

4. Sets up a Dash web application that functions as a "Portfolio Optimization Dashboard", which allows users to input stock tickers, set a risk preference, and see optimized portfolio results based on those inputs. The dashboard is composed of various elements, including input fields, sliders, and visualizations like data tables and charts.

```
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([
        # Left column with trending tickers rectangles
        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)', # Dark gray with 40% opacity
                        'border-radius': '15px',
                        'padding': '20px',
                        'margin': '10px',
                        'width': '100%', # Fixed width for consistency
                        'height': '95%',
                        'display': 'flex', # Use Flexbox
                        'flexDirection': 'column', # Stack children vertically
                        'justifyContent': 'center', # Center vertically
                        'alignItems': 'center' # Center horizontally
                    }
                )
            ],
            width=6, style={'margin-left': '25px', 'margin-right': '5px'} # Adjust width and margins
        ),
        # Right column with input fields, slider, and submit button moved lower
        dbc.Col([
            html.Div(
                style={
                    'background-color': 'rgba(51, 51, 51, 0.4)', # Dark gray with 40% opacity
                    'border-radius': '15px',
                    'padding': '20px',
                    'margin': '10px',
                    'width': '100%', # Fixed width for consistency
                    '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-stddev-info', style={'fontFamily': 'Lato'}),
html.Div(id='adjusted-return-info', style={'fontFamily': 'Lato', 'textAlign':
'center'}),
html.Div(id='adjusted-stddev-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'), # Added to pass combined_data
    Output('invalid-tickers-store', 'data'), # Added to pass invalid_tickers
    Output('young-tickers-store', 'data'),
    Output('risk-preference-store', 'data')], # Added to pass risk preference
    [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):
    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), # Convert invalid_tickers to JSON
    json.dumps(young_tickers), # Convert young_tickers to JSON
    json.dumps(risk_preference) # Convert risk_preference to JSON
)

# 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'), # Added to receive combined_data
     Input('invalid-tickers-store', 'data'), # Added to receive invalid_tickers
     Input('young-tickers-store', 'data'),
     Input('risk-preference-store', 'data')], # Added to receive risk_preference
    [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):
    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')], # Add this line
    [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):
    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)

```