Actividad 6.7 Propuesta de Dashboard con elementos streamlit

Instituto Tecnológico de Estudios Superiores de Monterrey

ANALÍTICA DE DATOS Y HERRAMIENTAS DE INTELIGENCIA ARTIFICIAL II

Grupo: 501

Profesor:

Rigoberto Cerino Jiménez

Autores:

Julio Alejandro Sotero Montiel A01656310

César Alejandro Rivera Guzmán A01567012

Diego Soto Camacho A01732608

Fecha de entrega:

25 de noviembre 2024

Introducción

En este reporte se describe la propuesta final del dashboard interactivo. El tablero está dividido en 2 secciones, una en donde analizamos la evolución de las variables a lo largo del tiempo y en la otra una sección comparativa en donde se ve más detalle el enfoque de los puntos de la cara. Durante el desarrollo de la actividad se optó por utilizar la librería de Streamlit, por mencionar alguna, se busca crear una herramienta visual intuitiva que permita explorar los datos de manera dinámica. A continuación, se explicarán los elementos visuales, como la implementación en Streamlit y la sintaxis empleada, permitiéndoles comprender a fondo el funcionamiento del Dashboard.

Propuesta final, nuestro avance:

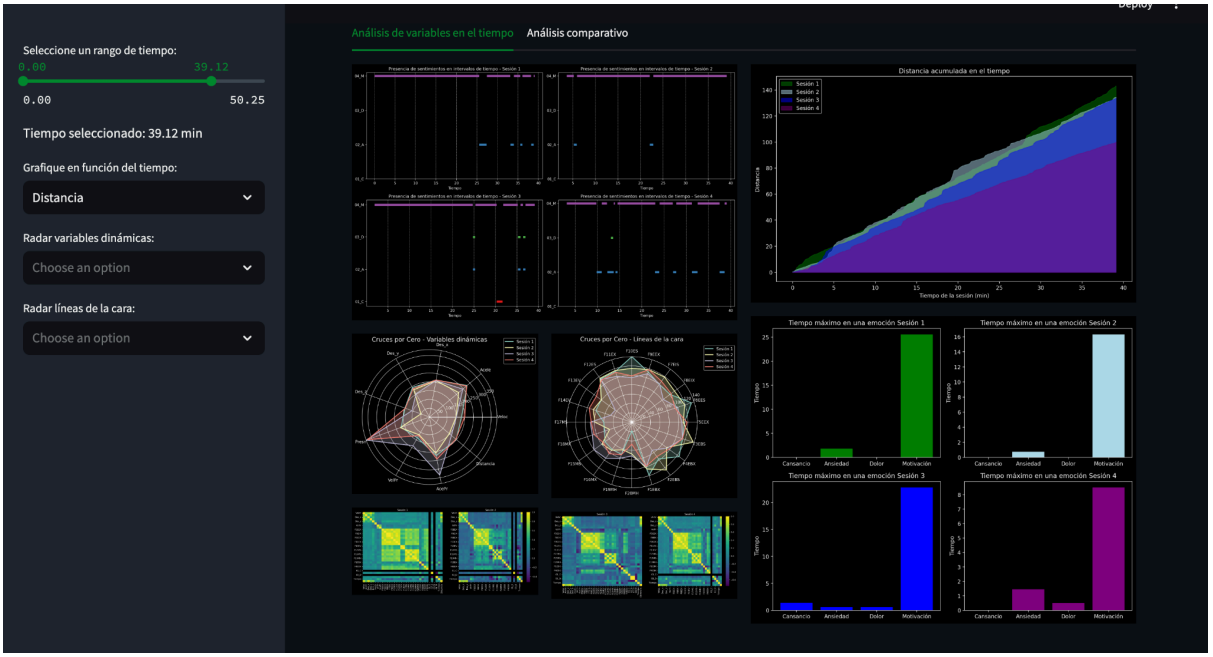

Imagen 1: Interfaz de pestaña “Análisis de variables en el tiempo”

