UNIVERSIDAD TECNOLÓGICA NACIONAL

Facultad Regional de La Plata

Erik Kjölhede D'Annunzio

Visualización de datos con DASH

Índice

Resumen

Es común, en el campo de la ciencia, que una vez obtenidos los datos de un experimento, el investigador se enfrente a la tarea de realizar gráficas para tener una visión más clara de los mismos. Este proceso se ha logrado resolver de manera eficiente con la creación de softwares específicos para cada problemática. En su mayoría, para poder utilizarlos se deben adquirir licencias que dependiendo el área pueden ser costosas, a esto se le suma que para poder compartir el trabajo con un colega es necesario que ambos cuenten con la licencia, dificultando el trabajo en equipo. Una solución sencilla, gratuita y eficiente es DASH, el cual nos permite crear diferentes gráficas, tablas y demás componentes, totalmente personalizables para que el usuario tenga una mayor flexibilidad a la hora de visualizar datos. DASH solo necesita de 2 componentes, un navegador web y python, por lo que prácticamente todos los dispositivos pueden utilizarlo, desde computadoras o notebooks hasta tablets y celulares.

El principal objetivo de este artículo es servir de apoyo a los futuros becarios y becarias del grupo de materiales granulares que deseen utilizar DASH para apoyarse en sus experimentos. Pudiendo también servir como guía para todo aquel que esté dando sus primeros pasos en DASH y quiera profundizar sus conocimientos.

1. Introducción

En la actualidad, los sensores pueden extraer de un ensayo un sin fin de mediciones con errores relativamente despreciables, esto le permite al investigador poder contar con una fuente muy rica de información en un lapso corto de tiempo. Si a esto le sumamos las herramientas que nos ofrece la informática podríamos almacenar miles o incluso millones de mediciones en un archivo compacto y explícito, en donde la información esté organizada de forma tal que un programa pueda seleccionar rápidamente los fragmentos de datos que se necesiten. A este archivo, generalmente de extensión .Dat o .Csv, lo denominaremos **Base de datos** o también conocido en inglés como **Database**.

Al igual que en la filosofía una pregunta desencadena decenas de nuevas preguntas, una herramienta que busca satisfacer una necesidad genera decenas de nuevas necesidades. Con la posibilidad de crear bases de datos extensas surgieron nuevos desafíos para el investigador, el principal o más evidente es poder seleccionar, ordenar y graficar datos. En este artículo trabajaremos con DASH, el cual es un entorno de trabajo o **Framework** de python especializado en la construcción de aplicaciones analíticas o de visualización de datos.

2. Explorando DASH

2.1. ¿Qué es DASH?

En palabras simples, a través de ciertos comandos DASH nos permite crear nuestras propias aplicaciones y ejecutarlas en un navegador web, además de tener la posibilidad de interactuar con las mismas y poder personalizar cada uno de sus componentes a nuestro gusto. Al basarse en el lenguaje de programación de PYTHON es posible utilizarlo en múltiples plataformas, su simpleza y versatilidad sumadas a que no es necesario contar con una licencia para poder utilizarlo, lo convierten en una gran herramienta para investigadores y estudiantes. Vale agregar que al ejecutarse en un navegador web, DASH nos permite compartir nuestras aplicaciones con cualquier persona que cuente con acceso a internet

Todas estas características nos permiten imaginar su aplicación, por ejemplo, en una página meteorológica donde se muestren gráficas y paneles de humedad, viento y probabilidades de lluvia. También nos podríamos imaginar un reporte financiero del valor de las acciones de una empresa, el control periódico de una granja hidropónica o cualquier cultivo indoor. En cualquier área en la que surja la necesidad de poder visualizar datos, ya sean estáticos o dinámicos, DASH toma ventaja con respecto a muchos softwares existentes.

