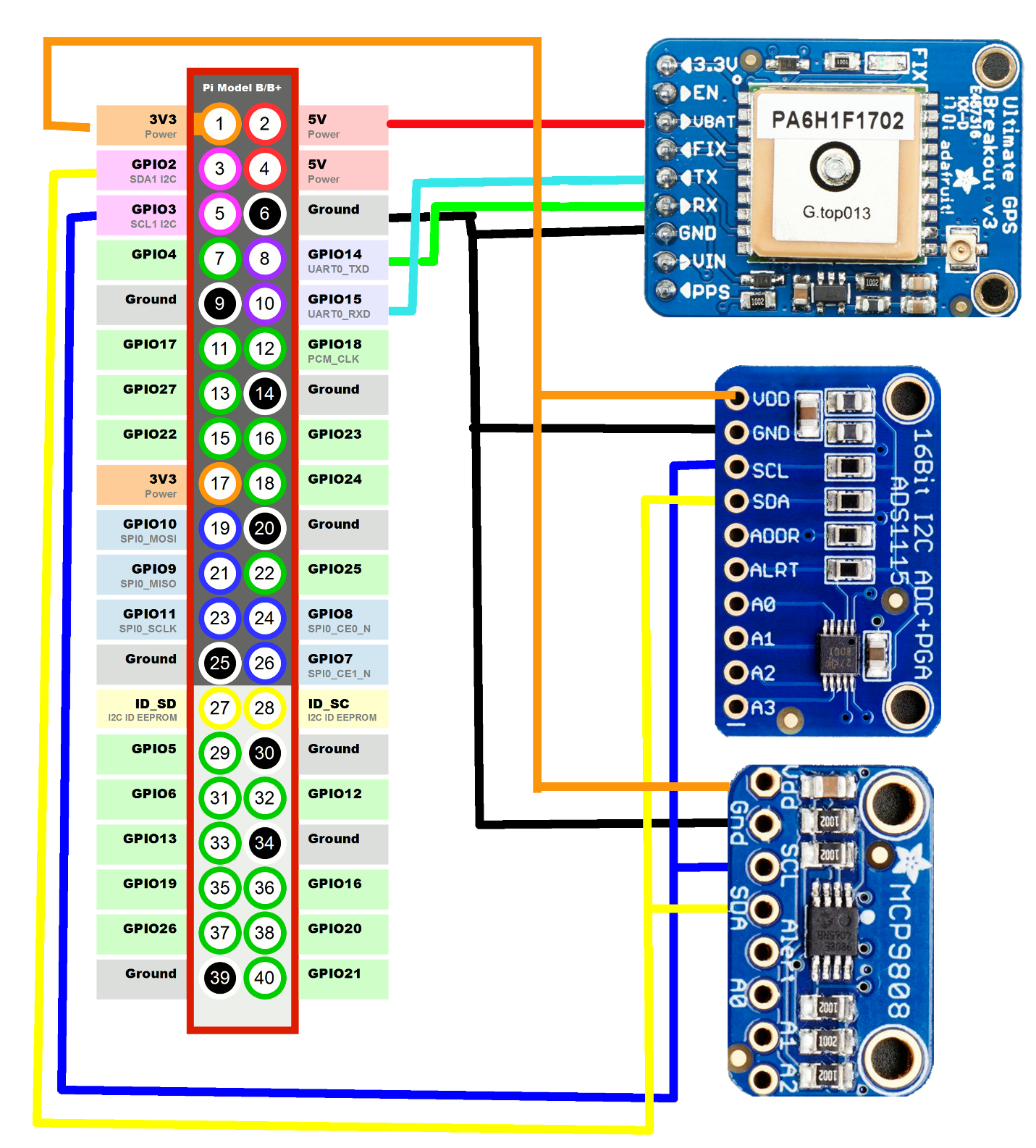
Because I only built a demonstration board and not an entire solar car, all of my connections are hardwired. I highly recommend against this course of action for your car. It is much easier to change things if you have connector, sockets, and screw terminals instead of soldering every connection. I accidentally blew up my analog to digital converter in testing, instead of being a 30 second fix if it was socketed, it ended up with 45 minutes to de-solder the old one and re-solder the new

