

Taller 4 de Analítica Tableros en EC2

Isabela Castillo 201813093

Carlos Molano 201922691

Taller 4 - Primeros pasos en la Nube: Tableros en EC2 Profesor: Juan F. Pérez

Pre-requisitos

1. Para esta sesión va a requerir una cuenta de Amazon Web Services - AWS. Para esto hay varias opciones:
 - a) Usar la cuenta de AWS Academy enviada a su correo por el instructor, y acceder al curso **Learner Lab** (recomendado).
 - b) Utilizar una cuenta propia ya creada.
 - c) Crear una cuenta nueva. En este caso recuerde que requiere ingresar datos de una tarjeta de crédito. Usaremos recursos disponibles en la capa gratuita (<https://aws.amazon.com/free/>), pero es posible que se generen algunos costos menores.
2. **Nota:** este taller se debe realizar en parejas. Defina un nombre para su grupo, defínalo claramente en su **reporte** y use este nombre como parte inicial de **todos los recursos que cree en la nube**.
3. **Nota 2:** la entrega de este taller consiste en un **reporte** y unos **archivos de soporte**. Cree el archivo de su **reporte** como un documento de texto en el que pueda fácilmente incorporar capturas de pantalla, textos y similares. Puede ser un archivo de word, libre office, markdown, entre otros.
4. Asegúrese de tener instalado Python en versión 3.11 o superior. Para este taller usaremos Python y en particular las librerías *dash*, *plotly* y *pandas*. Puede instalar las librerías con el comando

```
pip install dash plotly pandas
```

o si prefiere repositorios de anaconda use

```
conda install dash plotly pandas
```

Sección Pre-requisitos:

Literal 2: Nombre del Grupo: Asistentes

1. Tableros locales

1.1. Un primer tablero en Dash

Dash es un framework para el desarrollo de aplicaciones web en python, desarrollado por Plotly y construido sobre Flask. En particular, Dash ofrece facilidades para crear tableros para la visualización de datos interactivos y dinámicos que proveen una buena interfaz al usuario del tablero.

Dash interactúa además con Pandas, la librería estándar para la manipulación de datos en python. Esto permite fácilmente cargar datos desde una fuente externa y generar visualizaciones de gran calidad.

Taller 4 - Primeros pasos en la Nube: Tableros en EC2 Profesor: Juan F. Pérez

Al ser un framework web, el resultado es una página web que puede accederse usando navegadores estándar, ya sea localmente o en un servidor.

Como parte de este taller encontrará algunos ejemplos sencillos de tableros en Dash, que nos sirven como punto de partida.

1. Abra el archivo `app1.py` en un editor de python como Visual Studio code. Para abrir los archivos de base de este taller **evite** usar soluciones orientadas a cuadernos como jupyter.
2. Note que en la parte final del archivo se define qué hacer cuando se ejecuta el archivo. En este caso se llama al objeto `app` y se ejecuta su método `run_server`. El objeto `app` se crea en las primeras líneas del script. En su **reporte** incluya el código con el que se crea el objeto `app`.
3. Después de crear el objeto `app` (y extraer el atributo `server`) el programa define un dataframe de pandas en el que se almacena un conjunto de datos.

4. Ejecute la aplicación usando su editor o ejecutando en consola

```
python app1.py
```

o

```
python3 app1.py
```

dependiendo de su sistema operativo e instalación de python.

5. En su navegador (Chrome, Firefox, Safari, Edge) visite el servidor local usando el puerto 8050, es decir, vaya a la página

`http://127.0.0.1:8050/`

En su **reporte** incluya un pantallazo donde aparezca el código en ejecución y la página cargada, lado a lado.

6. Considerando la aplicación resultante, describa en su **reporte** qué hace el comando

```
fig = px.bar(df, x="Fiebre", y="Casos", color="Diagnostico",
             barmode="group")
```

También puede apoyarse en la documentación de plotly:

- <https://plotly.com/python-api-reference/generated/plotly.express.bar.html>
- <https://plotly.com/python/bar-charts/>

7. La siguiente (y última sección de código) crea el documento html que es renderizado por el navegador. Note que se crea definiendo secciones (Div) y título (H1), entre otros elementos html. En su reporte describa brevemente qué hacen las líneas

```
dcc.Graph(
    id='example-graph',
    figure=fig
),

y

html.Div(
    className="Columnas",
    children=[
        html.Ul(id='my-list', children=[html.Li(i) for i in df.columns
        ])
    ],
)
```

8. Realice cambios en los datos: modifique los nombres de las columnas y los valores que toman. Realice todos los cambios necesarios hasta obtener una nueva versión del tablero con datos de su elección. Incluya su archivo modificado como **soporte** de su entrega. Incluya un pantallazo de la aplicación en ejecución en su **reporte**.

