Desglose del Código Python

1 Nota

Datos en ROOT: el código utiliza el formato de datos ROOT, que es común en la física de partículas y los experimentos científicos. ROOT es un sistema de análisis de datos utilizado principalmente en experimentos de física de partículas en el CERN.[**?**]

Código Python con Explicaciones

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import plotly.express as px
import numpy as np
import pandas as pd
import ROOT
from scipy.stats import norm
```

1. Importaciones

- `dash`, `dcc`, `html`: Importa las herramientas para construir la aplicación web en Dash.
- `plotly.graph_objs`, `plotly.express`: Herramientas para crear gráficos interactivos.
- `numpy`: Para realizar cálculos numéricos y generar datos aleatorios.
- `pandas`: Para trabajar con datos en forma de tablas.
- `ROOT`: Biblioteca para trabajar con datos científicos en formato ROOT.
- `scipy.stats`: Para trabajar con distribuciones estadísticas, como la normal.

```
app = dash.Dash(__name__)
```

2. Inicialización de la Aplicación

- Para la iniciación de la aplicación web, se crea una nueva usando Dash.

```
file = ROOT.TFile.Open("https://atlas-opendata.web.cern.ch/atlas-opendata/samples/2020/GamGam/Data/data_A.GamGam.root")
tree = file.Get("mini")
```

3. Leer Datos ROOT

- se abren los datos root desde la URL donde se obtiene el árbol de datos llamado `"mini"`.

```
data = []
for event in tree:
    if event.trigP:
        for j in range(event.photon_n):
            if (event.photon_isTightID[j] and event.photon_pt[j] > 25000 and
                (abs(event.photon_eta[j]) < 2.37) and
                (abs(event.photon_eta[j]) < 1.37 or abs(event.photon_eta[j]) > 1.52)):
                data.append({
                    'photon_pt': event.photon_pt[j] / 1000,
                    'photon_eta': event.photon_eta[j],
                    'photon_phi': event.photon_phi[j],
                    'photon_E': event.photon_E[j] / 1000
                })
```

4. Extracción de Datos

- Después de obtenido esos datos, se extraen, donde se recorre cada evento en el árbol de datos y se extrae información de los fotones si cumplen ciertas condiciones
- Los datos se almacenan en una lista `data`.

```
df = pd.DataFrame(data)
```

5. Convertir a DataFrame

- Convierte la lista `data` en un DataFrame de pandas, que es una tabla de datos que facilita el análisis.

```
np.random.seed(42)
mass = np.random.normal(125, 10, 1000)
energy = np.random.normal(50, 20, 1000)
```

6. Simulación de Datos

- Genera datos ficticios para masa invariante (`mass`) y energía transversal (`energy`) usando distribuciones normales.

```
app.layout = html.Div(children=[
    html.H1(children='An lisis de Masa Invariante de Diphotones'),
    html.Div(children='''Visualizaciones y an lisis de datos.'''),
    dcc.Graph(id='histogram-graph'),
    dcc.Graph(id='heatmap-graph'),
    html.Label("Rango de Masa Invariante (GeV)"),
    dcc.RangeSlider(
        id='mass-slider',
        min=105,
        max=160,
        step=1,
        value=[120, 130],
        marks={i: f'{i}' for i in range(105, 161, 5)}
    ),
    dcc.Graph(id='histogram'),
    dcc.Graph(id='heatmap'),
    dcc.Graph(id='scatter-graph'),
    dcc.Graph(id='bar-graph'),
    dcc.Interval(
        id='interval-component',
        interval=1*1000,
        n_intervals=0
    )
])
```

7. Diseño de la Aplicación

- Define la apariencia de la aplicación web.
- Incluye títulos, gráficos y controles interactivos como sliders y un componente de actualización en tiempo real.