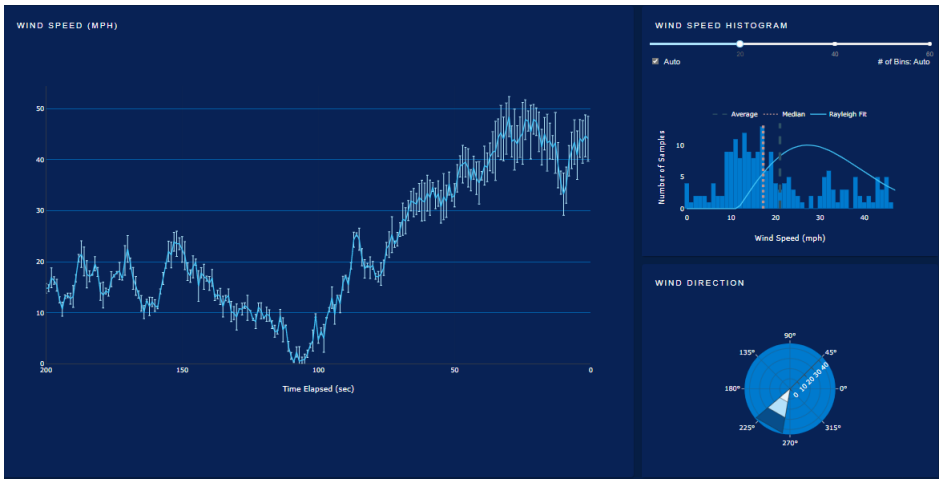

Figura 1: Aplicación para la retransmisión o *streaming* del viento

2.2. Instalación

Una vez instalado python (el cual se descarga de su página oficial www.python.org), instalaremos las siguientes librerías:

- Dash
- Dash daq
- Plotly
- Pandas

Dash es la librería principal la cual nos da las bases para poder crear aplicaciones simples, complementada con Dash daq dispondremos de una mayor cantidad de componentes como medidores de aguja, pulsadores lógicos y diversos tipos de gráficas que aporta Plotly. La librería Pandas la utilizaremos para trabajar con la base de datos de extensión .dat o .csv.

Para instalarlas abriremos la terminal del sistema (Símbolo del sistema en windows 10) y pondremos los siguientes comandos:

- pip install dash
- pip install dash_daq
- pip install plotly
- pip install pandas

3. ¿Cómo está constituida una aplicación de DASH?

Una aplicación consta de dos partes bien diferenciadas, el diseño o *layout* y la parte interactiva. La primera está formada por componentes visuales como marcadores, gráficas, pulsadores, títulos y textos; es la parte principal e indispensable del proyecto, en esta especificamos que elementos van a formar nuestra aplicación, que posición van a ocupar, modificamos funciones internas de cada elemento como colores, títulos y escalas. Cada elemento tiene la posibilidad de interactuar con los demás, por ejemplo, podríamos crear una aplicación que esté formada por un marcador aguja que represente los datos de un sensor de temperatura y una luz, programados de tal manera que cuando el marcador llegue a 30°C se encienda la luz.

Cuando un elemento envía una señal a otro, se dice que lo está llamando o se está efectuando un *Callback*, esta es una de las mejores herramientas que nos ofrece DASH, ya que una vez dominada nos permite realizar aplicaciones bastante complejas en pocas líneas.

3.1. Layout

Describe como se ve la aplicación, Dash provee una serie de componentes visuales en 2 librerías diferentes:

- Elementos HTML: Son componentes simples como títulos y textos, mayormente son estáticos, es decir que no interactúan con los demás elementos. Para poder importarlos normalmente se utiliza el comando *import dash_html_components as html*.
- Elementos Core: Son componentes complejos, algunos están construidos con JavaScript y React.js. Pueden ser gráficas, pulsadores, marcadores, en su mayoría permiten una interacción con demás elementos. Se los importa con el comando *import dash_core_components as dcc*.

Existen diferentes librerías que aportan nuevos elementos como Dash.daq, la cual utilizaremos. Además si poseemos conocimientos en JavaScript y React.js, Dash nos permite crear nuestros propios componentes.

