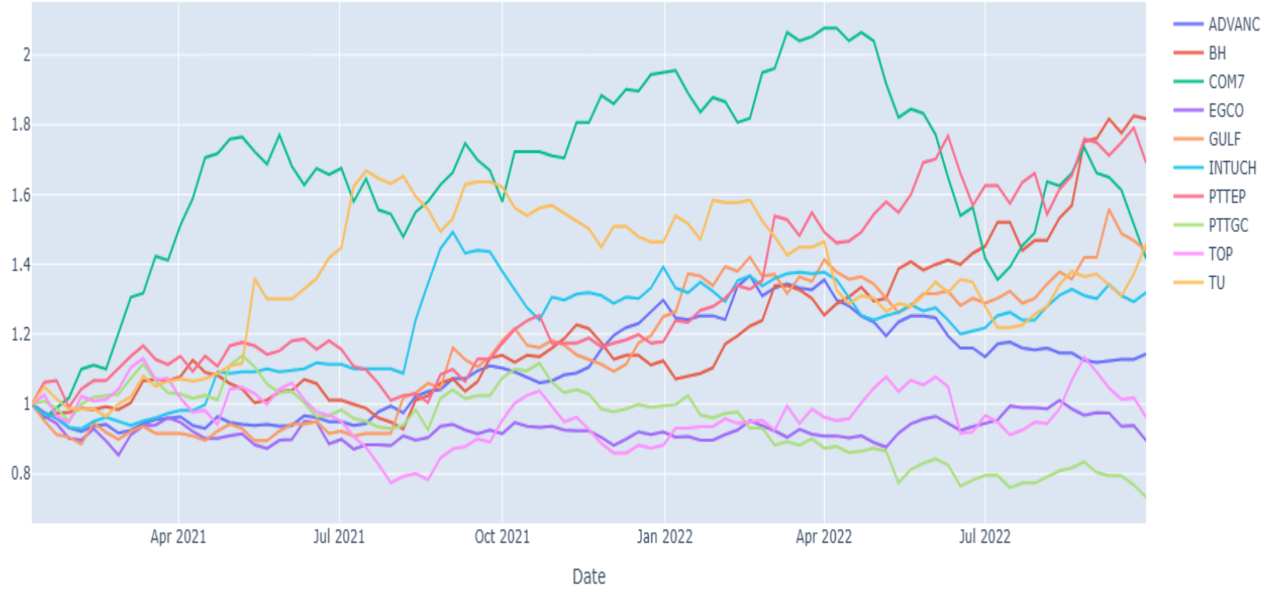
Visualizing-Data-A-Journey-Through-Insights-with-Python

This project is dedicated to the art and science of data visualization using Python. In a world driven by data, effective visualization is the bridge between raw information and actionable insights. Through this project, I aim to showcase how complex datasets can be transformed into compelling, interactive, and insightful visual stories.