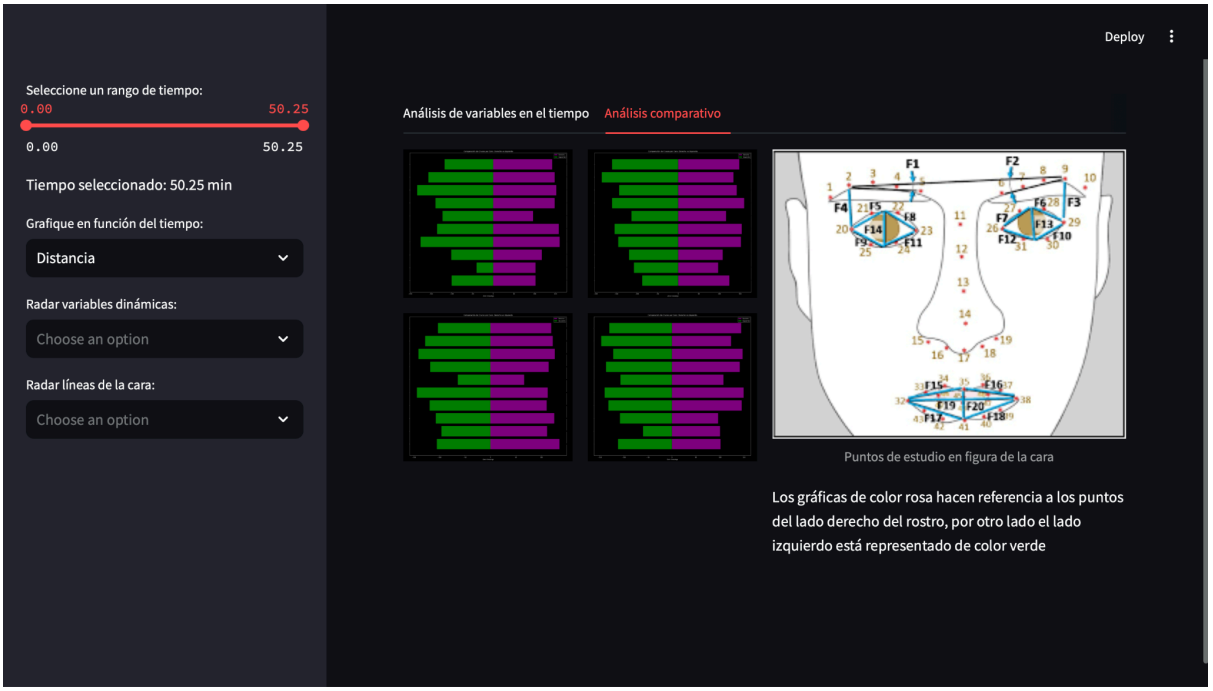

Imagen 2: Interfaz de pestaña “Análisis comparativo”

1. Explicación de cada elemento visual del Dashboard

Sidebar

- **Slider:** El slider se utiliza para hacer un “zoom in” o un “zoom out” a los datos en base al tiempo. Dado que el sidebar tiene que comparar 4 data frames y estos no son del mismo tamaño, se utiliza el tiempo del df de menor tamaño y se elimina el tiempo excedente de los data frames más grandes. El slider regresa una tupla con su valor máximo y su valor mínimo, lo que permite filtrar los data frames en rangos y se recalculan todos los gráficos en base a eso.
- **Multi-select box:** Esta sección servirá para poder eliminar las sesiones que se muestran en el dashboard. Para que el usuario pueda comparar las sesiones que desee y no necesariamente 4 todo el tiempo. Dependiendo de los valores que tome el multi select box, se definirá cuántas gráficas mostrará.

1er sección: Interfaz de pestaña “Análisis de variables en el tiempo”

- **Gráfico de intervalos emocionales**: Presenta los intervalos de tiempo donde se detecta la presencia de emociones como *Cansancio*, *Ansiedad*, *Dolor* y *Motivación* para cada sesión. Utiliza barras horizontales para representar estos intervalos en diferentes colores.