Cuando especificamos el layout de nuestro proyecto iniciamos la primera línea con el comando *app.layout = html.Div(...)*, al ejecutar la aplicación veremos la página en blanco ya que el elemento div no tiene otra función que la de contener otros elementos. Para añadir un componente lo incluiremos dentro del mismo de la siguiente manera:

app.layout = html.Div(html.H1('laboratorio GMG 2020'))

Si quisieramos añadir más componentes simplemente separamos con comas dentro del elemento Div.

3.2. Callbacks

Cuando queremos vincular algunas partes de nuestra aplicación para tener cierto sincronismo que nos permita realizar nuestro experimento, o como sistema de seguridad para detenerlo, Dash nos ofrece una herramienta simple y eficiente para poder resolverlo, los callbacks. Con ellos podemos tomar un dato de entrada, como el valor de un termómetro por ejemplo, y cuando ocurre una determinada acción este dato es enviado al elemento de salida, que puede ser un cuadro, gráfica o el valor de otro componente. Cuando trabajemos con elementos de entrada y salida es necesario importar desde las dependencias de dash el paquete correspondiente, esto se hace con la siguiente línea:

from dash.dependencies import Output, Input

Para crear un callback tenemos que declarar de manera explícita que componentes van a funcionar como entrada y salida, que parámetros de los mismos queremos enviar, y que variación o cambio va a ser el detonante para que se ejecute la llamada. En el desarrollo del proyecto veremos con mayor detenimiento esta sección.

4. Panel de visualización

Dividiremos al proyecto en 4 partes, cada una con su respectiva sección donde se desarrollará su funcionamiento para que futuros becarios y becarias puedan modificar el programa según surjan nuevas necesidades o aplicaciones:

- Aplicación base
- Elementos HTML
- Gráfica
- Medidor de humedad y temperatura

Para el desarrollo de este informe utilizaremos como base de datos un archivo de texto plano compuesto por 4 columnas separadas por espacios, que representan magnitudes medidas por sensores que controla una placa Arduino, provenientes de un ensayo de descarga de material granular de un silo.

	tiempo	temperatura	humedad	masa
2	0.12	18	59	-0.07
3	0.23	18	59	0.02
4	0.33	18	59	-0.22
5	0.43	18	59	-0.01

Figura 2: Primeras columnas del archivo de texto

4.1. Aplicación base

Toda aplicación creada con Dash tiene en común una cierta cantidad de líneas a las que llamaremos base, las cuales contienen los requisitos mínimos para poder ejecutar el elemento contenedor Div. Comenzamos por importar Dash y los elementos html y core como se mostró en secciones anteriores, luego escribiremos lo siguiente:

```
app = dash.Dash(__name__)
app.layout = html.Div()
if __name__ == '__main__' :
    app.run_server(debug = True)
```

Al ejecutar el archivo .py en la consola veremos un mensaje como el siguiente:
Dash is running on http://127.0.0.1:8050/

Si abrimos el enlace en el navegador web veremos la aplicación base ya en ejecución, la misma estará en blanco ya que todavía no incorporamos ningún elemento. En la esquina inferior derecha veremos un menú desplegable el cual contiene 3 indicadores, el de la derecha señala que el servidor se está ejecutando correctamente, el del medio informa con detalles los errores que sucedan, y el izquierdo muestra un árbol de las conexiones realizadas con callbacks, que al no contar con ninguno, por el momento no se puede desplegar.

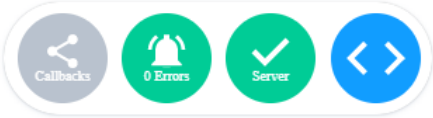

Figura 3: Menú desplegable

4.2. Elementos HTML

Comenzaremos por agregar los títulos del programa, como ya contamos con la librería con los elementos html importada solo tenemos que incluirlos dentro del elemento Div. Además modificaremos algunas funciones internas para centrar su alineación respecto a la página:

```
app.layout = html.Div(
    html.H1('Laboratorio GMG 2020',
            style={'textAlign':'center'}),
    html.H2('Panel de visualización de sensores',
            style={'textAlign':'center'}))
```