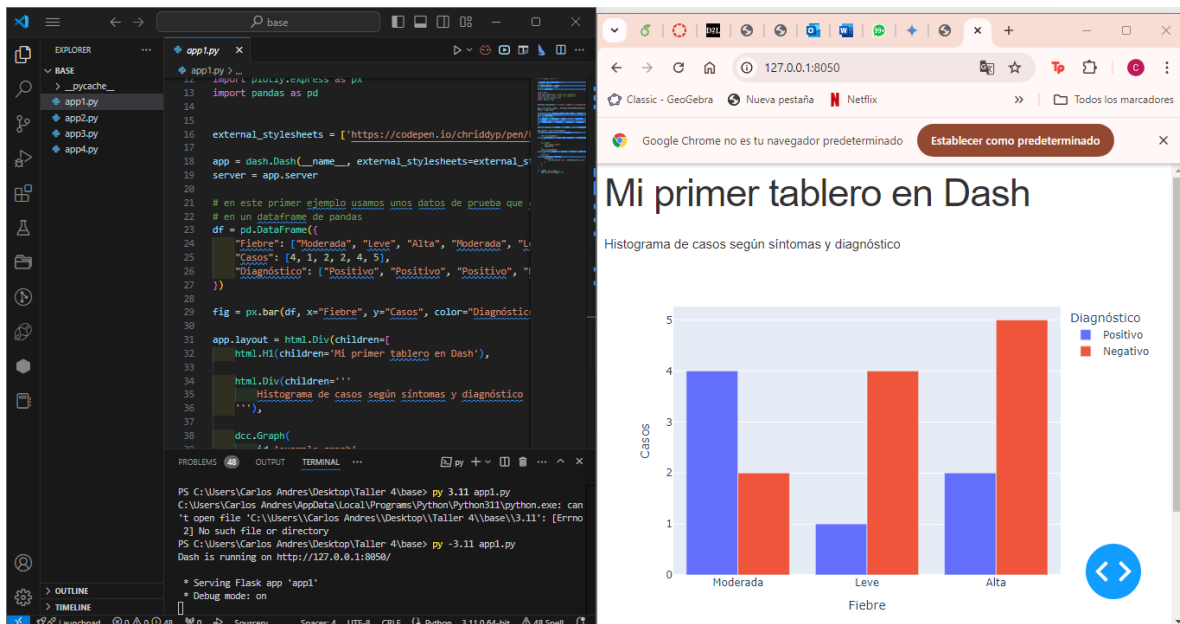
Parte 1: Tableros Locales

Literal 2:

```
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
server = app.server
```

py -3.11 app1.py

Literal 5:



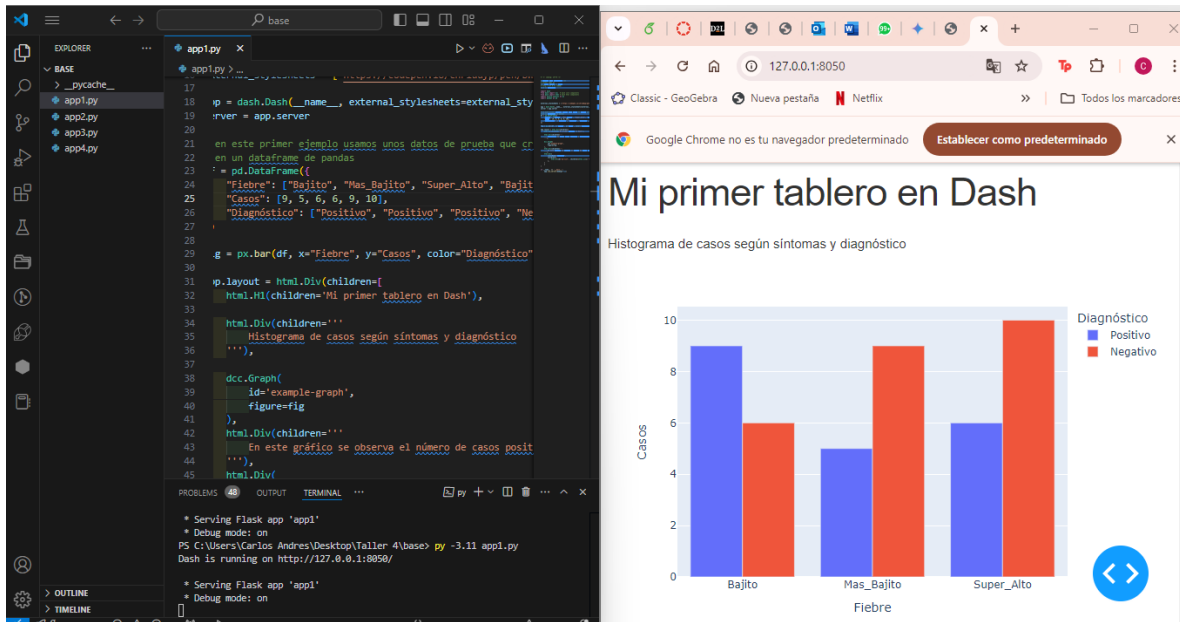
Literal 6:

Este comando utiliza la función `px.bar` de Plotly Express para crear un gráfico de barras. El gráfico muestra la cantidad de casos en el eje y, agrupados según la variable `Diagnostico`, y distribuidos en el eje x de acuerdo con la variable `Fiebre`. El parámetro `barmode="group"` asegura que las barras de diferentes categorías de diagnóstico se coloquen una al lado de la otra para facilitar la comparación directa.

