```
"""
Defining a Dash web application for portfolio optimization analysis. The application allows
users to input ticker symbols and specify a risk preference to analyze and visualize the
optimal portfolio allocation and other related metrics.

The application features the following components:
1. **External Stylesheets**: Uses external stylesheets for theming and font styling.
2. **Layout**:
   - **Header**: Displays the title of the dashboard.
   - **Trending Tickers Section**: Shows trending tickers and their associated rectangles.
   - **Ticker Input and Risk Preference Section**: Allows users to input ticker symbols,
adjust a risk preference slider, and submit the data.
   - **Key Statistics Table**: Displays key financial statistics for the provided tickers.
   - **Efficient Frontier Graph**: Visualizes the efficient frontier for the portfolio.
   - **Portfolio Pie Chart**: Shows the optimal portfolio allocation in a pie chart.
   - **Weight Adjustment Section**: Allows users to adjust asset weights and submit new
weights.

Callbacks:
1. **`update_key_stats_table`**:
   - **Inputs**: Triggered by the submit button click and inputs from ticker symbols and
risk preference.
   - **Outputs**: Updates the key statistics table, error messages, and stores relevant
data.
   - **Function**: Processes the ticker inputs, checks validity, retrieves key statistics,
and prepares data for further analysis.

2. **`update_output`**:
   - **Inputs**: Triggered by the submit button click and includes data from storage
components and ticker inputs.
   - **Outputs**: Updates the efficient frontier graph, pie chart, and weight sliders based
on the provided tickers and risk preference.
   - **Function**: Calculates the efficient frontier, optimal portfolio, and other metrics.
Updates the UI components with the calculated data.

3. **`update_efficient_frontier`**:
   - **Inputs**: Triggered by the submit new weights button click and includes ticker
symbols, adjusted weights, risk preference, and existing figure data.
   - **Outputs**: Updates the efficient frontier graph with both the optimal and manually
adjusted portfolios. Provides updated return and standard deviation information.
   - **Function**: Processes user-adjusted weights, recalculates the efficient frontier, and
updates the visualization to reflect the adjusted portfolio.

Notes:
- The application uses Plotly for interactive visualizations and Dash components for the
user interface.
- Data handling includes downloading and processing financial data, calculating statistical
measures, and performing portfolio optimization.

To run the application, execute this script directly. The server will start in debug mode,
and the application will be accessible through a web browser.
"""
external_stylesheets = [dbc.themes.SLATE, {'href': 'https://fonts.googleapis.com/css2?
family=Lato&display=swap', 'rel': 'stylesheet'}]
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div(style={'backgroundColor': '#000000', 'fontFamily': 'Lato, sans-
serif'}, children=[
    html.H1("Portfolio Optimization Dashboard", style={'text-align': 'center', 'font-
weight': 'bold', 'padding-top': '20px', 'color': 'white'}),
    dbc.Row([
        dbc.Col(
    [
        html.Div(
            [
                html.H2("Trending Tickers from Yahoo Finance", style={
                    'text-align': 'center',
```

```
                    'padding': '25px',
                    'font-size': '28px',
                    'font-weight': 'bold',
                    'color': 'white',
                    'font-family': 'Lato, sans-serif'
                }),
                html.Div(generate_ticker_rectangles(), style={
                    'display': 'flex',
                    'flexWrap': 'wrap',
                    'justifyContent': 'center'
                })
            ],
            style={
                'background-color': 'rgba(51, 51, 51, 0.4)',
                'border-radius': '15px',
                'padding': '20px',
                'margin': '10px',
                'width': '100%',
                'height': '95%',
                'display': 'flex',
                'flexDirection': 'column',
                'justifyContent': 'center',
                'alignItems': 'center'
            }
        )
    ],
    width=6, style={'margin-left': '25px', 'margin-right': '5px'}
),
        dbc.Col([
            html.Div(
                style={