Time series analysis

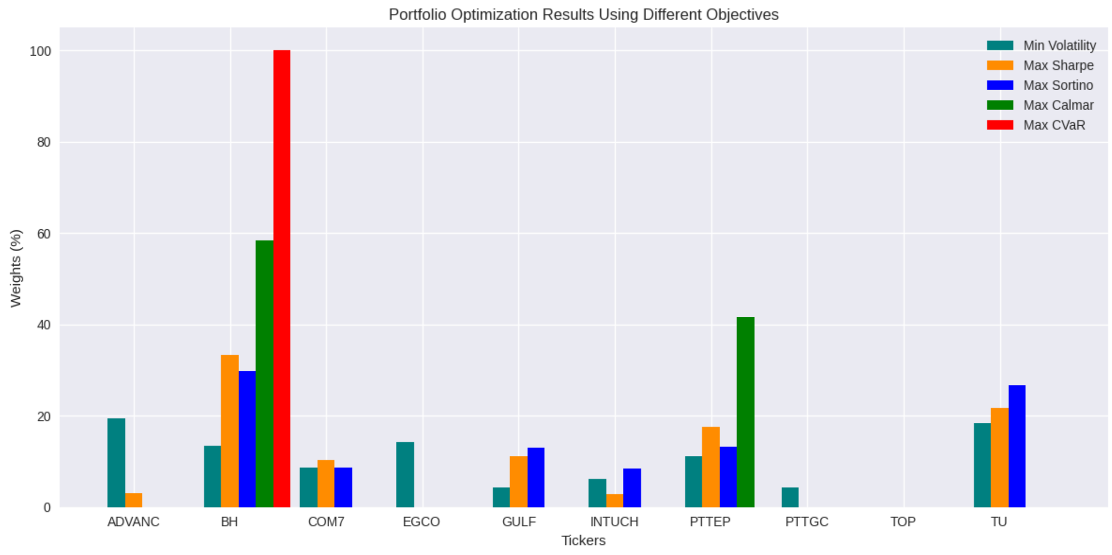
Example 1: Normalized Prices



Code:

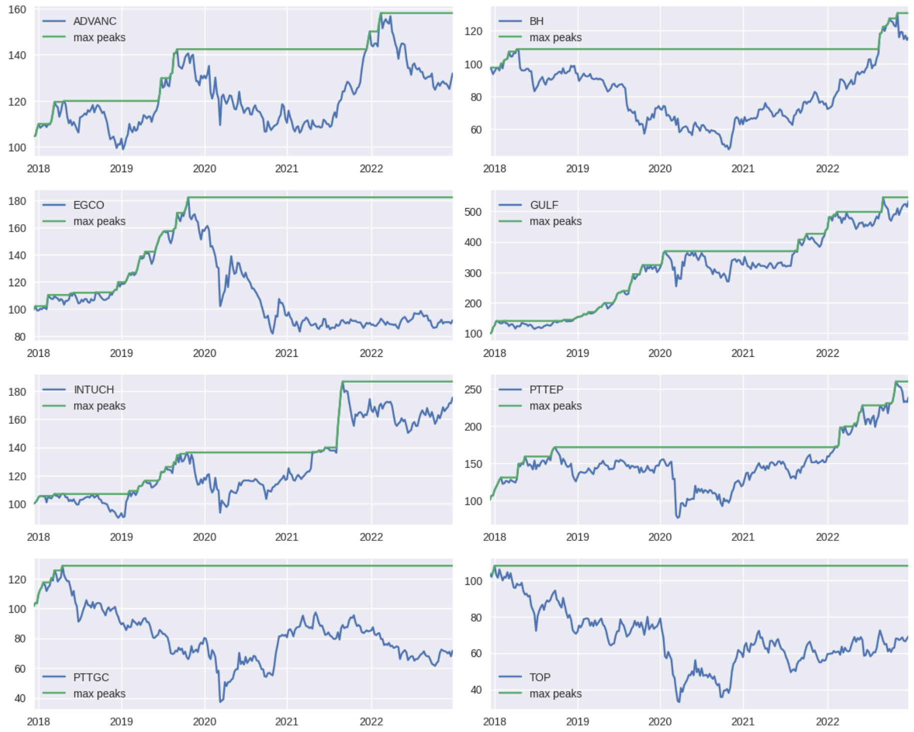
```
import pandaas pd
import plotly.graph_objects as go

# Function to normalize prices
def normalizedprice(df):
    """
    Normalizes the prices in a DataFrame so they all start at 1.0.
    This makes it easier to compare trends over time.
    """
    normalized_df = df / df.iloc[0]  # Divide each value by the first row's value
    return normalized_df

# Example DataFrame (replace this with your actual data)
# df = pd.read_csv('data/stock_data.csv', index_col='Date', parse_dates=True)

# Normalize the prices
normalized_price = normalizedprice(df)

# Create a Plotly figure
fig = go.Figure()

# Add a line for each column in the normalized DataFrame
for col in normalized_price.columns:
    fig.add_trace(go.Scatter(
        x=normalized_price.index,  # X-axis: Dates
        y=normalized_price[col],   # Y-axis: Normalized prices
        name=col,                  # Name of the line (e.g., stock name)
        mode='lines'               # Draw a line chart
    ))

# Update the layout of the chart
fig.update_layout(
    title='Normalized Stock Prices Over Time',  # Chart title
    xaxis_title='Date',                         # X-axis label
    yaxis_title='Normalized Price',             # Y-axis label
    showlegend=True                             # Show the legend
)

# Save the chart as an image
fig.write_image("figures/normalized_prices.png")

# Display the chart
fig.show()

# Sort and display the normalized prices on a specific date
sorted_prices = normalized_price.loc["2022-09-30"].sort_values(ascending=False)
print("Normalized Prices on 2022-09-30 (Sorted):")
print(sorted_prices)
```

