ChatGPT

Finance_portfolio リポジトリ: 新機能実装ガイド

以下では、指定された新機能（1〜4）を実装するための具体的なコード改修手順を示します。それぞれ**機能ごとに**必要な変更内容と実装ステップを詳述し、実装者が直接作業に取り掛かれるように解説します。各ステップでは、既存コード構造を尊重しつつ必要な箇所のみ変更・追加する方針とします。

1. S&P500との Sharpe Ratio 比較機能の追加

目的: ユーザーが指定した期間において、自身のポートフォリオとS&P500指数の Sharpe Ratio を比較できる機能を実装します。Sharpe Ratio は日次リターンに基づき計算し、リスクフリーレートとして米国10年国債利回りを使用します [1] [2]。以下の手順でコードを改修してください。

1. **比較期間の選択UI追加:** Streamlitアプリのサイドバーまたはメイン画面に、ユーザーが比較期間を選択できる入力を追加します。例えば、ドロップダウン（`st.selectbox`）や日付入力（`st.date_input`を開始日・終了日で2つ）を用意し、「過去1年」「年初来（YTD）」「過去3年」などの選択肢や日付範囲を指定できるようにします。これによりユーザーはSharpe Ratioを比較する期間を任意に設定可能になります。

2. **S&P500指数データの取得:** 上記で選択された期間に対応するS&P500の価格データを取得します。Yahooファイナンスの ^GSPC （S&P500種指数）を yfinance ライブラリでダウンロードします。例えば次のように実装します [3] [4]。

```
import yfinance as yf
start_date = ...  # ユーザー指定の開始日
end_date = ...    # ユーザー指定の終了日
sp500_data = yf.download('^GSPC', start=start_date, end=end_date, interval='1d')
```

上記により、指定期間のS&P500日次終値（`sp500_data['Adj Close']`など）を取得できます。**注:** `yf.download`を用いると一度の呼び出しで時系列データを取得できます [3]。データ取得後、欠損値の処理（`dropna()`）やインデックスの調整を行ってください。

1. **リスクフリーレート(米国10年債利回り)の取得:** Sharpe Ratio計算に用いるリスクフリーレートとして、米国10年国債利回りを取得します。Yahooファイナンスでは **^TNX** が10年債利回り（CBOE 10 Year Treasury Note Yield）に対応します [2]。yfinanceで最新利回りを取得し、年率の割合（%）を得てください。例えば:

```
tnx = yf.Ticker('^TNX')
tnx_data = tnx.history(period='1d')
if not tnx_data.empty:
    current_yield = tnx_data['Close'][-1]  # 例: 4.30 (%)
    rf_rate = current_yield / 100  # 4.30% -> 0.0430
```

取得した `rf_rate` は年率無リスク利子率（約束収益率）です [5]。Sharpe Ratio計算では後述のように日次リターンに合わせて使用します。**参考:** Medium記事でもポートフォリオ分析で `^TNX` をデータに含め、10年債利回りを取得しています [6]。

1. **S&P500の Sharpe Ratio 計算:** S&P500指数の日次リターンからSharpe Ratioを算出します。計算手順は以下の通りです（ポートフォリオの場合も同様）：
2. **日次リターン計算:** 取得したS&P500の調整終値系列から日次リターンを算出します。例えば、`sp500_returns = sp500_data['Adj Close'].pct_change().dropna()` のように前日比%変化を求めます。必要に応じて `.pct_change()` ではなく対数リターン `np.log(today/prev)` を用いても構いません。
3. **平均リターンおよびボラティリティ計算:** 日次リターン系列から平均日次リターン(mean)と標準偏差(std)を計算します。標準偏差は日次ボラティリティに相当します。これらを年率換算するには、平均日次リターンを252倍、標準偏差を$\sqrt{252}$倍します（252営業日=1年と仮定） [1]。

4. **Sharpe Ratio算出:** Sharpe Ratio = (年間平均リターン - リスクフリーレート) / 年間ボラティリティ で求めます [7]。ここでリスクフリーレートには上で取得した `rf_rate` を使用します。例えば、年間平均リターン = `sp500_returns.mean() * 252`、年間ボラティリティ = `sp500_returns.std() * np.sqrt(252)` とし、Sharpeを計算します:

   ```
   annual_ret = sp500_returns.mean() * 252
   annual_vol = sp500_returns.std() * np.sqrt(252)
   sp500_sharpe = (annual_ret - rf_rate) / annual_vol
   ```

   ※Sharpe Ratioは投資の超過リターン（＝リターン - 無リスク率）をリスク（標準偏差）で割った指標です [8] [9]。

5. **ポートフォリオの Sharpe Ratio 計算:** 同様にユーザーポートフォリオのSharpe Ratioを**同じ期間**で算出します。ポートフォリオ全体の日次リターンを計算する必要があります。以下の方法で実装してください。

6. **価格データの取得:** ポートフォリオを構成する全銘柄の、指定期間における日次株価を取得します（既存コードで過去1年の株価を取得していますが、期間がユーザー指定になるため必要に応じ追加取得します）。yfinanceのマルチティッカー機能を使うと便利です（例: `yf.download(list_of_tickers, start=..., end=...)` で複数銘柄を一括ダウンロード可能 [3]）。**注意:** 米国株と日本株が混在する場合、それぞれ現地通貨建ての株価が取得されます。ポートフォリオ全体のリターンを一貫した通貨で計算するために、米国株の株価は対応する為替レートで円換算するか、逆に日本株をドル換算してください。為替レートについては、Yahooファイナンスの `USDJPY=X`（ドル円レート）を同期間で取得し、日次の為替レートを用いて各日の評価額を算出すると正確です。簡易的には現在の為替レートで全期間一定とみなす方法もありますが、正確な比較のためには可能であれば日次で為替適用してください。
7. **日次ポートフォリオ価値の算出:** 取得した各銘柄の日次株価データを用いて、ポートフォリオ全体の時系列価値を計算します。具体的には、ユーザー保有株数 (入力CSVから取得済) を各銘柄について固定し、各日の**各銘柄評価額** = `株価 × 保有株数`（必要に応じ為替換算）と計算します。全銘柄の評価額を合計することでその日のポートフォリオ総評価額となります [10] [11]。この計算を期間中のすべての日について行い、ポートフォリオ価値の時系列データ（DataFrame）を作成してください。既に実装されているUSD/JPYレートの取得 (`usd_jpy_rate`) を活用し、米国株は円換算、日本株はそのまま円建てで評価額を算出すると、一つの通貨（円）で総額を計算できます。

8. **日次リターンとSharpe算出:**　得られたポートフォリオ総評価額の時系列から、S&P500と同様に日次リターン系列を計算します（前日からの割合変化率）。次に、その日次リターン系列の平均と標準偏

差を求め、年率換算してSharpe Ratioを算出します（リスクフリーレートには同じ `rf_rate` を使用）。計算式は上記S&P500の場合と同様です。こうして求まった **`portfolio_sharpe`** が比較対象となるポートフォリオのSharpe Ratioです。

9. **Sharpe Ratio比較結果の表示:** S&P500のSharpe Ratio（`sp500_sharpe`）とポートフォリオのSharpe Ratio（`portfolio_sharpe`）をユーザーに分かりやすく提示します。Streamlitアプリ上で、例えば以下のような表示を追加してください。

10. テキストまたは数値指標: `st.metric` や `st.write` を用いて、選択期間における両者のSharpe Ratioを並べて表示します。例: 「**Sharpe Ratio (Portfolio)**: 0.85、 **Sharpe Ratio (S&P500)**: 0.65」のように明示します。また、期間も添えて「(期間: 2020-01-01 ～ 2023-01-01)」等と表示すると親切です。
11. 視覚的な比較: 2本の棒グラフによる比較チャートを描画することも検討してください（詳細は後述の「可視化の追加」参照）。小さな棒グラフでポートフォリオとS&P500のSharpeを示すと、一目で優劣が分かります。

以上により、ユーザーは指定期間における自身のポートフォリオのリスク調整後リターンが市場ベンチマークであるS&P500と比べてどうかを把握できるようになります。

2. Sharpe Ratio最大化のリバランス提案機能の実装

目的: ユーザーポートフォリオの資産配分を、Sharpe Ratioが最大となるように再構成する**リバランス案**を提示する機能を実装します。ここでは**各銘柄のSharpe Ratioに基づいてスコアリング**を行い、「高Sharpe・低ボラティリティ」の銘柄に高い比重を置く最適化ロジックを使用します。必要に応じてSharpeやボラティリティの影響度を調整できるパラメータ (べき乗係数 a, b) を導入します。以下の手順で実装してください。

