Python - Basic Example

PI Web API Connection

Installing Modules

Requests

The PI Web API requires Basic Authentication headers in order to connect. In Python there aren't built-in functions that can handle connecting to a server with authentication headers, so we need to install a package that will handle this for us. A package that can do this is `requests`. To install `requests`, follow the installation guide on the `requests` website.

Now we can load the module in our script by adding the following line at the beginning the of our program.

```
import requests
```

This will make all the functions that `requests` provides available to us in our script. The main function we are interested in is the `requests.get()` function which will, given the required username and password, handle the authentication headers required to connect to the PI Web API.

```
response = requests.get(url=url, auth=requests.auth.HTTPBasicAuth("USERNAME","PASSWORD"))
```

Pandas

A popular module used in data science is the `pandas` module. This package provides easy-to-use data structures and tools for python. We will be taking advantage of the `pandas` data frame data structure as well as the time formatting tools it provides. To install `pandas` follow the installation guide provided on the `pandas` website. After installing the module we can load in our script by adding the following line to the beginning of our program

```
import pandas
```

Matplotlib

A useful module that provides tools for plotting and visualization of data is the `matplotlib` module. To install `matplotlib` follow the installation guide on the `matplotlib` website. After installing the module we can import it at the beginning of the script. The main function we are interested in for this example is the `pyplot`, so we can directly import that function as follows

```
from matplotlib import pyplot
```

Datetime

An important module that comes installed with python is the `datetime` module. This module provides with functions that can extract information such as time, date, time zone, etc. from a datetime object. Because the module comes with python there is no extra installation step we just have to add it to the beginning of our script. For this example we only need the `datetime` class from the `datetime` module, so we can import it directly.

```
from datetime import datetime
```

Dateutil

In order to obtain the local time zone we need to call on another module that comes with the python installation, `dateutil`. We just have to add the following the beginning of our script

```
import dateutil
```

IO

In order to read the data returned from the PI Web API directly in a data frame we have to use a `StringIO` object. We have to import the `io` module to use the `StringIO` class, so we just have to add the following line to the beginning of our script.

```
import io
```