Header Sockets:



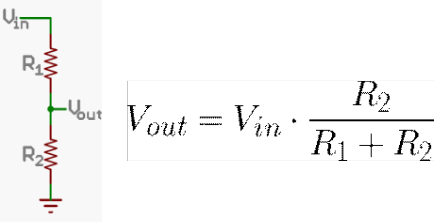
For connector I tend to favor Deutsch or Weatherpack. However, both are expensive and require dedicated tools. Another of my favorites is aviation panel connectors. There are lots of very reasonable priced connectors out there on Alibaba and Amazon. Two things to look for. First, that it is indexed so you can’t hook it up backwards. Hooking power to the data line is a quick way to blow things up. Second the connection is secure/locking in some way so it doesn’t come apart as the car vibrates.

Below is the basic wiring diagram:

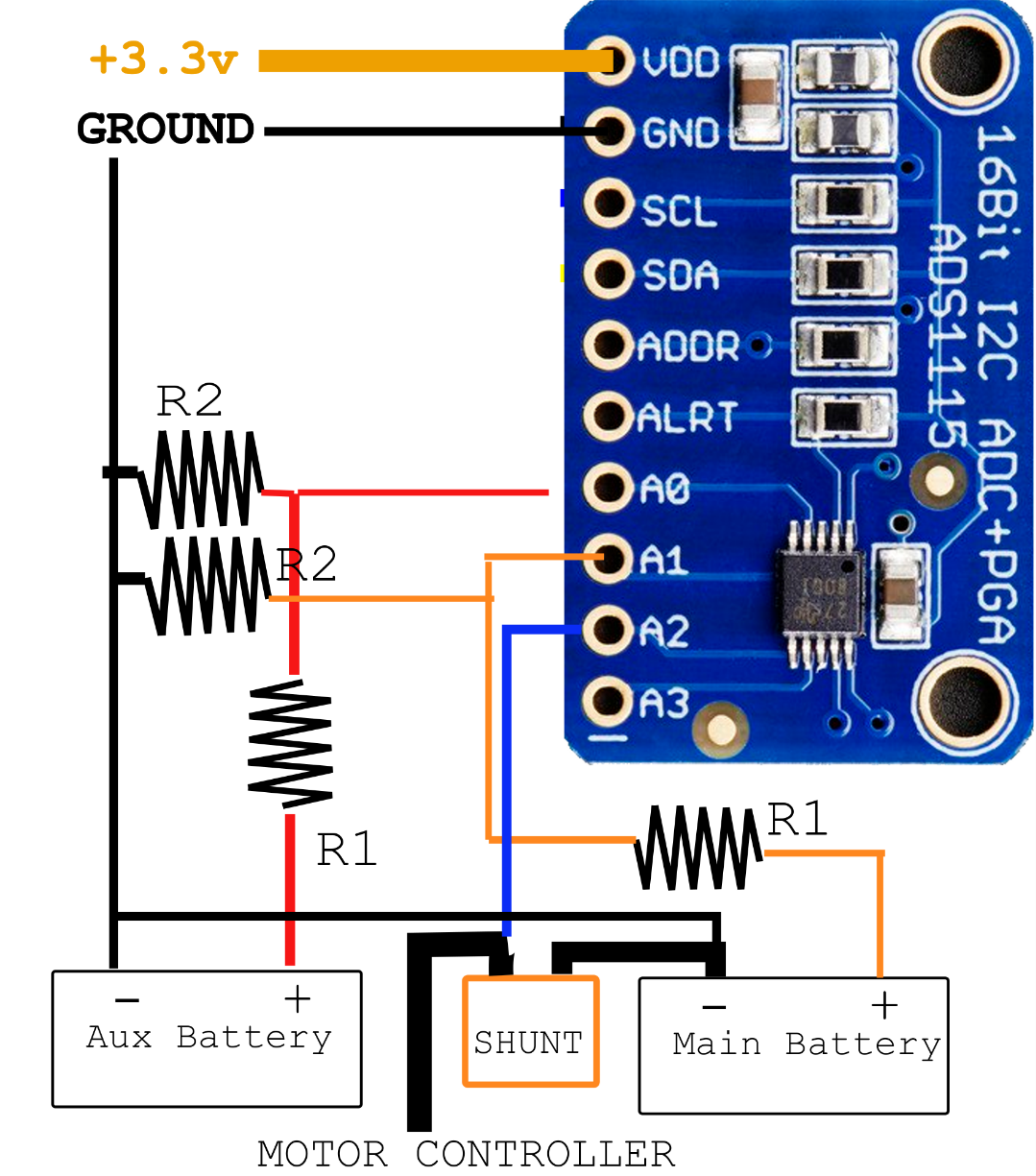


You can add as many I2C devices as you like so long as they have unique address. There are also lots of opportunities to expand. There are many unused GPIO pins on the Raspberry Pi that could be used to control switches, indicators, LEDs, etc. The Pi also has a USB port available as yet another way to receive data.

The analog can only take in five volts. At first glance it seems that might not be useful when the majority of the signals we are interested in start at +12 volts and a go to +48 volts or higher. Thankfully there is an easy way to scale any voltage to a readable voltage using two resisters. Using excel you can determine two resisters to take your system voltage from its actual voltage down to less then five volts. It is important to remember this scaling factor as you will need it to correct the voltage that your system reads back to actual voltage



ADC Wiring



**Step Five**

Program the Raspberry

The entire source code is available on the GitHub page for this project at <https://github.com/SolarCarChallenge/SolarCarTelemetry2019>.

The Tests folder contains a couple of individual programs that test the individual components of the system. This allows you to test the individual component with a simple program before combining all of the elements into the final program. These also so the implementation of each sensor in case you want to add additional sensors.

You will also find the file makeadatabase.py. The name is descriptive of what the file does. If you have setup Heroku from above, all you need to do is enter you db name, username, password, and server into the script and it will create an empty database for you.

There is also a text file called amphour.txt. This is used by the program and user to store the current number of amphours. If you don’t agree with the programs counter, you can change it by modifying the value in the file.