- **Gráfico de áreas acumuladas**: Muestra el comportamiento acumulado de variables como *Distancia*, *Velocidad*, *Aceleración*, etc., a lo largo del tiempo en las cuatro sesiones.
- **Radar Chart**:
- **Variables dinámicas**: Analiza los cruces por cero de las variables relacionadas con el movimiento dinámico.
- **Líneas de la cara**: Representa los cruces por cero en las variables de las líneas faciales.
- **Mapas de calor**: Comparan la correlación entre variables numéricas para dos sesiones simultáneamente.

2da sección: Análisis comparativo (enfocado a los puntos de la cara)

- **Gráficos de pirámide**: Para analizar de forma intuitiva y comparativa ambos lados de la cara y refutar nuestra tesis de que un lado se mueve más que el otro. Se optó por utilizar este gráfico ya que se pueden plasmar ambas variables que se están comparando y así se puede ver la comparación de manera visual.
- **Imagen rostro con puntos de estudio:** En la interfaz se agregó una figura del rostro de la cara, que dice cómo se clasificaron los puntos del rostro durante el proyecto, esto sirve para poder dimensionar cómo se clasificaron los puntos de la cara y poder observar qué variable de las que están graficadas hace mención a cual.
- **Gráficas de área:** En esta interfaz se busca agregar 4 gráficas de áreas para poder evaluar el comportamiento de los ojos, boca y cejas de las 4 sesiones que estamos evaluando, se busca agregarlas de la siguiente manera, se piensa agregar 4 cuadrantes esto con la finalidad de poder comparar los puntos mencionados y se quiere agregar una columna extra para poder comparar los totales de cada dataframe.

2. Explicación de cómo funciona cada elemento asociado en Streamlit (45 pts)

2.1 Gráficos de intervalos emocionales

- **Función utilizada**: `st.pyplot(fig)`
- **Explicación**:
    - El gráfico se genera utilizando `matplotlib`, que dibuja barras horizontales representan los intervalos de emociones.

- La figura es renderizada en Streamlit mediante `st.pyplot(fig)`.'

2.2 Gráfico de áreas acumuladas

- **Herramienta Streamlit**: `st.pyplot(fig2)`
- **Explicación**:
  - Se utiliza `fill_between` de `matplotlib` para crear áreas acumuladas para cada sesión.
  - `st.pyplot(fig2)` se emplea para mostrar este gráfico en la interfaz.

2.3 Radar Charts

- **Herramienta Streamlit**: `st.pyplot(combined_radar_chart)`
- **Explicación**:
  - Cada radar chart combina datos de cruces por cero de variables dinámicas o líneas faciales por sesión.
  - `matplotlib` es usado para construir gráficos radiales que se despliegan con `st.pyplot`.

2.4 Mapas de calor

- **Herramienta Streamlit**: `st.pyplot(heatmap)`
- **Explicación**:
  - `sns.heatmap` de `seaborn` genera mapas de calor con correlaciones.
  - Cuatro gráficos se muestran simultáneamente comparando sesiones seleccionadas.

---

3. Explicación de la sintaxis necesaria para programar en Python (30 pts)

3.1 Filtrado de datos

Código:
python
Copiar código

```
def filter(minn, maxx):

    df1 = df1[(df1['Tiempo'] >= minn) & (df1['Tiempo'] <= maxx)]

    return df1, df2, df3, df4
```

- **Explicación**:
    - Filtra los datos dentro del rango de tiempo seleccionado por el usuario en el slider de Streamlit. El slider retorna dos valores un una tupla, que son “minn” y “maxx”.

3.2 Construcción de gráficos

Intervalos emocionales:

1. **Inicialización de variables**:

```
sesiones = [df1, df2, df3, df4]

list_sesiones, intervalos_main, emoc = [], [], []

sentiment_columns = ['01_C', '02_A', '03_D', '04_M']
```

Lista de DataFrames (`df1`, `df2`, `df3`, `df4`), cada uno representando una sesión diferente.

`list_sesiones`, `intervalos_main`, `emoc`: Listas vacías para almacenar datos de las sesiones, intervalos de tiempo y emociones respectivamente.

- `sentiment_columns`: Lista de nombres de columnas que representan diferentes sentimientos.