Example 2: Comparison Bar Chart



```
import numpy as np
import matplotlib.pyplot as plt

# Example data (replace with your actual data)
# BestW_minvol = [0.1, 0.2, ...]
# BestW_MaxSharp = [0.1, 0.2, ...]
# BestW_MaxSortino = [0.1, 0.2, ...]
# BestW_Calmar = [0.1, 0.2, ...]
# BestW_cvar = [0.1, 0.2, ...]
# tickers = ['ADVANC', 'BH', 'COM7', ...]
# nTickers = len(tickers)

# Convert weights to percentages
Wminglobal = 100 * np.array(list(BestW_minvol))
Wmaxsharp = 100 * np.array(list(BestW_MaxSharp))
WSortiono = 100 * np.array(list(BestW_MaxSortino))
Wcalmar = 100 * np.array(list(BestW_Calmar))
Wcvar = 100 * np.array(list(BestW_cvar))

# Create the bar chart
X = np.arange(start=1, stop=nTickers + 1, step=1)
width = 0.18  # Adjusted width for better spacing

plt.figure(figsize=(12, 6))

# Plot each set of results
plt.bar(X - width, Wminglobal, width=width, color='teal', label='Min Volatility')
plt.bar(X, Wmaxsharp, width=width, color='darkorange', label='Max Sharpe')
plt.bar(X + width, WSortiono, width=width, color='blue', label='Max Sortino')
plt.bar(X + 2 * width, Wcalmar, width=width, color='green', label='Max Calmar')
plt.bar(X + 3 * width, Wcvar, width=width, color='red', label='Max CVaR')

# Add labels and title
plt.title('Portfolio Optimization Results Using Different Objectives')
plt.xlabel('Tickers')
plt.ylabel('Weights (%)')
plt.xticks(X, tickers)
plt.legend()
plt.tight_layout()  # Adjust layout to fit labels

# Save the chart
plt.savefig("figures/portfolio_optimization.png")

# Display the chart
plt.show()
```