The main program is called Telemetry.py. This program pulls all the individual sensor data and stores it in the database. As of this writing, the program has no output to the user. It also has to be started manually. This presents a problem.

The simplest solution is to add a screen and a touchscreen mouse. However, that might not be the most cost effective or user-friendly idea for a solar car. However, it does leave you the option of building a full featured dashboard for your driver.

The second option is to use VNC to connect to the car and have a someone remotely start the program using a laptop. However for this to work, both devices must be connected to the same network. If something happens on the track, you have no recourse but to pull in.

The final option is to have the program run on start up and run all the time. <https://www.raspberrypi.org/documentation/linux/usage/rc-local.md> Be sure to take note of the warning to fork the process, or your raspberry will never boot. Assuming your database has room to store the data, there is very little downside to this option. If you are on the free tier of Heroku, it could eat up all of your available database space.

A far more elegant option is to have a program run at startup to monitor a GPIO pin and start the program when a button is pushed. To take this to the extreme, you could have a button push start and stop the program. However, this becomes a much more complicated problem. You either have to keep track of the process name (requires a much higher level of programming) or store a variable in a place both programs can access it (either a specific memory location or a simple textfile).

Even if you do not pursue this option, I highly recommend implementing a pushbutton shut down. Like most computers, Raspberry Pis do not like to be powered off without a proper shutdown. More then likely you can get away with it, but there is always a chance of corrupting the memory card and making the system unbootable. This always happens at the worst time. This could be added to the Telemetry.py program or run as its own program. An example can be found: <https://core-electronics.com.au/tutorials/how-to-make-a-safe-shutdown-button-for-raspberry-pi.html>

References:

<https://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/>

<https://circuitpython.readthedocs.io/projects/ads1x15/en/latest/>

<http://goinkscape.com/how-to-create-a-tachometer-in-inkscape/>

Software:

QT Creater Community Edition <https://www.qt.io/download>

InkScape <https://inkscape.org/>

Atom <https://atom.io/>