2. **Creación de la figura con subplots:**

```
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))
```

- Se crea una figura (`fig`) con 4 subplots (`axs`), organizados en 2 filas y 2 columnas, con un tamaño de 15x10 pulgadas.

3. **Iteración sobre cada sesión y su subplot:**

```
for s, ax in enumerate(axs.flatten()):

    session_data = sesiones[s]
```

- Se itera sobre cada sesión (`s`) y su subplot correspondiente (`ax`).

- `session_data` contiene los datos de la sesión actual.

4. **Iteración sobre cada columna de sentimiento:**

```
for i, col in enumerate(sentiment_columns):
    intervalos = []
    start = None
```

- Se itera sobre cada columna de sentimiento (`col`).

- `intervalos` es una lista para almacenar los intervalos de tiempo donde el sentimiento está presente.

- `start` se usa para marcar el inicio de un intervalo.

5. **Filtrado de intervalos de tiempo:**

```
for j in range(len(session_data)):
    if session_data[col].iloc[j] == 1:
        if start is None:
            start = session_data['Tiempo'].iloc[j]
    else:
        if start is not None:
            end = session_data['Tiempo'].iloc[j - 1]
            intervalos.append((start, end))
            list_sesiones.append(s+1)
            intervalos_main.append((start, end))
            emoc.append(col)
            start = None
```

- Se itera sobre cada fila de `session_data`.

- Si el valor de la columna de sentimiento es 1, se marca el inicio del intervalo (`start`).

- Si el valor cambia de 1 a 0, se marca el final del intervalo (`end`) y se guarda el intervalo en `intervalos`.

- También se actualizan `list_sesiones`, `intervalos_main` y `emoc` con la sesión, el intervalo y la emoción correspondiente.

6. **Captura del último intervalo:**

```
if start is not None:
    intervalos.append((start, session_data['Tiempo'].iloc[-1]))
    list_sesiones.append(s+1)
    intervalos_main.append((start, session_data['Tiempo'].iloc[-1]))
    emoc.append(col)
```

- Si el último valor de la columna es 1, se captura el intervalo final.

7. **Dibujo de barras horizontales:**

```
for (start, end) in intervalos:
    ax.hlines(y=i, xmin=start, xmax=end, color=plt.get_cmap("Set1")(i), linewidth=6)
```

- Se dibujan barras horizontales en el subplot (`ax`) para cada intervalo, usando diferentes colores para cada sentimiento.

El código visualiza los intervalos de tiempo en los sentimientos están presentes en diferentes sesiones, utilizando subplots para mostrar cada sesión por separado.

Gráfico de áreas acumuladas:

1. Asignación de variables según la sección:

```
if 'Distancia' in SECTION:
    var = 'Distancia'
    var2 = 'Distancia'
if 'Velocidad' in SECTION:
    var = 'Velocidad'
    var2 = 'Veloc'
```

```
    if 'Aceleración' in SECTION:
        var = 'Aceleración'
        var2 = 'Acele'
    if 'Presión' in SECTION:
        var = 'Presión'
        var2 = 'Presn'
    if 'Velocidad de la presión' in SECTION:
        var = 'Velocidad de la presión'
        var2 = 'VelPr'
    if 'Aceleración de la presión' in SECTION:
        var = 'Aceleración de la presión'
        var2 = 'AcePr'
```

- Este bloque de código verifica qué tipo de variable está presente en `SECTION` y asigna valores a `var` y `var2` en consecuencia.

- `var` se usa para etiquetas y títulos, mientras que `var2` se usa para acceder a las columnas correspondientes en los DataFrames.

2. Creación de la figura:

```
    fig2, axs2 = plt.subplots(nrows=1, ncols=1, figsize=(10, 6.23))
```

- Se crea una figura (`fig2`) con un solo subplot (`axs2`), con un tamaño de 10x6.23 pulgadas.