Obtaining URL

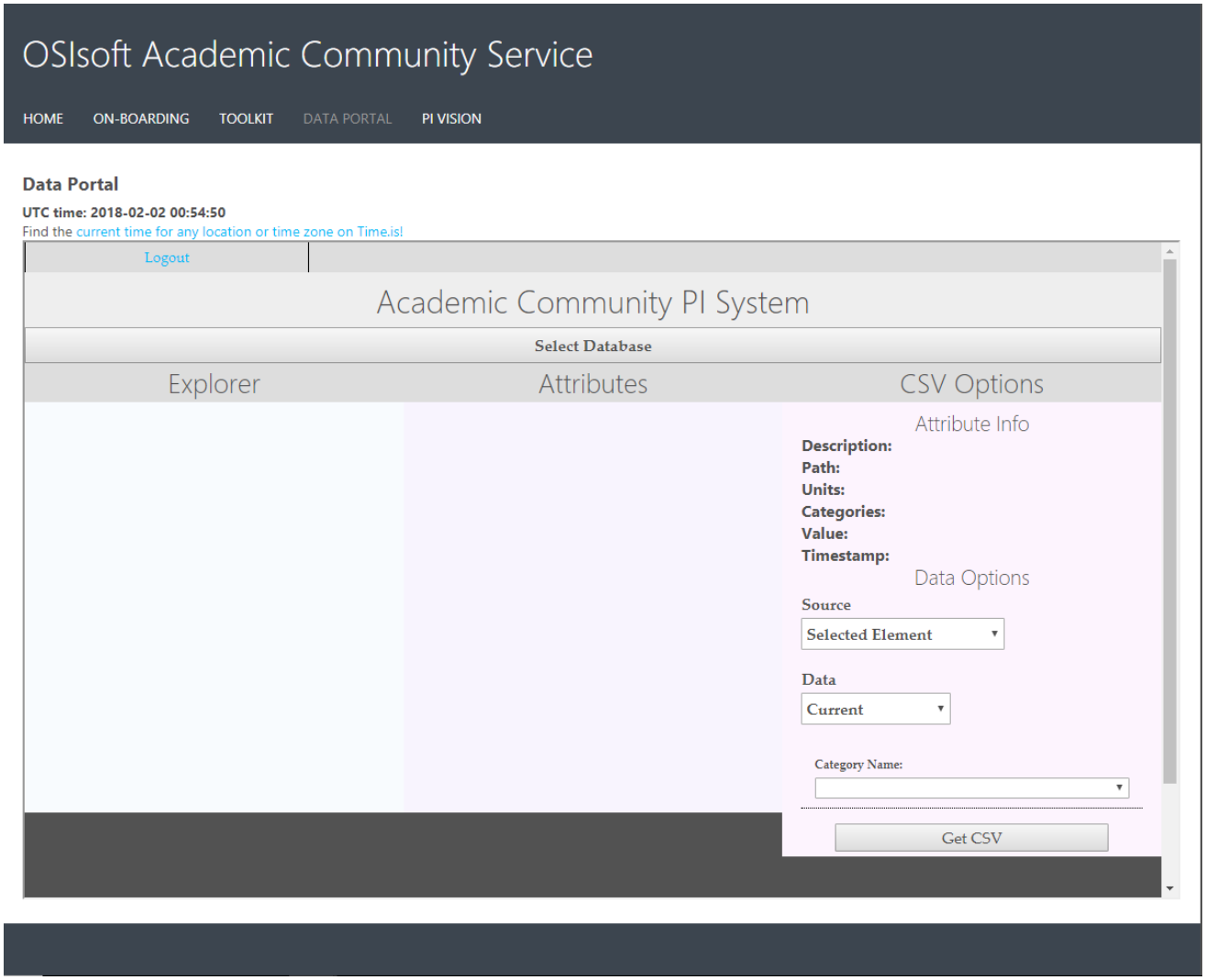
Connecting to the CSV controller requires a URL to the controller which is,

```
controller_url = "https://academicpi.osisoft.com:444/piwebapi/Csv/ElementInterpolated"
```