Literal 7:

Crea un layout para la aplicación Dash que incluye un gráfico (`dcc.Graph`) con el identificador `example-graph`, el cual renderiza el gráfico de barras previamente definido en la variable `fig`. Además, define un contenedor `Div` con la clase `Columns`, dentro del cual se genera una lista no ordenada (`html.Ul`) identificada como `my-list`, que contiene los nombres de todas las columnas del `DataFrame df`, representados como elementos de lista (`html.Li`). Esto permite que el gráfico se muestre junto con una lista de los nombres de las columnas del `DataFrame`.

Literal 8:



1.2. Callbacks

Un mecanismo muy útil de Dash es el uso de callbacks para modificar dinámicamente elementos del tablero. Veamos un ejemplo sencillo de callbacks.

1. Abra el archivo `app2.py` en un editor de python.
2. Note que el código tiene muchas similitudes con el anterior, incluyendo el main y la definición de los objetos `app` y `server`.
3. La gran diferencia de esta nueva aplicación radica en la función

```
def update_output_div(input_value):  
    return 'Output: {}'.format(input_value)
```

Note que a esta función se le agrega un *decorador*, el cual extiende la función asociando sus entradas y salidas con elementos del documento HTML

```
@app.callback(  
    Output(component_id='my-output', component_property='children'),  
    [Input(component_id='my-input', component_property='value')]  
)
```

4. En su **reporte** describa qué hace la función `update_output_div`, considerando el decorador y qué elementos el documento HTML asocia con la función.
5. Modifique la frase que retorna la función, por ejemplo incluyendo un texto personalizado, además del texto que ingresa el usuario. Incluya el código modificado como **soporte** de su entrega. Incluya un pantallazo de la aplicación modificada en ejecución en su **reporte**.

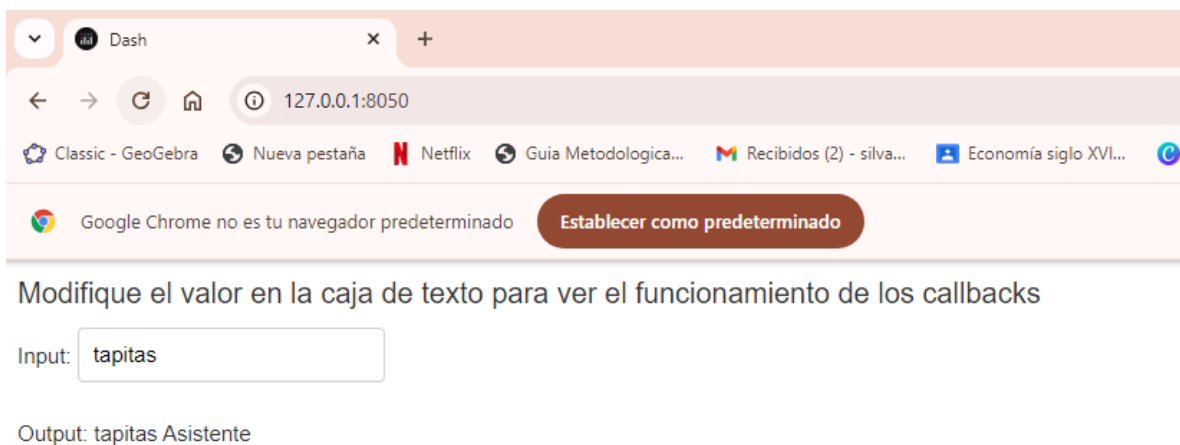
Parte 1.2: Callbacks

Literal 4:

La función `update_output_div` es un callback de Dash que, utilizando el decorador `@app.callback`, actualiza el contenido de un elemento `Div` en tiempo real basado en el valor ingresado en un campo de texto (`dcc.Input`). Cada vez que el usuario modifica el valor en el campo de texto, la función toma este valor como entrada y actualiza el `Div` identificado por `my-output` con una cadena que incluye el texto "Output:" seguido del valor ingresado, mostrando así el resultado de manera dinámica en la aplicación. En el `Html` se asocia a el valor de output

Output: valor inicial Asistente

Literal 5:



1.3. Más visualizaciones e interacciones

Hasta el momento hemos visto un tipo de visualización muy sencilla. Ahora consideremos una visualización más elaborada.