                    'background-color': 'rgba(51, 51, 51, 0.4)',
                    'border-radius': '15px',
                    'padding': '20px',
                    'margin': '10px',
                    'width': '100%',
                    'height': '95%'
                },
                children=[
                    html.Div(id='initial-message', children="Please enter at least one
ticker and specify the risk preference", style={'text-align': 'center', 'font-size':
'28px','color': 'white', 'padding': '20px', 'fontFamily': 'Lato, sans-serif', 'font-
weight': 'bold'}),
                    html.Div([
                        dcc.Input(id=f'ticker-input-{i}', type='text', placeholder=f'Ticker
{i+1}', style={'margin-bottom': '10px', 'fontFamily': 'Lato', 'text-align': 'center'}) for
i in range(5)
                        ], style={
                            'display': 'flex',
                            'flex-direction': 'column',
                            'align-items': 'center',
                            'margin-top': '20px'
                            }),
                    html.Div([
                        dcc.Slider(
                            id='risk-slider',
                            min=0,
                            max=100,
                            step=1,
                            marks={0: 'Low', 50: 'Moderate', 100: 'High'},
                            value=50,
                        ),
                        html.Div(id='slider-output', style={'text-align': 'center',
'padding': '10px', 'fontFamily': 'Lato'}),
                        dcc.Store(id='risk-preference-store', data=50)
                    ], style={
                        'maxWidth': '400px',
```

```
                        'margin': '0 auto',
                        'width': '100%',
                        'padding-top': '20px'
                    }),
                    html.Div([
                        dbc.Button('Submit', id='submit-button', n_clicks=0,
color='primary', style={'margin': '10px'}),
                    ], style={
                        'text-align': 'center',
                        'padding-top': '20px'
                    }),
                    html.Div(id='error-message-ticker', style={'text-align': 'center',
'padding': '20px', 'fontFamily': 'Lato'})
                ]
            )
        ], width=5, style={'margin-left': '25px'})
    ]),
    html.Div(
    [
        html.H2(
            "Key Statistics of Selected Tickers",
            style={'font-weight': 'bold', 'text-align': 'center', 'color': 'white'}
        ),
        dcc.Loading(
            id="loading-key-stats",
            type="default",
            children=dash_table.DataTable(
                id='key-stats-table',
                columns=[
                    {"name": "Ticker", "id": "Ticker"},
                    {"name": "Market Cap", "id": "Market Cap"},
                    {"name": "Trailing P/E", "id": "Trailing P/E"},
                    {"name": "PEG Ratio", "id": "PEG Ratio"},
                    {"name": "Price/Sales", "id": "Price/Sales"},
                    {"name": "Enterprise Value", "id": "Enterprise Value"},
                    {"name": "EV/Revenue", "id": "EV/Revenue"}
                ],
                data=[],
                style_table={'height': '200px', 'overflowY': 'auto', 'backgroundColor':
'transparent'},
                style_cell={
                    'backgroundColor': 'transparent',
                    'color': 'white',
                    'border': 'none',
                    'fontFamily': 'Lato, sans-serif',
                    'text-align': 'center',
                    'color': '#00ffff',
                    'font-size': '20px'
                },
                style_header={
                    'backgroundColor': 'transparent',
                    'color': '#00ffff',
                    'border': 'none',
                    'font-weight': 'bold'
                },
                tooltip_data=[],
                tooltip_duration=None,
                tooltip_header={
                    col: {'value': TOOLTIP_TEXT[col], 'type': 'markdown'} for col in
TOOLTIP_TEXT
                }
            )
        )
    ],
    style={
        'backgroundColor': 'rgba(51, 51, 51, 0.4)',
        'borderRadius': '15px',
```

```
        'padding': '20px',
        'margin-left': '40px',
        'margin-right': '40px',
    }
),
    dcc.Loading(
        id="loading-efficient-frontier",
        type="default",