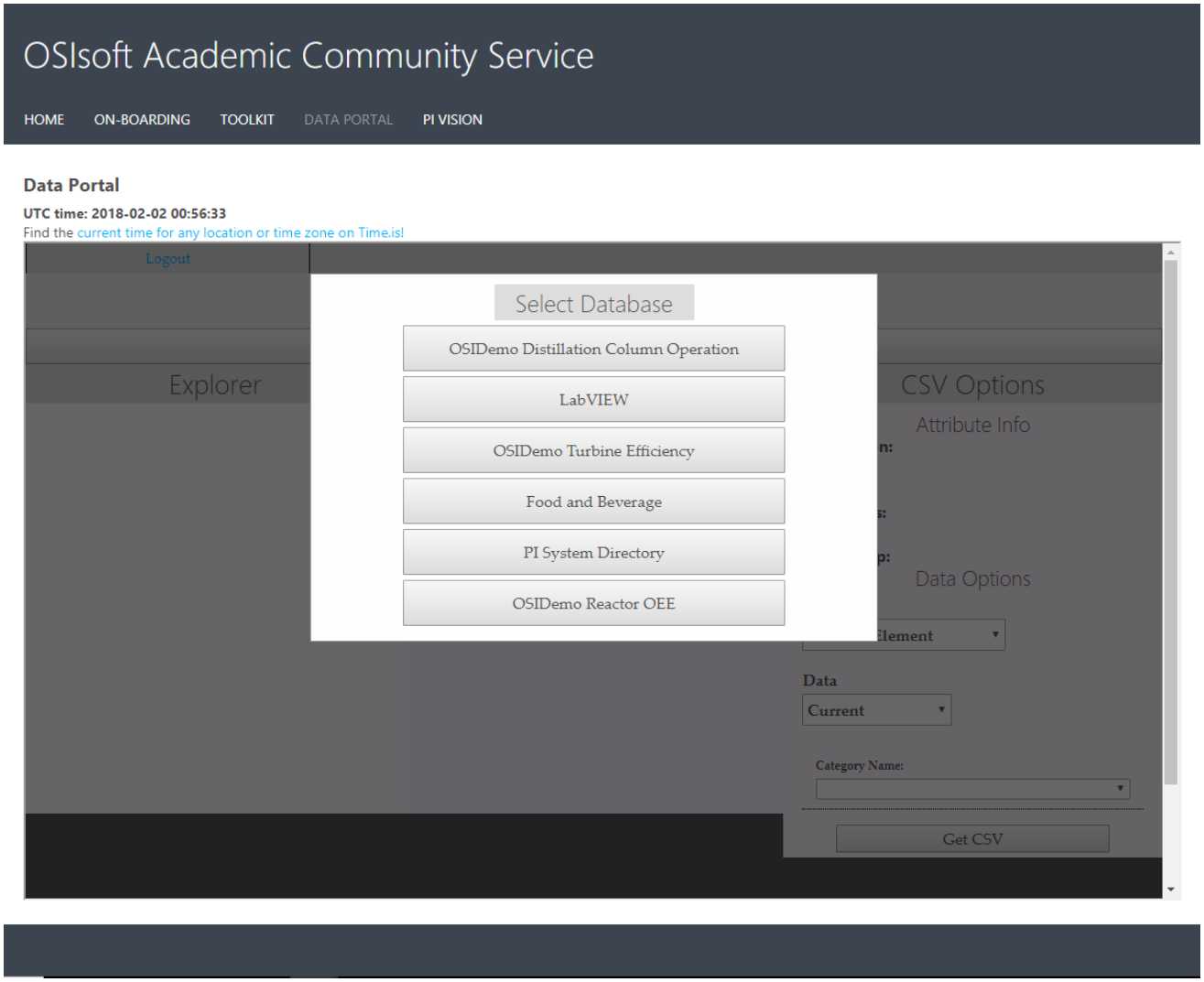
Also required is the element path to the desired asset.