1. Abra el archivo `app3.py` en un editor de python.
2. Note que el código tiene muchas similitudes con el anterior, no solo en el `main`, `app` y `server`, sino también en la definición de `layout` y una función con `callback`.
3. Ejecute la aplicación.
4. Describa en su **reporte** el `layout` de esta aplicación. ¿Qué elementos tiene?
5. En su **reporte** describa qué hace la función `update_figure` y su decorador, considerando los elementos del documento HTML asociados.
6. Ahora realice *una nueva visualización* de interés. Considere las opciones de gráficas que ofrece `plotly.express` <https://plotly.com/python/plotly-express/>. También puede ser de interés la Coda a esta sección. Incluya títulos y textos que describan las visualizaciones incluidas.
7. Incluya el código modificado como **soporte** de su entrega. Incluya un pantallazo de la aplicación modificada en ejecución en su **reporte**.
8. A manera de ejemplo adicional incluimos el archivo `app4.py` donde podrá ver elementos adicionales y una actualización un poco más elaborada de una gráfica y sus elementos usando `callbacks`.
9. Para terminar esta sección descomprima el archivo `data-food-consumption.zip` que encontrará en Bloque Neón. Allí encontrará una app tomada del repositorio <https://github.com/plotly/dash-sample-apps>, y en particular de la sección de releases <https://github.com/plotly/dash-sample-apps/releases>. Esta aplicación es específicamente <https://github.com/plotly/dash-sample-apps/releases/download/v0.23.5/dash-food-consumption.zip> con algunas actualizaciones para ejecutar sin errores/warnings en versiones recientes de `dash/pandas/statsmodels`. Explore la estructura de carpetas y los archivos. Ejecute la aplicación y explore el tablero resultante. En su **reporte** explique *cada uno* de los `callbacks` que tiene la aplicación, específicamente cuál es su `input`, su ejecución y su `output`.

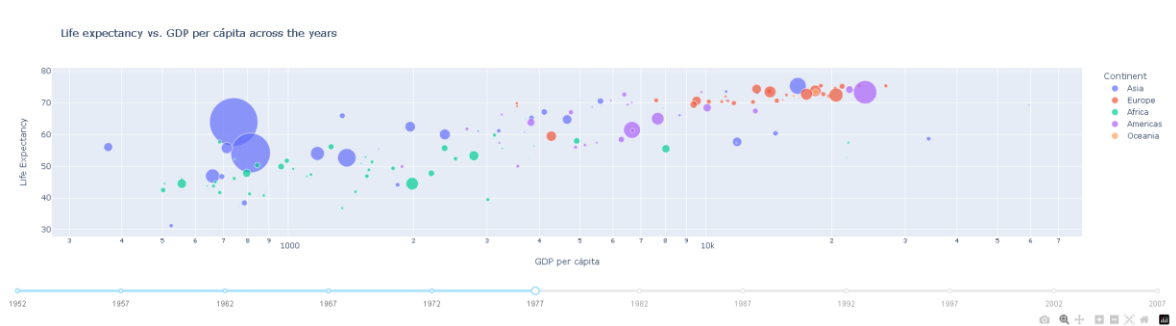
Parte 1.3: Más Visualizaciones e interacciones

Literal 4:

El `layout` de la aplicación está organizado dentro de `app.layout` utilizando componentes de Dash. Este incluye un contenedor general (`html.Div([])`) que agrupa todos los componentes, un gráfico interactivo (`dcc.Graph(id='graph-with-slider')`) que se actualiza según el año seleccionado en un deslizador (`dcc.Slider(id='year-slider', ...)`), y un gráfico de líneas predefinido (`dcc.Graph(id='graph2', figure=fig2)`) que muestra la evolución de la esperanza de vida promedio por continente. En resumen, el `layout` consiste en un contenedor principal que agrupa un gráfico interactivo, un deslizador para seleccionar años, y un gráfico de líneas.

Literal 5:

La función `update_figure` es un callback en Dash que, decorado con `@app.callback`, actualiza dinámicamente el gráfico interactivo `graph-with-slider` en respuesta al año seleccionado en el deslizador `year-slider`. Al recibir el año seleccionado, la función filtra los datos correspondientes y genera un gráfico de dispersión que muestra la relación entre el PIB per cápita y la esperanza de vida, con el tamaño de los puntos indicando la población y el color representando el continente. Este gráfico actualizado se muestra en el componente `dcc.Graph`, permitiendo a los usuarios explorar cómo varía esta relación a lo largo del tiempo.