        children=html.Div(
            dcc.Graph(
                id='efficient-frontier1',
                style={'backgroundColor': 'transparent', 'padding': '20px'}
            ),
            style={
                'backgroundColor': 'rgba(51, 51, 51, 0.4)',
                'borderRadius': '15px',
                'padding': '20px',
                'margin': '10px',
                'margin-left': '40px',
                'margin-right': '40px'
            }
        )
    ),
    dcc.Store(id='combined-data-store'),
    dcc.Store(id='data-store', data={}),
    dcc.Store(id='young-tickers-store', data={}),
    dcc.Store(id='invalid-tickers-store', data={}),
    html.Div(id='optimal-return-info', style={'fontFamily': 'Lato'}),
    html.Div(id='optimal-stddv-info', style={'fontFamily': 'Lato'}),
    html.Div(id='adjusted-return-info', style={'fontFamily': 'Lato', 'textAlign':
'center'}),
    html.Div(id='adjusted-stddv-info', style={'fontFamily': 'Lato', 'textAlign':
'center'}),
    html.Div([
        dbc.Row([
            dbc.Col(
                dcc.Loading(
                    id="loading-pie-chart",
                    type="default",
                    children=dcc.Graph(
                        id='portfolio-pie-chart',
                        style={'padding': '20px', 'backgroundColor': 'rgba(0,0,0,0)'}
                    ),
                ),
                width=6,
                style={'margin-left': '25px', 'margin-right': '5px', 'border-radius':
'15px', 'background-color': 'rgba(51, 51, 51, 0.4)', 'padding': '20px'}
            ),
            dbc.Col(
                html.Div([
                    html.P("Please adjust asset weights if needed", style={'fontFamily':
'Lato', 'font-size': '20px', 'color': 'white', 'textAlign': 'center', 'margin-bottom':
'10px'}),
                    html.Div(id='weight-sliders', style={'padding': '20px',
'backgroundColor': 'rgba(0,0,0,0)'}),
                    dbc.Row(
                        dbc.Button('Submit new weights', id='submit-button2', n_clicks=0,
color='secondary', style={'margin': '10px', 'width': 'auto'}),
                        justify='center'
                        ),
                    html.Div(id='weight-error-message', style={'text-align': 'center',
'color': 'red', 'margin-top': '10px'})  # Add this line
                    ]),
                width=5,
                style={'margin-left': '25px', 'border-radius': '15px', 'background-color':
'rgba(51, 51, 51, 0.4)', 'padding': '20px'}
                ),
```

```
            ], style={'padding': '20px'}),
        html.Div(id='sliders-output', style={'fontFamily': 'Lato', 'textAlign': 'center',
'marginTop': '20px'})
    ])
])

@app.callback(
    [Output('key-stats-table', 'data'),
     Output('error-message-ticker', 'children'),
     Output('data-store', 'data'),
     Output('invalid-tickers-store', 'data'),
     Output('young-tickers-store', 'data'),
     Output('risk-preference-store', 'data')],
    [Input('submit-button', 'n_clicks')],
    [State(f'ticker-input-{i}', 'value') for i in range(5)] + [State('risk-slider',
'value')],
    prevent_initial_call=True
)
def update_key_stats_table(n_clicks, *args):
    """
    Updates the key statistics table based on the tickers provided by the user.

    This function processes the input tickers, checks for validity, and retrieves key
statistics
    for the valid tickers. It returns the key statistics data, error messages (if any), and
other
    relevant data for storage.

    Args:
        n_clicks (int): Number of clicks on the submit button. Used to trigger the update.
        *args: Variable length argument list for ticker inputs and risk preference. The last
argument
               is expected to be the risk preference, and the preceding arguments are ticker
symbols.

    Returns:
        tuple: A tuple containing the following elements:
            - key_stats_data (list): A list of dictionaries with ticker symbols and their
key statistics.
            - error_message (str): An error message string if tickers are invalid or if none
are provided.
            - data_json (str): A JSON string containing the combined data.
            - invalid_tickers_json (str): A JSON string containing the list of invalid
tickers.
            - young_tickers_json (str): A JSON string containing the list of tickers with
insufficient data.
            - risk_preference_json (str): A JSON string containing the risk preference.