Getting Element Path

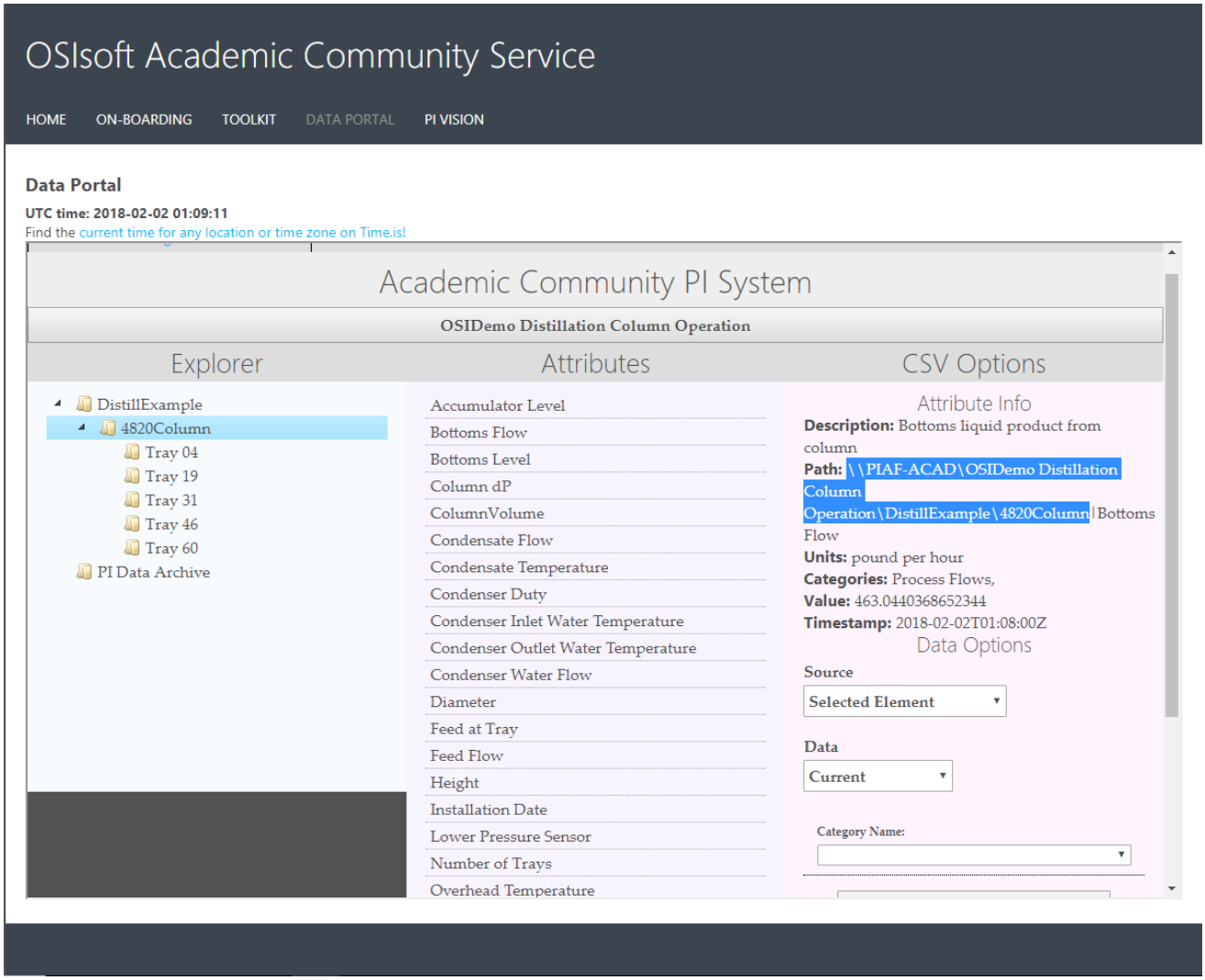
The element path can be found in the Data Portal tool on the ACS webpage. After logging in to the data portal you'll see the following window.



Select the database containing your assets. In this example we are using the "OSIDemo Distillation Column Operation"



Under the left panel titled "Explorer" select the desired asset. When you hover over one of the Attributes in the center panel the attribute info will appear in the right panel titled "CSV Options". The element path can be seen in the "Path" field in the attribute info. The path that we require is everything before the vertical bar, `|`, character. See the highlighted text in the image below.



We must also add `?path=` to the beginning of the element path before combining it with the controller URL. Using the distillation column, the following is an example of the required element path.

```
element_path = "?path=\\\\PIAF-ACAD\\OSIDemo Distillation Column
Operation\\DistillExample\\4820Column"
```

Optional parameters

The following parameters can be appended to the element path to request specific data from the PI Web API.

startTime

The starting time of the data collection can be specified, and if no start time is specified the default time of 24 hours before current time will be used. So, for example, if we want the data starting at 5 AM on January 30, 2018 we can append `startTime=01/30/2018 05:00:00` to the end of the element path and join it with an ampersand in between.

```
element_path = "?path=\\\\PIAF-ACAD\\OSIDemo Distillation Column
Operation\\DistillExample\\4820Column&startTime=01/30/2018 05:00:00"
```

To see more valid time formats visit the time format help section.

Note: Start time and end time parameters are considered to be in the user's local time

endTime

Similar to the start time parameter, we can also specify the end time of the data, and if no end time is specified the current time will be used as it's default value. So, if we want the data collected between 5 AM January 30, 2018 and 2 PM January 30, 2018 we can append the end time the same way we did the start time.

```
element_path = "?path=\\\\PIAF-ACAD\\OSIDemo Distillation Column
Operation\\DistillExample\\4820Column&startTime=01/30/2018 05:00:00&endTime=01/30/2018 14:00:00"
```