3. Creación del gráfico de áreas:

```
    plt.fill_between(df1['Tiempo'], df1[var2].cumsum(), color='green', alpha=0.5, label='Sesión 1')
    plt.fill_between(df2['Tiempo'], df2[var2].cumsum(), color='lightblue', alpha=0.5, label='Sesión 2')
```

plt.fill_between(df3['Tiempo'], df3[var2].cumsum(), color='blue', alpha=0.5, label='Sesión 3')

plt.fill_between(df4['Tiempo'], df4[var2].cumsum(), color='purple', alpha=0.5, label='Sesión 4')

- Se utiliza `plt.fill_between` para crear gráficos de áreas acumuladas para cada sesión (`df1`, `df2`, `df3`, `df4`).

- `df1[var2].cumsum()` calcula la suma acumulada de la columna `var2` para cada DataFrame.

- Se asignan diferentes colores y etiquetas a cada sesión.

4. Etiquetas y título del gráfico:

plt.xlabel('Tiempo de la sesión (min)')

plt.ylabel(f'{var}')

plt.title(f'{var} acumulada en el tiempo')

plt.legend(loc='upper left')

plt.tight_layout()

st.pyplot(fig2)

- `plt.xlabel` y `plt.ylabel` establecen las etiquetas de los ejes X e Y respectivamente.

- `plt.title` establece el título del gráfico.

- `plt.legend` agrega una leyenda en la esquina superior izquierda.

- `plt.tight_layout` ajusta automáticamente los parámetros de la figura para que se ajuste bien dentro del área de dibujo.

- `st.pyplot(fig2)` muestra la figura utilizando Streamlit.

El código genera un gráfico de áreas acumuladas para diferentes variables (como Distancia, Velocidad, etc.) a lo largo del tiempo para cuatro sesiones diferentes, permitiendo comparar visualmente cómo se acumulan estas variables en cada sesión.

Radar Chart:

1. Función `calculate_zero_crossings`:

```
def calculate_zero_crossings(df):
    numeric_cols = df.select_dtypes(include=['number']).columns
    scaler = StandardScaler()
    df_normalized = pd.DataFrame(scaler.fit_transform(df[numeric_cols]), columns=numeric_cols)
    zero_crossings = {
        col: ((df_normalized[col].shift(1) * df_normalized[col]) < 0).sum()
        for col in df_normalized.columns
    }
    return pd.DataFrame(list(zero_crossings.items()), columns=['Column', 'Zero_Crossings'])
```

- Propósito: Calcular el número de cruces por cero para cada columna numérica en un DataFrame.

- Pasos:

  - `numeric_cols`: Selecciona las columnas numéricas del DataFrame.

  - `scaler = StandardScaler()`: Inicializa un escalador estándar para normalizar los datos.

  - `df_normalized`: Normaliza las columnas numéricas del DataFrame.

  - `zero_crossings`: Calcula los cruces por cero para cada columna normalizada. Un cruce por cero ocurre cuando el producto de un valor y el valor anterior es negativo.

  - La función devuelve un DataFrame con los nombres de las columnas y el número de cruces por cero.

2. Función `create_combined_radar_chart`:

```
def create_combined_radar_chart(radar_data, labels, title):
    fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))
```

```
    categories = radar_data[0]['Column']
    num_vars = len(categories)
    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()
    angles += angles[:1]

    for session_data, label in zip(radar_data, labels):
        values = session_data['Zero_Crossings'].tolist()
        values += values[:1]
        ax.plot(angles, values, linewidth=2, linestyle='solid', label=label)
        ax.fill(angles, values, alpha=0.25)

    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(categories, fontsize=10)
    ax.set_ylim(0, max(max(df['Zero_Crossings']) for df in radar_data))
    ax.set_title(title, size=15, color='white', pad=20)
    ax.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))
    return fig
```

- **Propósito**: Crear un gráfico de radar combinado para múltiples sesiones.
- Pasos:
  - Se crea una figura polar (`fig`) y un eje (`ax`).
  - `categories`: Obtiene las categorías (columnas) del primer DataFrame en `radar_data`.
  - `angles`: Calcula los ángulos para las variables en el gráfico de radar.
  - Para cada sesión en `radar_data`, se trazan los valores de cruces por cero en el gráfico de radar.