    """
    tickers = [ticker for ticker in args[:-1] if ticker]
    risk_preference = args[-1]

    if n_clicks > 0 and not tickers:
        return [], "Please enter at least one ticker.", [], [], [], []

    if not tickers:
        return [], "", [], [], [], []

    data, young_tickers, invalid_tickers = download_data_fillna(tickers, start_date="2023-
01-03", end_date=datetime.today()-timedelta(days=1))

    if invalid_tickers:
        return [], f"Ticker is invalid: {', '.join(invalid_tickers)}", [], [], [], []

    if young_tickers:
        return [], f"Ticker has too little available information to be used: {',
'.join(young_tickers)}", [], [], [], []
```

```
    if not data.empty:
        key_stats = get_key_statistics(data.columns)
        key_stats_data = [
            {"Ticker": ticker, **stats} for ticker, stats in key_stats.items()
        ]
    else:
        key_stats_data = []

    return (
        key_stats_data,
        html.Span(f'Risk Preference: {risk_preference:.2f}%', style={'color': 'white'}),
        data.to_json(),  # Convert combined_data to JSON for storage
        json.dumps(invalid_tickers),
        json.dumps(young_tickers),
        json.dumps(risk_preference)
    )

# Second Callback: update_output
@app.callback(
    [Output('slider-output', 'children'),
     Output('efficient-frontier1', 'figure', allow_duplicate=True),
     Output('optimal-return-info', 'children'),
     Output('optimal-stddv-info', 'children'),
     Output('portfolio-pie-chart', 'figure'),
     Output('weight-sliders', 'children'),
     Output('combined-data-store', 'data')],
    [Input('submit-button', 'n_clicks'),
     Input('data-store', 'data'),
     Input('invalid-tickers-store', 'data'),
     Input('young-tickers-store', 'data'),
     Input('risk-preference-store', 'data')],
    [State(f'ticker-input-{i}', 'value') for i in range(5)],
    prevent_initial_call=True
)
def update_output(n_clicks, data_json, invalid_tickers_json, young_tickers_json,
risk_preference_json, *args):
    """
    Updates the output components of the dashboard based on the tickers and risk preference.

    This function calculates the efficient frontier, optimal portfolio, and other relevant
metrics
    based on the tickers and risk preference provided. It updates the graph, pie chart, and
weight sliders
    accordingly.

    Args:
        n_clicks (int): Number of clicks on the submit button. Used to trigger the update.
        data_json (str): JSON string containing the combined data for selected tickers.
        invalid_tickers_json (str): JSON string containing the list of invalid tickers.
        young_tickers_json (str): JSON string containing the list of tickers with
insufficient data.
        risk_preference_json (str): JSON string containing the user's risk preference.
        *args: Variable length argument list for ticker inputs.

    Returns:
        tuple: A tuple containing the following elements:
            - slider_output (str): Text representation of the risk preference.
            - fig (dict): A Plotly figure dictionary representing the efficient frontier
graph.
            - optimal_return_info (str): Text displaying the optimal portfolio's expected
return.
            - optimal_stddv_info (str): Text displaying the optimal portfolio's standard
deviation (risk).
            - pie_chart (dict): A Plotly figure dictionary representing the portfolio
allocation pie chart.
            - weight_sliders (list): A list of Dash components for adjusting asset weights.
            - combined_data_json (str): A JSON string containing the combined data
```

```
including forecasts.
    """
    tickers = [ticker for ticker in args if ticker]
    risk_preference = json.loads(risk_preference_json) / 100
    print(risk_preference)

    if not tickers:
        return "", {}, "", "", {}, [], pd.DataFrame()

    data = pd.read_json(StringIO(data_json))
    invalid_tickers = json.loads(invalid_tickers_json)
    young_tickers = json.loads(young_tickers_json)

    forecasted_data, arima_params = forecast_arima(data)
    combined_data = pd.concat([data, forecasted_data], axis=0)
    daily_returns = calculate_daily_returns(combined_data)
    mean_returns, corr_matrix, cov_matrix = calculate_statistics(daily_returns)
    optimal_portfolio = optimize_portfolio(mean_returns, cov_matrix,
risk_preference=risk_preference)

    num_assets = len(mean_returns)
    num_steps = 17
    weights_range = np.linspace(0, 1, num_steps)
    weights_grid = np.array(list(product(weights_range, repeat=num_assets)))
    valid_weights = weights_grid[np.isclose(weights_grid.sum(axis=1), 1)]

    num_portfolios = len(valid_weights)
    results = np.zeros((3, num_portfolios))
    weight_array = np.zeros((num_portfolios, num_assets))

    for i, weights in enumerate(valid_weights):
        portfolio_return, portfolio_stddev = portfolio_performance(weights, mean_returns,
cov_matrix)
        results[0, i] = portfolio_return * 100
        results[1, i] = portfolio_stddev * 100