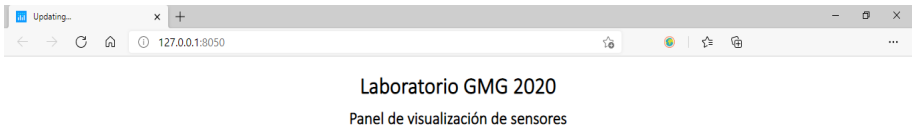

Figura 4: Captura de la aplicación con los elementos html agregados

Dash nos ofrece una guía de usuario con todas las funciones internas de cada elemento para poder personalizarlos a nuestro gusto, se puede encontrar en la página oficial **dash.plotly.com**

4.3. Gráfica

Construiremos una gráfica en la que se represente el tiempo como abscisa y la columna masa como ordenada. Cabe aclarar que el archivo que vamos a graficar es dinámico, los sensores conectados al simulador van a imprimir decenas de líneas segundo a segundo, por lo tanto hay que releer el archivo constantemente y enviar esta información a la gráfica. En resumen, debemos enviar mediante un callback la lectura de la base de datos a la gráfica, existe un elemento que nos puede facilitar esta tarea, que a primera vista se ve compleja.

El elemento **Interval**, incluido en la librería Core, funciona como detonante para realizar un callback cada un intervalo de tiempo programado. Es decir, que cada cierto lapso de tiempo, se releerá la base de datos y se actualizará la gráfica. Como todo componente, se incluye dentro del html.Div, utilizaremos el siguiente código:

```
dcc.Interval(
    id='detonador',
    interval=200,
    n_intervals = -1)
```

Le asignamos el id 'detonador' para poder identificarlo cuando realicemos el callback, todos los elementos tienen que tener id único para que el programa no presente fallas. El parámetro interval es el tiempo de espera, en milisegundos, para ejecutar nuevamente el callback de relectura. Cuando el dcc.Interval está en ejecución, internamente funciona como un contador, es decir, al ejecutar la aplicación inicia con un n = 0, y al cumplirse los ciclos de espera el valor de n se va incrementando de 1 en 1. La función n_intervals define el valor máximo que puede tomar n hasta que se detiene, es decir, el número máximo de actualizaciones que recibirá la gráfica, en nuestro caso utilizaremos el valor -1 que representa infinitas actualizaciones.

Una gráfica cuenta con 2 parámetros, id y figure, en la función figure se definen los datos que representarán el eje x e y, así como el *layout* o diseño de la gráfica. Como la información de la gráfica la tenemos que enviar mediante un callback, dentro del html.Div solo incluiremos el id. Es importante haber importado con anterioridad las biblioteca plotly, como se muestra a continuación:

```
import plotly
import plotly.graph_objs as go


app.layout = html.Div(
                    .....
                     dcc.Graph(id = 'graf 1')
```

Para construir el callback definiremos que componente funcionará como entrada y cual como salida. El callback se ejecutará cada vez que cambie el valor de la función n_intervals y el dato que se envíe va a tener como destino la función *figure* de la gráfica, por lo tanto nuestro callback queda de la siguiente manera:

```
    @app.callback(Output('graf 1','figure'),
                [Input('detonador',n_intervals)])
```

a continuación debemos especificar que dato es el que enviaremos. El tiempo, ubicado en la primera columna de la base de datos, será el eje de abscisas y la masa, ubicada en la cuarta columna, será el eje y. Para leer la base de datos con la librería pandas usaremos el comando *pd.read_csv*, y dentro del mismo pondremos, el nombre del archivo, el delimitador entre columnas (en nuestra base de datos se utilizó el espacio) y el formato en el que está escrita. Debajo crearemos 2 funciones que representarán los datos ubicados en la primera y cuarta columna:

```
def actualización(n):
    df = pd.read_csv('datos.dat',delimiter = ' ',
                     encoding = 'utf_16')
    X = df['tiempo']
    Y = df['masa']
```