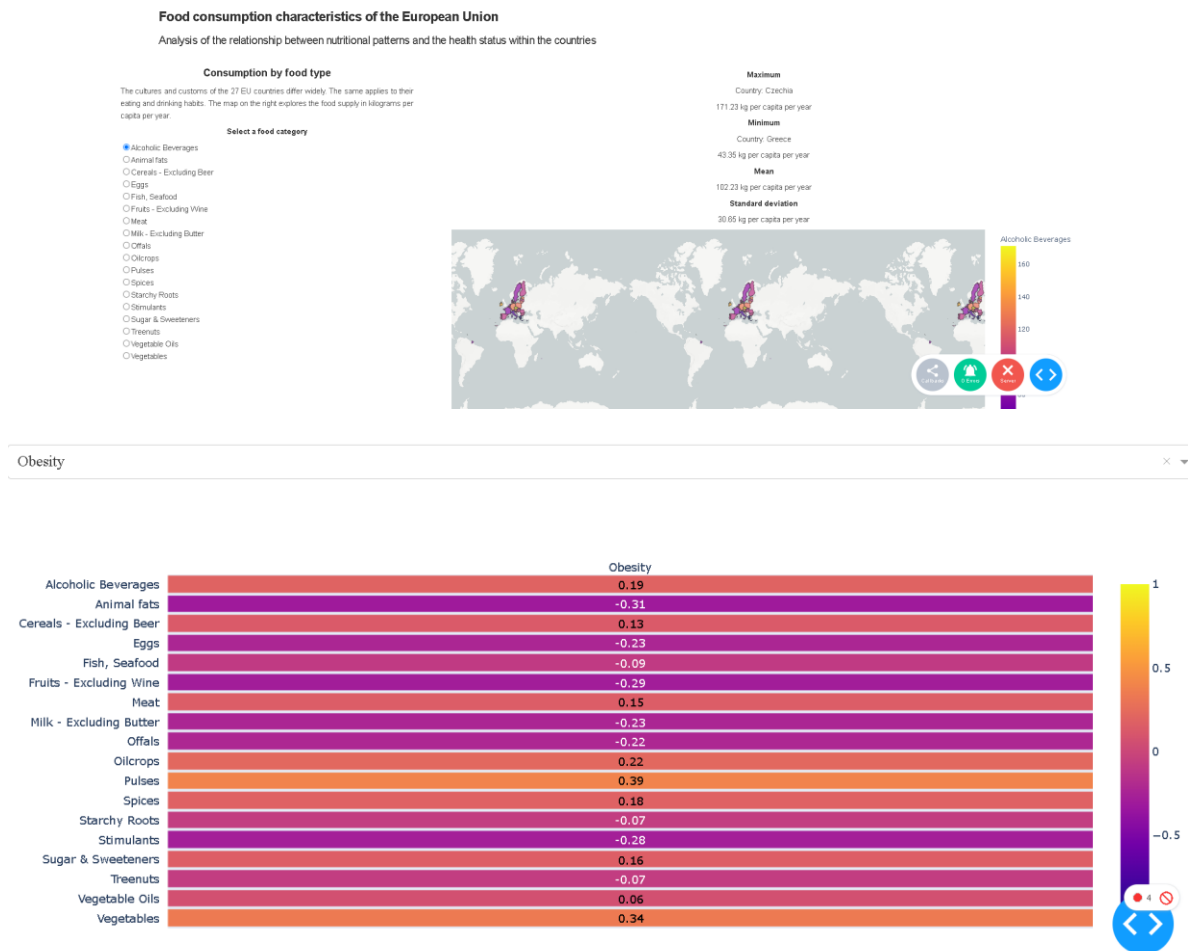


En el Html, se ve cuando se utiliza el slider dependiendo el año se actualiza la gráfica como se observa en la imagen de arriba

Literal 7:



Literal 9:



Dentro de la app, se encuentran 6 callbacks:

1. Callback ``display_choropleth_map``:

- Output:
 - `"choropleth", "figure"`
- Input:
 - ``"nutrition_types", "value"``

Ejecución:

Cuando el usuario selecciona un valor diferente en el componente ``"nutrition_types"`` (por ejemplo, un tipo de alimento), este callback se activa.

Se genera un nuevo mapa coroplético utilizando ``plotly.express.choropleth_mapbox``. El mapa muestra los países de la Unión Europea coloreados según el consumo del alimento seleccionado. La visualización se centra en Europa, con un zoom de 2.5, y utiliza un mapa de fondo de Mapbox.

- *Descripción:* Actualiza la figura del mapa coroplético (choropleth) mostrando el consumo de alimentos por país en función de la categoría seleccionada por el usuario.

2. Callback `display_box_plot`:

- Output:
 - `"boxes", "figure"`
- Inputs:
 - `"clust_types", "value"`
 - `"box_dd", "value"`
- Ejecución: Este callback se activa cuando el usuario selecciona un tipo de clúster y una variable en los componentes `"clust_types"` y `"box_dd"`. Lee los datos del archivo `box_cluster.csv`, ajusta los valores de clústeres, y ordena los datos según el clúster seleccionado. Luego, genera un gráfico de cajas (`box plot`) usando `plotly.express.box`, con colores asignados a cada clúster. El gráfico resultante permite visualizar la distribución de la variable seleccionada en función de los diferentes clústeres.

Descripción: Genera un gráfico de cajas para visualizar la distribución de una variable seleccionada en función de diferentes clústeres de alimentos o salud.

3. Callback `display_cluster_distribution_map`:

- Output:
 - `"cluster_map", "figure"`
- Input:
 - `"clust_types", "value"`

Ejecución: Se activa cuando el usuario selecciona un tipo de clúster en el componente `"clust_types"`.

Ordena el DataFrame `clusters` según el tipo de clúster seleccionado y genera un mapa coroplético utilizando `plotly.express.choropleth_mapbox`. El mapa muestra cómo se distribuyen los clústeres seleccionados en los países de la Unión Europea.

El mapa se centra en Europa, y se colorea según los valores del clúster, permitiendo una visualización clara de las agrupaciones geográficas.

Descripción: Muestra un mapa que representa la distribución de clústeres (alimentarios, de salud o combinados) en los países de la Unión Europea, basado en la selección del usuario.

4. Callback `display_correlation_heatmap`:

- Output:
 - `"cor_ma", "figure"`
- Input:
 - `"cor_behave", "value"`

Ejecución: Se activa cuando el usuario selecciona una variable en el componente `"cor_behave".