interval

The interval parameter can be used to request data from the PI Web API at regularly spaced timestamps. If no interval is specified the default value used is one hour. Adding to our distillation column example we can specify that we want the event data every 10 minutes between 5 AM and 2 PM on January 30, 2018.

```
element_path = "?path=\\\\PIAF-ACAD\\OSIDemo Distillation Column
Operation\\DistillExample\\4820Column&startTime=01/30/2018 05:00:00&endTime=01/30/2018
14:00:00&interval=10m"
```

nameFilter

We can specify which attributes we would like returned by the PI Web API using the name filter parameter. If no name filter is given, data for all of the attributes will be returned. If we wanted all of the types of flow data we can use `nameFilter=*flow*`. This would specify that we want data from every attribute that contains the word flow in the name. We can then append the parameter in the same way we did with the previous ones.

```
element_path = "?path=\\\\PIAF-ACAD\\OSIDemo Distillation Column
Operation\\DistillExample\\4820Column&startTime=01/30/2018 05:00:00&endTime=01/30/2018
14:00:00&interval=10m&nameFilter=*flow*"
```

We can then combine the controller URL and element path as follows.

```
url = controller_url + element_path
```

Now we can get the data from the PI Web API with the encoded URL using the previously mentioned `requests.get()` function. We do not need to encode the URL ourselves because the `requests.get()` function handles encoding for us.

```
data_response = requests.get(url=url, auth=requests.auth.HTTPBasicAuth("USERNAME","PASSWORD"))
```

The value returned by `requests.get()` is the CSV file in the form of a Response class.

Time Zone

When we made the data request to the PI Web API it used our local time zone when retrieving the date, but the timestamps that are returned by the PI Web API are in the UTC time zone. We can convert the timestamps to our local time zone..

Data Processing

Now, that we have obtained the CSV from the PI web API we need to process the data into something useful like a data frame. To do this we can use the `read_csv` function from the `pandas` module as follows.

Method 1: Directly creating data frame without saving to a file

We can create a data frame directly from the response of the PI Web API. To do this, we have to use a file-like class, `StringIO`, which is provided by the `io` module. After loading the text of the response into a `StringIO` object, we can then pass that object into the `read_csv` function provided by the `pandas` module.

```
data_buffer = io.StringIO(response.text)
data_frame = pandas.read_csv(data_buffer, parse_dates=["Timestamp"])
```

Method 2: Saving to a File

Another way to create a data frame is by saving the response to a file. Because the data was returned by the `requests.get()` function as a `Response` class we must write the data to a file which can then be passed into the `read_csv` function. It is also important to pass the `'wb'` parameter when opening the file because it specifies that we can write to it and the type we are writing.

```
filename = "flow_data.csv"
with open(filename, 'wb') as file:
    file.write(response.content)
data_frame = read_csv(filename, parse_dates=["Timestamp"])
```

The `read_csv` function also can convert our timestamps from a string to a Datetime class for us by telling it which the column contains dates. Giving the `parse_dates` parameter the timestamp columns tells the `read_csv` function that the timestamp column contains dates and to convert them from strings to a date-time class.

Simply calling the `read_csv` function as follows will result in an error because the function would expect a file name.

```
data_frame = read_csv(response)
```

Time Zone Conversion

When requesting data from the PI Web API the request is made in the local time zone, but the timestamps are returned in UTC. In python when we created the timestamp date-time classes by default they are known as "timezone-naive", so the first thing we have to do is make them "timezone-aware". To do this we can use a function from the `pandas` module.

```
data_frame["Timestamp"] = data_frame["Timestamp"].dt.tz_localize('utc')
```

This tells the timestamps that they are in the UTC time zone. Now that the timestamps are "timezone-aware" we can convert them to our local time zone. First, we get our local time zone using the `dateutil` package's `tz.tzlocal()` function. Then we can convert the timestamps into the local time zone using the `dt.tz_convert` function from the `pandas` module.

```
local_timezone = datetime.now(dateutil.tz.tzlocal()).tzinfo
data_frame["Timestamp"] = data_frame["Timestamp"].dt.tz_convert(local_timezone)
```