        results[2, i] = portfolio_return / portfolio_stddev
        weight_array[i, :] = weights

    trace1 = go.Scatter(
        x=results[1, :],
        y=results[0, :],
        mode='markers',
        marker=dict(
            color=results[2, :],
            colorscale='Viridis',
            showscale=True,
            size=5
        ),
        text=[f"Weights: {', '.join([f'{ticker}: {weight * 100:.2f}%' for ticker, weight in
zip(tickers, weight_array[int(idx)])])}" for idx in range(num_portfolios)],
        hoverinfo='text'
    )

    fig = go.Figure(data=[trace1])

    optimal_weights = optimal_portfolio.x
    optimal_return, optimal_stddev = portfolio_performance(optimal_weights, mean_returns,
cov_matrix)
    optimal_return *= 100
    optimal_stddev *= 100

    trace2 = go.Scatter(
        x=[optimal_stddev],
        y=[optimal_return],
        mode='markers',
        marker=dict(color='red', size=20, line=dict(color='black', width=2)),
        showlegend=False,
```

```
        hovertext=f"Optimal Weights: {', '.join([f'{ticker}: {weight * 100:.2f}%' for
ticker, weight in zip(tickers, optimal_weights)])}",
        hoverinfo='text'
    )

    fig.add_trace(trace2)

    for ticker in tickers:
        # Calculate returns and stddev for 100% allocation to this asset
        single_asset_weights = np.zeros(len(tickers))
        single_asset_weights[tickers.index(ticker)] = 1.0
        single_asset_return, single_asset_stddev =
portfolio_performance(single_asset_weights, mean_returns, cov_matrix)
        single_asset_return *= 100
        single_asset_stddev *= 100

        # Add trace for this point
        fig.add_trace(go.Scatter(
            x=[single_asset_stddev],
            y=[single_asset_return],
            mode='markers',
            marker=dict(color='green', size=12, line=dict(color='black', width=2)),
            name=f'100% {ticker}',
            showlegend=True,
            hovertext=f"100% {ticker}",
            hoverinfo='text'
        ))

    fig.update_layout(
    title={
        'text': 'Efficient Frontier with Optimal Portfolio',
        'x': 0.5,
        'font': {
            'size': 30,
            'family': 'Lato',
            'weight': 'bold'
        }
    },
    xaxis={
        'title': {
            'text': 'Portfolio Risk (Standard Deviation %)',
            'font': {
                'family': 'Lato',
                'weight': 'bold',
                'size': 20
            }
        }
    },
    yaxis={
        'title': {
            'text': 'Portfolio Return %',
            'font': {
                'family': 'Lato',
                'weight': 'bold',
                'size': 20
            }
        }
    },
    showlegend=False,
    template='plotly_dark'
)

    colors = ['#00FFFF', '#7FFFD4', '#76EEC6', '#66CDAA', '#458B74']

    pie_chart = go.Figure(
    data=[go.Pie(
        labels=tickers,
```

```
        hole=0.7,
        values=[round(weight * 100, 1) for weight in optimal_weights],
        hoverinfo='label+percent',
        textinfo='percent',
        marker=dict(colors=colors)
    )]
)
    pie_chart.update_layout(
    title={
        'text': 'Optimal Portfolio Weights',
        'x': 0.5,
        'font': {
            'size': 30,
            'family': 'Lato',
            'weight': 'bold'
        }
    },
    legend={
        'font': {
            'family': 'Lato',
            'size': 20,
            'weight': 'bold'
        }
    },
    template='plotly_dark'
)

    sliders = [html.Div([
        html.Label(ticker),
        dcc.Slider(
            id={'type': 'weight-slider', 'index': i},
            min=0,
            max=100,
            step=1,
            value=round(weight * 100, 1),
            marks={i: f'{i}%' for i in range(0, 101, 10)}
        ),
        html.Div(id={'type': 'sliders-output', 'index': i})
    ]) for i, (ticker, weight) in enumerate(zip(tickers, optimal_weights))]

    return (
        "",
        fig,
        html.Div(f'Optimal Portfolio Return: {optimal_return:.2f}%', style={'textAlign':
'center', 'color': 'white'}),
        html.Div(f'Optimal Standard Deviation: {optimal_stddev:.2f}%', style={'textAlign':
'center', 'color': 'white'}),
        pie_chart,
        sliders,
        combined_data.to_json()
    )

@app.callback(
    [Output('sliders-output', 'children', allow_duplicate=True),
     Output('efficient-frontier1', 'figure'),
     Output('adjusted-return-info', 'children'),
     Output('adjusted-stddv-info', 'children'),
     Output('weight-error-message', 'children')],
    [Input('submit-button2', 'n_clicks')],
    [State(f'ticker-input-{i}', 'value') for i in range(5)] + [State({'type': 'weight-
slider', 'index': ALL}, 'value')] + [State('risk-slider', 'value')] + [State('efficient-
frontier1', 'figure')] +
    [State('combined-data-store', 'data')],
    prevent_initial_call=True
)
def update_efficient_frontier(n_clicks, *args):
    """
```

```
    Updates the efficient frontier graph with both optimal and manually adjusted portfolios.