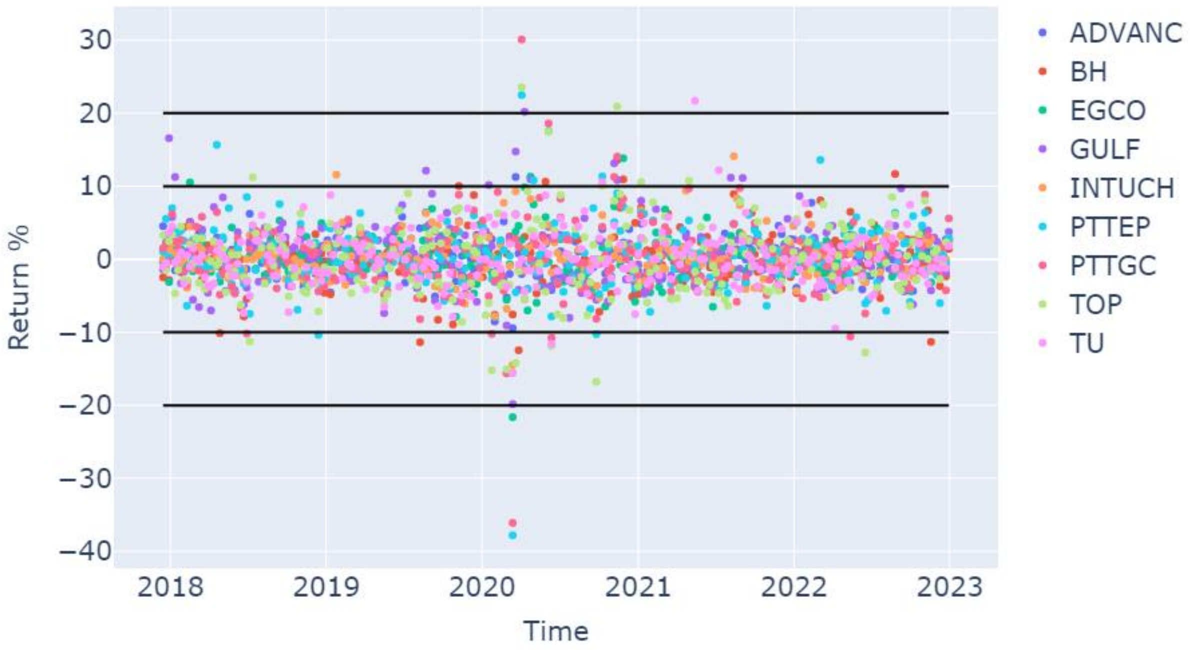
Example 3: Multiple line Charts in a frame (Max Drawdown)



```
import matplotlib.pyplot as plt

# Example data (replace with your actual data)
# wealth_index = pd.DataFrame(...)
# previous_peaks = pd.DataFrame(...)

# Define the columns to plot
columns = ['ADVANC', 'BH', 'EGCO', 'GULF', 'INTUCH', 'PTTEP', 'PTTGC', 'TOP', 'TU']

# Create subplots
fig, ax = plt.subplots(figsize=(12, 12), nrows=5, ncols=2)
ax = ax.flatten()

# Loop through the columns and plot each one
for i, col in enumerate(columns):
    wealth_index[col].plot(grid=True, ax=ax[i], legend=True, label="Wealth Index")
    previous_peaks[col].plot(grid=True, ax=ax[i], legend=True, label="Max Peaks")

# Save the chart
plt.tight_layout()
plt.savefig("figures/max_drawdown.png")
```

```
# Display the chart
plt.show()
```

Example 4: scatter plot



```
import plotly.graph_objects as go

# Example data (replace with your actual data)
# Returns = pd.DataFrame(...)

# Create a Plotly figure
fig = go.Figure()

# Add scatter plots for each column in Returns
for col in Returns.columns:
    fig.add_trace(go.Scatter(x=Returns.index, y=Returns[col] * 100, name=col, mode='markers'))

# Define horizontal lines at +5%, +10%, +20%, and their negative counterparts
horizontal_lines = [10, 20, -10, -20]  # Y-values for the horizontal lines

for y in horizontal_lines:
    fig.add_shape(
        type="line",
        x0=Returns.index.min(),  # Start of the line (minimum x-axis value)
        x1=Returns.index.max(),  # End of the line (maximum x-axis value)
        y0=y,  # Y-coordinate for start of the line
        y1=y,  # Y-coordinate for end of the line (same as start for horizontal line)
        line=dict(
            color="black",  # Line color
            width=2,        # Line width
            dash="solid"    # Line style (e.g., "solid", "dot", "dash")
        )
    )

# Update layout for axis titles, legend, and size
fig.update_layout(
    title='Returns Analysis',  # Title of the plot
    xaxis=dict(
        title='Time',
        title_font=dict(size=20),
        tickfont=dict(size=20)
    ),
    yaxis=dict(
        title='Return %',
        title_font=dict(size=20),
        tickfont=dict(size=20)
    ),
    legend=dict(
        font=dict(size=20)
    ),
    showlegend=True,
    width=900,  # Set the width of the plot
    height=600  # Set the height of the plot
)

# Save the chart
fig.write_image("figures/returns_analysis.png")

# Display the chart
fig.show()
```