- Se configuran las etiquetas de los ejes, el título y la leyenda del gráfico.

3. Preparación de los DataFrames:

```
dff1 = df1.drop(columns=['F5EEX', 'F6EES', 'F8EIX', 'F7EIS', 'F9EEX', 'F10ES', 'F11EX', 'F12ES', 'F13EV', 'F14EV', 'F17MS', 'F18MX', 'F15MS', 'F16MX', 'F19MH', 'F20MH', 'F1EBX', 'F2EBS', 'F4EBX', 'F3EBS', '01_C', '02_A', '03_D', '04_M', 'Tiempo'])

dff2 = df2.drop(columns=['F5EEX', 'F6EES', 'F8EIX', 'F7EIS', 'F9EEX', 'F10ES', 'F11EX', 'F12ES', 'F13EV', 'F14EV', 'F17MS', 'F18MX', 'F15MS', 'F16MX', 'F19MH', 'F20MH', 'F1EBX', 'F2EBS', 'F4EBX', 'F3EBS', '01_C', '02_A', '03_D', '04_M', 'Tiempo'])

dff3 = df3.drop(columns=['F5EEX', 'F6EES', 'F8EIX', 'F7EIS', 'F9EEX', 'F10ES', 'F11EX', 'F12ES', 'F13EV', 'F14EV', 'F17MS', 'F18MX', 'F15MS', 'F16MX', 'F19MH', 'F20MH', 'F1EBX', 'F2EBS', 'F4EBX', 'F3EBS', '01_C', '02_A', '03_D', '04_M', 'Tiempo'])

dff4 = df4.drop(columns=['F5EEX', 'F6EES', 'F8EIX', 'F7EIS', 'F9EEX', 'F10ES', 'F11EX', 'F12ES', 'F13EV', 'F14EV', 'F17MS', 'F18MX', 'F15MS', 'F16MX', 'F19MH', 'F20MH', 'F1EBX', 'F2EBS', 'F4EBX', 'F3EBS', '01_C', '02_A', '03_D', '04_M', 'Tiempo'])
```

- Se eliminan ciertas columnas de los DataFrames `df1`, `df2`, `df3` y `df4` para crear `dff1`, `dff2`, `dff3` y `dff4`.

4. Cálculo de cruces por cero y creación de gráficos de radar:

```
lista2 = [dff1, dff2, dff3, dff4]

radar_data = [calculate_zero_crossings(df) for df in lista2]

combined_radar_chart1 = create_combined_radar_chart(
    radar_data,
    labels=["Sesión 1", "Sesión 2", "Sesión 3", "Sesión 4"],
    title="Cruces por Cero - Variables dinámicas"
)
```

- Se calcula el número de cruces por cero para cada DataFrame en `lista2`.

- Se crea un gráfico de radar combinado para las sesiones, con el título "Cruces por Cero - Variables dinámicas".

5. Repetición del proceso para otro conjunto de variables:

```
dff1 = df1.drop(columns=['Veloc', 'Acele', 'Des_x', 'Des_y', 'Des_z', 'Presn', 'VelPr', 'AcePr', '01_C', '02_A', '03_D', '04_M', 'Tiempo', 'Distancia'])

dff2 = df2.drop(columns=['Veloc', 'Acele', 'Des_x', 'Des_y', 'Des_z', 'Presn', 'VelPr', 'AcePr', '01_C', '02_A', '03_D', '04_M', 'Tiempo', 'Distancia'])

dff3 = df3.drop(columns=['Veloc', 'Acele', 'Des_x', 'Des_y', 'Des_z', 'Presn', 'VelPr', 'AcePr', '01_C', '02_A', '03_D', '04_M', 'Tiempo', 'Distancia'])

dff4 = df4.drop(columns=['Veloc', 'Acele', 'Des_x', 'Des_y', 'Des_z', 'Presn', 'VelPr', 'AcePr', '01_C', '02_A', '03_D', '04_M', 'Tiempo', 'Distancia'])



lista2 = [dff1, dff2, dff3, dff4]

radar_data = [calculate_zero_crossings(df) for df in lista2]

combined_radar_chart2 = create_combined_radar_chart(

    radar_data,

    labels=["Sesión 1", "Sesión 2", "Sesión 3", "Sesión 4"],

    title="Cruces por Cero - Líneas de la cara"

)
```

- Se repite el proceso de eliminación de columnas, cálculo de cruces por cero y creación de gráficos de radar para otro conjunto de variables, con el título "Cruces por Cero - Líneas de la cara".