Filtra los datos del DataFrame `df_scatter` para incluir solo las columnas relevantes para calcular la correlación. Luego, se calcula la matriz de correlación para la variable seleccionada y se redondea a dos decimales.

Crea un heatmap (mapa de calor) con `plotly.figure_factory.create_annotated_heatmap`, visualizando las correlaciones entre los diferentes alimentos y variables de salud seleccionadas.

Descripción: Crea un mapa de calor que muestra las correlaciones entre diferentes tipos de alimentos y variables de salud seleccionadas, destacando las relaciones más fuertes.

5. Callback `display_statistical_indicators`:

- Outputs:
 - `"max_name", "children"`
 - `"max_value", "children"`
 - `"min_name", "children"`
 - `"min_value", "children"`
 - `"mean", "children"`
 - `"st_dev", "children"`
- Input:

- ` "nutrition_types", "value" `

- **Ejecución:**

Se activa cuando el usuario selecciona un tipo de alimento en el componente ` "nutrition_types" `.

La función calcula el país con el consumo máximo y mínimo del alimento seleccionado, así como sus valores correspondientes. También calcula la media y la desviación estándar del consumo.

Estos valores se formatean y se muestran en la interfaz, proporcionando una visión rápida de los indicadores estadísticos clave para el alimento seleccionado.

Descripción: Calcula y muestra indicadores estadísticos clave (máximo, mínimo, media y desviación estándar) para el consumo de un tipo de alimento seleccionado.

6. Callback `update_scatter_plot_with_regression` :

- Output:

- ` "indicator-graphic", "figure" `

- Inputs:

- ` "xaxis-column", "value" `

- ` "yaxis-column", "value" `

- ` "xaxis-type", "value" `

- ` "yaxis-type", "value" `

Ejecución: Se activa cuando el usuario selecciona las columnas para los ejes x e y, y los tipos de ejes (lineal o logarítmico) en los componentes correspondientes.

Crea una nueva columna en ` df_scatter ` para categorizar si los valores de la variable en el eje y están por encima o por debajo de la media, y usa esta categorización para colorear los puntos en el gráfico.

Genera un gráfico de dispersión con ` plotly.express.scatter `, permitiendo la visualización de la relación entre las variables seleccionadas. También se incluye una línea de regresión lineal calculada con ` statsmodels ` para mostrar la tendencia general.

Descripción: Actualiza una gráfica de dispersión interactiva que permite explorar la relación entre dos variables seleccionadas, incluyendo distribuciones marginales y una línea de regresión.

2. Primeros pasos en AWS EC2: lanzar una máquina virtual

1. En esta sección desplegaremos una máquina virtual Linux usando el servicio EC2 de AWS, el cual es de tipo IaaS. Usaremos esta máquina para servir un tablero Dash.
2. En AWS Academy encontrará el curso Learner Lab. Ingrese al curso y en la sección de Contenidos seleccione el módulo Learner Lab.

3. Para empezar de click en el botón AWS en la parte superior del laboratorio, lo que le permitirá ingresar a la consola de AWS.
4. En la parte superior de la consola de AWS hay un filtro de servicios, busque allí EC2, el servicio de máquinas virtuales de AWS.
5. En la consola de EC2, en el panel izquierdo seleccione *Instancias* y click en *Lanzar instancias*. Lance una instancia con las siguientes características:
 - a) Nombre: asigne un nombre adecuado.
 - b) Imagen (Amazon Machine Image - AMI): Amazon Linux (note que hay muchas opciones).
 - c) Tipo de instancia: t2.micro (apta para la capa gratuita).
 - d) Par de claves: Cree un nuevo par de claves
 - 1) Asigne un nombre adecuado.
 - 2) Seleccione RSA como tipo y .pem como formato de archivo.
 - 3) Asegúrese de guardar la llave .pem localmente en un sitio de fácil acceso (idealmente en una carpeta creada para desarrollar este taller). En adelante lo llamaremos llave.pem.
 - e) En la configuración de red deje los valores por defecto (esto creará un grupo de seguridad con permisos de conexión por SSH, puerto 22).
 - f) Deje la configuración de almacenamiento (8 GB de disco) y los detalles avanzados por defecto.
 - g) Click en lanzar instancia.
 - h) Regrese a la consola de EC2 y en el ítem Instancias debe poder ver la instancia en proceso de inicio.
 - i) En su **reporte** tome un snapshot del tablero EC2 que muestre su instancia en ejecución.
 - j) Note que el campo Comprobación indica que la máquina está en proceso de inicialización, luego realiza dos chequeos y luego ya aparece como lista para usar.
6. En la consola de EC2 seleccione su instancia y copie algunos datos en su **reporte**:
 - a) Dirección IP v4 pública.
 - b) Dirección IP v4 privada.
 - c) Tipo de instancia.
 - d) Plataforma y Detalles de la plataforma.
 - e) Tipo de virtualización.
 - f) Número de CPU virtuales.
 - g) Zona de disponibilidad (lo encuentra en la pestaña Redes).