Ya leída la base de datos definiremos las secciónes *data* y *layout* para poder enviarlos a la gráfica:

```
    data = go.Scatter(
            x = list(X),
            y = list(Y),
            mode = 'lines')
```

```
return { 'data': [data],
         'layout':go.Layout(
             xaxis={'title':'Tiempo'},
             yaxis={'title':'Masa'})}
```

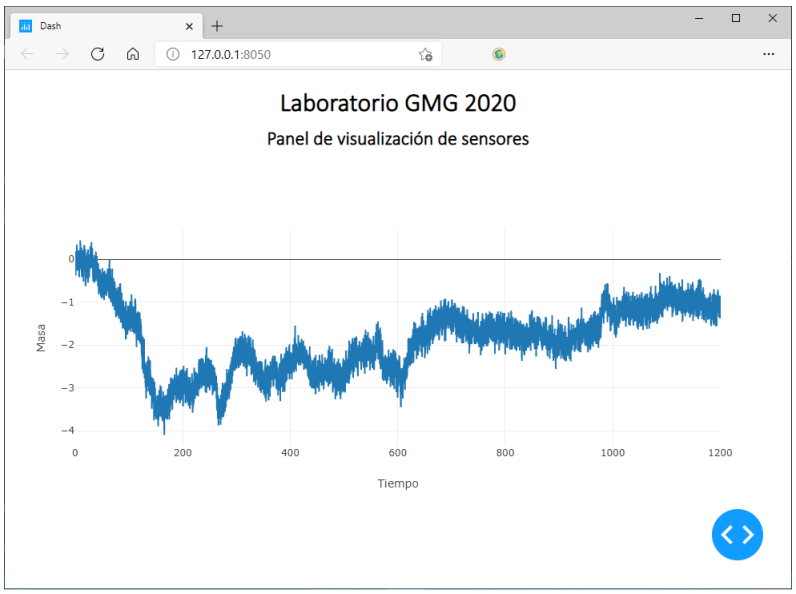

Figura 5: Gráfica de masa vs tiempo

4.4. Medidor de humedad y temperatura

La temperatura y la humedad a lo largo del experimento no presentan grandes cambios por lo que se consideró más apropiado utilizar marcadores que representen el último valor medido por el sensor. Se desarrollará solamente el medidor de humedad ya que no hay gran diferencia en la construcción de ambos.

Para visualizar la lectura de la humedad utilizaremos un componente estéticamente similar a un display o pantalla led. Para añadirlo al div se necesitan especificar 2 parámetros, el ID y el valor numérico que el componente va a marcar; este valor lo obtendremos de leer el último dato escrito por el sensor en el archivo, y lo enviaremos mediante un callback, por lo que dentro del layout solo especificaremos el ID:

```
app.layout = html.Div(
                    .......
                     daq.LEDDisplay(id = 'medidorhumedad'))
```

En el callback utilizaremos, igual que para la gráfica, el componente *interval* para actualizar la lectura. La única diferencia se va a encontrar en la variable; en la gráfica, X e Y representaban los datos de toda la columna, ahora solo leeremos el último dato de la columna humedad y lo enviaremos a la función *value* del display.

```
@app.callback(Output('medidorhumedad','value'),
              [Input('detonador','n_intervals')])

def marc_humedad(n):
    df = pd.read_csv('datos.dat',delimiter = ' ',)
    H = df.iloc[-1,2]

    return int(H)
```

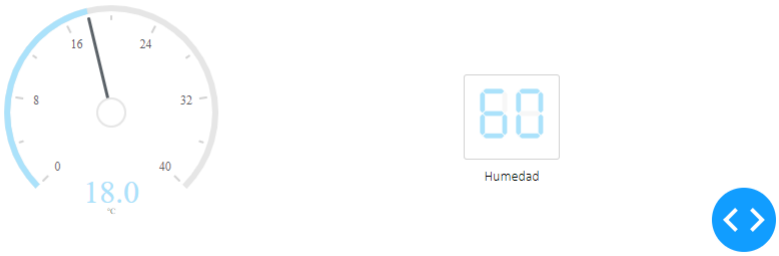

Figura 6: Marcadores de temperatura y humedad