El visualiza y compara el número de cruces por cero de diferentes variables en múltiples sesiones, utilizando gráficos de radar para una representación clara y concisa.

Mapas de calor:

1. Definición de la función `create_heatmap`:

```
def create_heatmap(df, df2, session_label1, session_label2):

    fig, axs = plt.subplots(1, 2, figsize=(15, 6))
```

- **Propósito**: Crear un gráfico de mapa de calor (heatmap) para dos DataFrames y mostrarlos uno al lado del otro.

- Parámetros:

  - `df`, `df2`: Los DataFrames para los cuales se crearán los mapas de calor.

  - `session_label1`, `session_label2`: Etiquetas para los títulos de los mapas de calor.

- Creación de la figura: Se crea una figura (`fig`) con dos subplots (`axs`), organizados en una fila y dos columnas, con un tamaño de 15x6 pulgadas.

2. Cálculo de las matrices de correlación:

```
numeric_cols = df.select_dtypes(include=['number']).columns

correlation_matrix = df[numeric_cols].corr()

numeric_cols = df2.select_dtypes(include=['number']).columns

correlation_matrix2 = df2[numeric_cols].corr()
```

- Selección de columnas numéricas: Se seleccionan las columnas numéricas de `df` y `df2`.

- Cálculo de la matriz de correlación: Se calculan las matrices de correlación para las columnas numéricas de `df` y `df2`.

3. Creación de los mapas de calor:

```
sns.heatmap(correlation_matrix, ax=axs[0], cmap='viridis', cbar=False)

axs[0].set_title(session_label1)

sns.heatmap(correlation_matrix2, ax=axs[1], cmap='viridis', cbar=True)

axs[1].set_title(session_label2)
```

- Mapa de calor para el primer DataFrame:

  - `sns.heatmap(correlation_matrix, ax=axs[0], cmap='viridis', cbar=False)`: Crea un mapa de calor para la matriz de correlación del primer DataFrame, sin barra de color.

- `axs[0].set_title(session_label1)`: Establece el título del primer subplot.

- Mapa de calor para el segundo DataFrame:

- `sns.heatmap(correlation_matrix2, ax=axs[1], cmap='viridis', cbar=True)`: Crea un mapa de calor para la matriz de correlación del segundo DataFrame, con barra de color.

- `axs[1].set_title(session_label2)`: Establece el título del segundo subplot.

4. Retorno de la figura:

return fig

- La función devuelve la figura (`fig`) que contiene los dos mapas de calor.

5. Creación de los gráficos:

heatmap1 = create_heatmap(df1, df2, "Sesión 1", "Sesión 2")

heatmap2 = create_heatmap(df3, df4, "Sesión 3", "Sesión 4")

- Se llama a la función `create_heatmap` para crear dos figuras de mapas de calor:

- `heatmap1` compara las correlaciones entre `df1` y `df2`, con los títulos "Sesión 1" y "Sesión 2".

- `heatmap2` compara las correlaciones entre `df3` y `df4`, con los títulos "Sesión 3" y "Sesión 4".

Este código permite visualizar y comparar las correlaciones entre las variables numéricas de dos pares de sesiones diferentes, utilizando mapas de calor para una representación clara y concisa.

3.3 Integración en Streamlit

Componentes clave:

```
SLIDER = st.sidebar.slider("Seleccione un rango de tiempo:",
value=(0.0, min(maxim)))

MULTI_SECTION1 = st.sidebar.multiselect("Radar variables
dinámicas:", options=['Sesión 1', 'Sesión 2', 'Sesión 3',
'Sesión 4'])
```