- h) ID de volumen (lo encuentra en la pestaña Almacenamiento).
 - i) Tome un pantallazo de la pestaña Monitoreo.
7. En la consola de EC2 seleccione su instancia y copie la dirección IP (v4) pública.
8. **Conéctese a la instancia:**
- a) Abra una terminal: tecla windows, escriba `cmd` y ENTER)
 - b) En la terminal emita el comando


```
ssh -i /path/to/llave.pem ec2-user@IP
```

donde `/path/to/` se refiere a la ubicación del archivo `llave.pem` que descargó, e IP es la dirección IPv4 pública de la instancia EC2 que lanzó. Si prefiere, en la terminal puede navegar a la ubicación del archivo `llave.pem` y emitir el comando

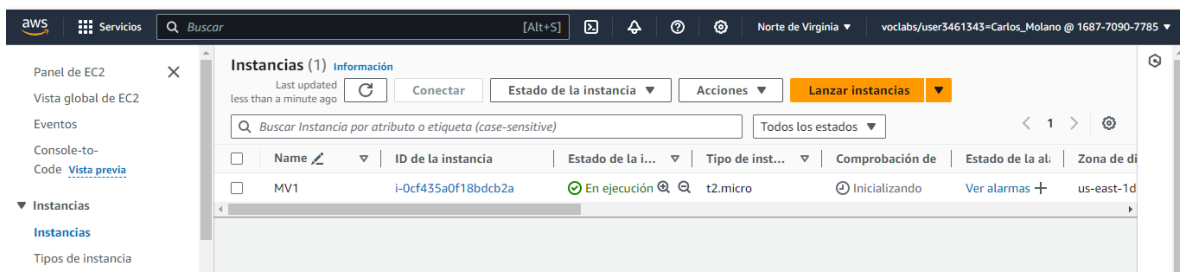
```
ssh -i llave.pem ec2-user@IP
```
 - c) A la pregunta


```
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

 Responda **yes** y ENTER.
 - d) En este momento debe estar **en la terminal de la instancia EC2** que creó. Es decir, los comandos que emita en esta terminal se ejecutan en su máquina virtual (servidor en la nube).
 - e) Tome un pantallazo de la terminal e inclúyalo en su **reporte**.

Parte 2: AWS

Literal 5i:



Literal 6:

A y B)

Dirección IPv4 pública

3.83.25.79 | [dirección abierta](#)


Direcciones IPv4 privadas

172.31.94.24

C)


Tipo de instancia
t2.micro

D)

Plataforma
 Amazon Linux (inferido)


Detalles de la plataforma
 Linux/UNIX

E y F)

Tipo de virtualización
 hvm

Número de CPU virtuales
1

G)

Zona de disponibilidad
 us-east-1d


H)



I)

i-0cf435a0f18bdc2a (MV1)

Detalles Estado y alarmas **Monitoreo** Seguridad Redes Almacenamiento Etiquetas

 **Métricas del agente de CloudWatch**

En la pestaña de supervisión ahora se incluirán métricas relacionadas con una sola instancia en el espacio de nombres de CWAgent. Si desea que se muestren las métricas emitidas del agente de CloudWatch, inclúyalas en el espacio de nombres de CWAgent.

☒ Incluir métricas en el espacio de nombres de CWAgent [Más información](#)

[Configurar el agente de CloudWatch](#) [Administrar el monitoreo detallado](#)

3. Configurar la instancia y lanzar el tablero

Los siguientes comandos se deben ejecutar en la terminal conectada a instancia en la nube, a menos que se indique algo diferente.

1. Actualice el sistema con el siguiente comando

```
sudo yum update -y
```

`sudo` emite comandos como administrador del sistema, tenga mucho cuidado al usarlo. `yum` es el administrador de paquetes/programas de varias distribuciones de Linux, entre ellas las de Amazon. La bandera `-y` genera la respuesta **yes** a todas las confirmaciones que surjan con el comando de actualización.

2. Verifique que tiene instalado Python 3 (al menos versión 3.7) con el comando

```
python3 --version
```

3. Instale pip3 para administrar paquetes de python

```
sudo yum install python3-pip
```

Verifique la versión de pip3

```
pip3 --version
```

4. Instale pandas con pip3

```
pip3 install pandas
```

5. Instale el paquete dash

```
pip3 install dash
```

6. Instale el paquete gunicorn para python

```
pip3 install gunicorn
```

7. Como parte de este taller encontrará el archivo `app1.py`. Queremos subir este archivo a la máquina en la nube. **En su máquina local, abra otra terminal** y navegue al sitio donde tiene este archivo (en adelante supongo que este archivo está en la misma carpeta que `llave.pem`). Para copiar ese archivo a la máquina en la nube, use el comando `scp` así

```
scp -i llave.pem app1.py ec2-user@IP:/home/ec2-user
```

que copia el archivo a la carpeta `home` del usuario `ec2-user` en su máquina virtual.

8. Verifique que se copió el archivo en su máquina virtual listando los contenidos de la carpeta `/home/ec2-user` con el comando

```
ls
```

Incluya un pantallazo de la terminal mostrando el archivo en su **reporte**.

9. En la máquina virtual cree una copia del archivo `app1.py` con nombre `app.py`