    This function processes the user input for tickers, adjusted weights, risk preference,
and
    existing figure, and updates the efficient frontier graph to include the adjusted
portfolio.
    It also calculates and returns the adjusted return and standard deviation, and handles
errors
    related to invalid weights.

    Args:
        n_clicks (int): Number of clicks on the submit button. Used to trigger the update.
        *args: Variable length argument list containing:
            - Ticker symbols (up to 5) as strings.
            - Adjusted weights for the tickers as a list of floats.
            - Risk preference as a float, which is converted to a percentage.
            - Existing figure dictionary representing the current state of the efficient
frontier graph.
            - Combined data in JSON format for calculating returns and statistics.

    Returns:
        tuple: A tuple containing the following elements:
            - error_message (str): An error message if the sum of weights is not equal to
100%.
            - updated_figure (go.Figure): A Plotly figure dictionary representing the
updated efficient frontier graph.
            - adjusted_return_info (html.Div): HTML element displaying the adjusted
portfolio return.
            - adjusted_stddev_info (html.Div): HTML element displaying the adjusted
portfolio standard deviation.
            - additional_info (str): Additional information or error messages related to the
adjusted weights.

    Notes:
        - If the sum of the adjusted weights is not equal to 100, an error message is
returned.
        - If the weights are valid, the function calculates and visualizes the adjusted
portfolio on the efficient frontier.
    """
    tickers = [ticker for ticker in args[:5] if ticker]
    adjusted_weights = args[5]
    risk_preference = args[-3] / 100
    existing_figure_dict = args[-2]
    combined_data_json = args[-1]
    combined_data = pd.read_json(StringIO(combined_data_json))

    normalized_adjusted_weights = np.array(adjusted_weights) / 100

    existing_figure = go.Figure(existing_figure_dict)

    if np.sum(normalized_adjusted_weights) != 1:
        return "The sum of weights should be equal to 100", existing_figure, "Adjusted
return info is not available due to invalid weights.", "Adjusted standard deviation info is
not available due to invalid weights.", "Sum of weights should be equal to 100."  # Update
this line

    daily_returns = calculate_daily_returns(combined_data)
    mean_returns, _, cov_matrix = calculate_statistics(daily_returns)

    adjusted_return, adjusted_stddev = portfolio_performance(normalized_adjusted_weights,
mean_returns, cov_matrix)
    adjusted_return *= 100
    adjusted_stddev *= 100

    trace_adjusted = go.Scatter(
        x=[adjusted_stddev],
        y=[adjusted_return],
```

```
        mode='markers',
        marker=dict(color='blue', size=17, line=dict(color='black', width=2)),
        showlegend=False,
        hovertext=f"Adjusted Weights: {', '.join([f'{ticker}: {normalized_adjusted_weight *
100:.2f}%' for ticker, normalized_adjusted_weight in zip(tickers,
normalized_adjusted_weights)])}",
        hoverinfo='text'
    )

    existing_figure.add_trace(trace_adjusted)

    existing_figure.update_layout(
        title={
        'text': 'Efficient Frontier with Optimal and Manually Adjusted Portfolio',
        'x': 0.5,
        'font': {
            'size': 40,
            'family': 'Lato',
            'weight': 'bold'
        }},
        xaxis=dict(title='Portfolio Risk (Standard Deviation %)'),
        yaxis=dict(title='Portfolio Return %'),
        showlegend=False,
        template='plotly_dark',
        title_x=0.5,
        title_font=dict(size=24)
    )

    return (
        "",
        existing_figure,
        html.Div(
            f'Adjusted Portfolio Return: {adjusted_return:.2f}%',
            style={'textAlign': 'center', 'color': 'white'}
        ),
        html.Div(
            f'Adjusted Standard Deviation: {adjusted_stddev:.2f}%',
            style={'textAlign': 'center', 'color': 'white'}
        ),
        ""  # Clear the error message if the weights are valid
    )

if __name__ == '__main__':
    app.run_server(debug=True)
```