```
@app.callback(
    [Output('histogram-graph', 'figure'),
     Output('heatmap-graph', 'figure')],
    [Input('interval-component', 'n_intervals')]
)
def update_graph_live(n):
    current_mass = np.random.normal(125, 10, 100)
    mass_hist = np.concatenate((mass, current_mass))

    hist_data = np.histogram(mass_hist, bins=30, range=(105, 160))
    bin_centers = 0.5 * (hist_data[1][1:] + hist_data[1][:-1])

    mean, std = norm.fit(mass_hist)
    pdf = norm.pdf(bin_centers, mean, std) * len(mass_hist) * (
        hist_data[1][1] - hist_data[1][0])

    histogram_figure = {
        'data': [
            go.Bar(
                x=bin_centers,
                y=hist_data[0],
                name='Masa Invariante',
                marker=dict(color='blue', opacity=0.6)
            ),
            go.Scatter(
                x=bin_centers,
                y=pdf,
                mode='lines',
                name='Ajuste Gaussiano',
                line=dict(color='red')
            )
        ],
        'layout': go.Layout(
            title='Histograma de Masa Invariante de Diphotones',
            xaxis={'title': 'Masa Invariante [GeV]'},
            yaxis={'title': 'Eventos'},
            showlegend=True
        )
    }

    heatmap_figure = {
        'data': [
            go.Histogram2d(
                x=mass,
                y=energy,
                colorscale='Jet'
            )
        ],
        'layout': go.Layout(
            title='Mapa de Densidad de Eventos',
```

```
            xaxis={'title': 'Masa Invariante [GeV]'},
            yaxis={'title': 'Energ a Transversal [GeV]'},
            coloraxis={'colorbar': {'title': 'Eventos'}}
        )
    }

    return histogram_figure, heatmap_figure
```

8. Actualizar Gráficos en Tiempo Real

- Actualiza el histograma y el mapa de calor en la interfaz de usuario cada segundo con datos simulados.
- Calcula un histograma y ajusta una distribución normal a los datos de masa invariante.

```
@app.callback(
    Output('histogram', 'figure'),
    [Input('mass-slider', 'value')]
)
def update_histogram(mass_range):
    filtered_df = df[(df['photon_pt'] >= mass_range[0]) & (df['
        photon_pt'] <= mass_range[1])]
    fig = px.histogram(filtered_df, x='photon_pt', nbins=30, title=
        'Diphoton Invariant Mass')
    fig.update_layout(xaxis_title='Invariant Mass m_{yy} [GeV]',
        yaxis_title='Events', bargap=0.2)
    return fig
```

9. Actualizar Histograma con Slider

- Actualiza el histograma basado en el rango de masa invariante seleccionado con el slider.

```
@app.callback(
    Output('heatmap', 'figure'),
    [Input('mass-slider', 'value')]
)
def update_heatmap(mass_range):
    x = np.random.normal(125, 10, 1000)
    y = np.random.normal(50, 20, 1000)
    fig = go.Figure(go.Histogram2d(x=x, y=y, nbinsx=50, nbinsy=50,
        colorscale='Jet'))
    fig.update_layout(title='Mapa de densidad de eventos',
        xaxis_title='Masa invariante [GeV]', yaxis_title='Energ a
        transversal [GeV]')
    return fig
```

10. Actualizar Heatmap con Slider

- Actualiza el mapa de calor con datos ficticios en función del rango de masa invariante seleccionado.

```
@app.callback(
    Output('scatter-graph', 'figure'),
    [Input('mass-slider', 'value')]
)
def update_scatter(mass_range):
    filtered_df = df[(df['photon_pt'] >= mass_range[0]) & (df['
        photon_pt'] <= mass_range[1])]
    fig = px.scatter(filtered_df, x='photon_pt', y='photon_E',
        title='Scatter Plot de Pt vs Energ a')
    fig.update_layout(xaxis_title='Pt [GeV]', yaxis_title='Energ a
         [GeV]')
    return fig
```

11. Actualizar Scatter Plot con Slider

- Actualiza el gráfico de dispersión en función del rango de masa invariante seleccionado.

```
@app.callback(
    Output('bar-graph', 'figure'),
    [Input('mass-slider', 'value')]
)
def update_bar(mass_range):
    filtered_df = df[(df['photon_pt'] >= mass_range[0]) & (df['
        photon_pt'] <= mass_range[1])]
    fig = px.histogram(filtered_df, x='photon_pt', nbins=30, title=
        'Distribuci n de Pt')
    fig.update_layout(xaxis_title='Pt [GeV]', yaxis_title='Conteo')
    return fig
```

12. Actualizar Gráfico de Barras con Slider

- Actualiza el gráfico de barras basado en el rango de masa invariante seleccionado.

```
if __name__ == '__main__':
    app.run_server(debug=True, host='0.0.0.0', port=8050)
```

13. Ejecutar el Servidor

- Ejecuta el servidor web para que la aplicación sea accesible en la red local en el puerto 8050.