```
cp app1.py app.py
```

Verifique la creación del archivo con el comando `ls`. Si quiere más detalles puede usar la bandera `-la`

```
ls -la
```

10. Necesitamos realizar una pequeña modificación sobre el archivo `app.py`, para lo cual usaremos el comando `nano`. En su máquina virtual use el comando

```
nano app.py
```

Esto abre un editor de texto. Usando las **flechas** navegue hasta la parte inferior del archivo y modifique la línea

```
app.run_server(debug=True)
```

para incluir un argumento adicional y que quede

```
app.run_server(host = "0.0.0.0", debug=True)
```

Para cerrar el archivo **CTRL+X**, escriba **Y** para confirmar que quiere guardar los cambios, y **ENTER** para regresar a la terminal principal. Para verificar el cambio puede usar el comando

```
cat app.py
```

que le permite observar el archivo rápidamente, sin acceder a modificarlo. Incluya un pantallazo de la terminal mostrando la salida de este comando en su **reporte**.

11. En la consola de EC2, panel izquierdo, seleccione Security Groups en redes y seguridad. Allí encontrará su grupo de seguridad, selecciónelo.
12. En la parte inferior verá las reglas de entrada, que definen cómo puede entrar tráfico a la instancia. Click en Editar reglas de entrada. Note que solo tiene una regla, que permite tráfico por el puerto 22, el cual usamos para conectarnos a la terminal por SSH.
13. Click en Agregar regla. En Tipo seleccione TCP personalizado, en Intervalo de puertos marque 8050, en Origin seleccione Anywhere IPv4. Click en Guardar regla. Incluya un pantallazo del grupo de seguridad modificado en su **reporte**.
14. Ya estamos listos para lanzar el tablero en el servidor. En la máquina virtual corra la aplicación con
- ```
python3 app.py
```
- En su navegador verifique que la aplicación está corriendo y disponible visitando la dirección
- ```
http://18:8050
```
- Incluya un pantallazo de la aplicación ejecutándose en su **reporte**. Por el canal de **Slack** incluya la URL completa (enlace) para acceder a su tablero.
15. Al terminar su taller seleccione la máquina en la consola, y en el menú Actions seleccione Terminate, para terminar la máquina completamente. Si no la termina, se seguirán cobrando cargos a su cuenta.

Literal 8:

```
Requirement already satisfied: packaging in ./local/lib/python3.9/site-
packages (from gunicorn) (24.1)
Installing collected packages: gunicorn
Successfully installed gunicorn-23.0.0
[ec2-user@ip-172-31-94-24 ~]$ ls
app1.py
[ec2-user@ip-172-31-94-24 ~]$
```

Literal 10:

```
),
    html.Div(children='''
        En este gráfico se observa el número de casos positivos y negati-
        vos para COVID-19 según síntomas de fiebre.
    '''),
    html.Div(
        className="Columnas",
        children=[
            html.Ul(id='my-list', children=[html.Li(i) for i in df.column
ns])
        ],
    ),
]
)

if __name__ == '__main__':
    app.run_server(host = "0.0.0.0", debug=True)
[ec2-user@ip-172-31-94-24 ~]$
```

Literal 13:

Editar reglas de entrada [Información](#)

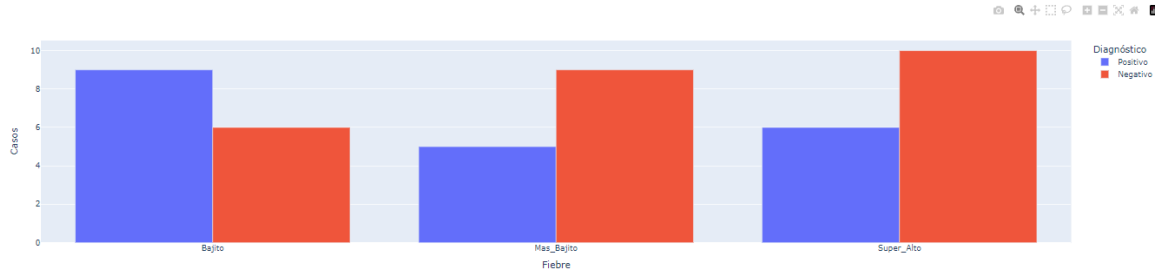
Las reglas de entrada controlan el tráfico entrante que puede llegar a la instancia.

ID de la regla del grupo de seguridad	Tipo Información	Protocolo Información	Intervalo de puertos Información	Origen Información	Descripción: opcional Información	
sgr-0e371cdc30ccb4efe	SSH	TCP	22	Pers... <input type="text"/>	<input type="text"/>	<input type="button" value="Eliminar"/>
sgr-03b13cb6dfbc05daf	TCP personalizado	TCP	8050	Pers... <input type="text"/>	<input type="text"/>	<input type="button" value="Eliminar"/>

Literal 14:

Mi primer tablero en Dash

Histograma de casos según síntomas y diagnóstico



En este gráfico se observa el número de casos positivos y negativos para COVID-19 según síntomas de fiebre.

- Fiebre
- Casos
- Diagnóstico