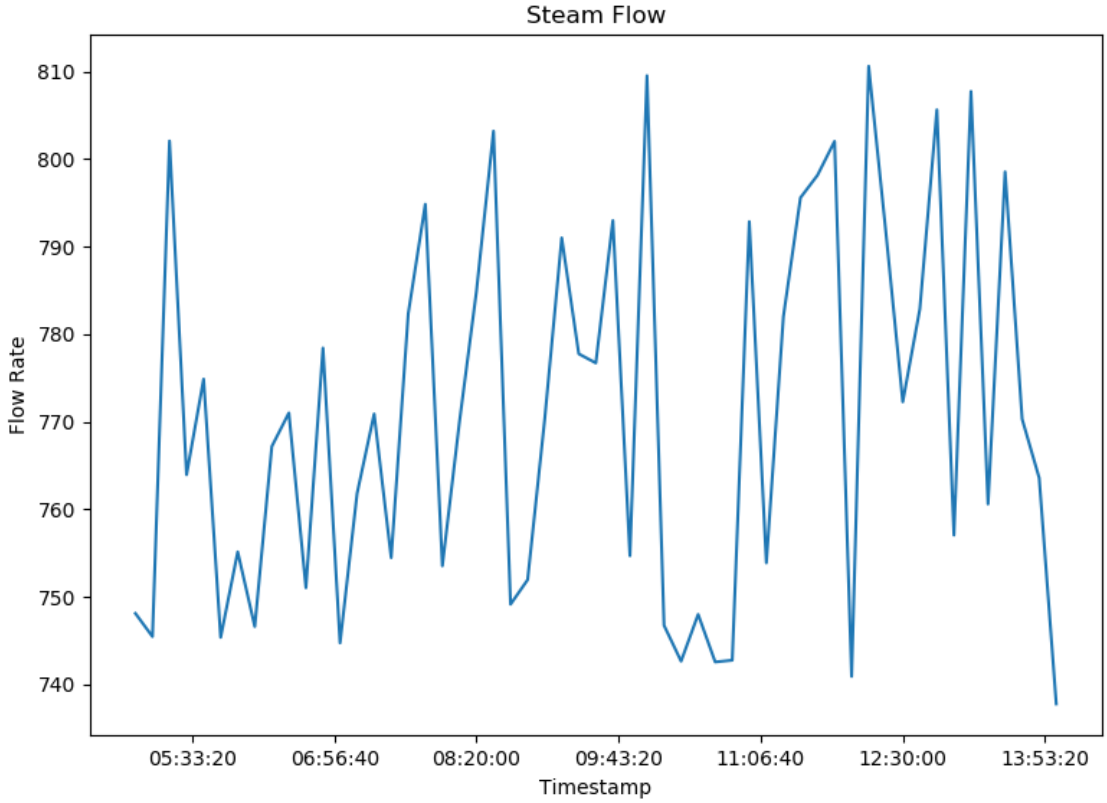
Plotting

Now that we have processed the data from the PI Web API request we can plot it using the `pyplot` function from the `matplotlib` module. In this example our timestamps cover only nine hour range so it isn't very useful to have the date on all of the x-axis markings. We can extract the time from the timestamps and use that instead. To do this, we can call the `datetime.time` method on every element in the timestamp column of our data frame utilizing the `apply` method that `pandas` provides.

```
time_labels = data_frame["Timestamp"].apply(datetime.time)
```

The following is a plot of the Steam Flow over our requested time range with an x-axis label of "Timestamp", y-axis label of "Flow Rate". For more information about the parameters that can be given see R's documentation of the `plot` function.

```
pyplot.plot(time_labels, data_frame["Steam Flow"])
pyplot.xlabel('Timestamp')
pyplot.ylabel('Flow Rate')
pyplot.title("Steam Flow")
pyplot.show()
```


Steam Flow
Flow Rate
810
800
790
780
770
760
750
740
05:33:20
06:56:40
08:20:00
09:43:20
11:06:40
12:30:00
13:53:20
Timestamp