1. **各銘柄のSharpe Ratio・ボラティリティ算出（再確認）:** ポートフォリオ内の各銘柄について、最新のSharpe Ratioとボラティリティ(年率換算)の値を取得します。これらは既存コードで **`portfolio_calculator.py`** 実行時に算出・出力されているものを利用できます（出力CSVの `sharpe` 列と `sigma` 列） [1]。実装時には、**機能1**で導入したリスクフリーレートの計算を考慮し、各銘柄Sharpeが最新の10年債利回りに基づいて計算されていることを確認してください（機能4で述べるように、固定4%から動的取得に改修します） [1] [2]。全銘柄の値が揃ったら、以下の最適化計算に移ります。

2. **Sharpe Ratioに基づくスコア計算:** 各銘柄に対し、Sharpe Ratioとボラティリティからリバランス用のスコアを計算します。スコアは次式で定義します。

$$ \text{score}_i = \frac{(\text{Sharpe}_i)^a}{(\sigma_i)^b} $$

ここで Sharpe_i は銘柄iのシャープレシオ、σ_i は銘柄iの年率ボラティリティです。a, b は調整パラメータで、デフォルトは1とします（後でユーザーが変更可能にします）。このスコアが高い銘柄ほど「Sharpeが高くリスク（ボラティリティ）が低い」ことを意味し、リバランス後のポートフォリオで比重を高めるべきと判断します。例えば a を2にすればSharpeの差をより強調し、b を大きくすればボラティリティの影響を強めて安全資産をより重視します。**注:** Sharpe Ratioが負の銘柄がある場合、スコアも負になってしまいます。その場合、通常はその資産への投資配分を0にするのが望ましいため、実装ではSharpeが非正(≦0)の銘柄のスコアは0と見なす等、負の値を扱わないようにしてください。負のSharpeの銘柄はリスクフリーレート未満のリターンしかないことを意味し、Sharpe最適化では除外すべきだからです。

1. **最適化後の目標配分の算出:** 全銘柄のスコアが計算できたら、それらを正規化して**目標アロケーション比率**を算出します。具体的には各銘柄の重み w_i を: $$ w_i = \frac{\text{score}_i}{\sum_j \text{score}_j} $$ と計算します。これにより全銘柄の w_i の総和は1（100%）になります。ただ

し、scoreが全て0になってしまうケース（極端にSharpeが低いポートフォリオ）は通常想定されませんが、その場合は等分配など適切な処理をしてください。こうして得られた {銘柄: 新割合} の一覧が**Sharpe最適化後の目標ポートフォリオ構成**です。

2. **調整パラメータ a, b の導入:** スコア計算式の a, b をユーザーが調整できるようにします。StreamlitアプリのUIにスライダーや数値入力を追加し、リアルタイムに `a`, `b` の値を変更できるようにしましょう。例えば:

3. `a = st.slider("Sharpe 指数の強調度 a", min_value=0.5, max_value=3.0, value=1.0, step=0.5)`

4. `b = st.slider("ボラティリティの強調度 b", min_value=0.5, max_value=3.0, value=1.0, step=0.5)` といった具合に実装します。デフォルトは1.0とし、0.5刻みや1.0刻みで調整できるようにすると良いでしょう。これにより、高Sharpe銘柄をどの程度優遇するか、低ボラ銘柄をどの程度重視するかをユーザーがインタラクティブに調節できます。スライダー操作時には上記ステップ2〜3の計算をリアルタイムに再実行し、新しい推奨ポートフォリオ配分を計算します。

5. **リバランス提案（売買指示）の計算:** 現在のポートフォリオから目標ポートフォリオへ移行するには各銘柄をどれだけ売買すべきか計算します。既存実装の「Required Trades（必要な売買）」機能（効率的フロンティアでのリバランス提案）を参考に、Sharpe最適化の場合も同様の出力を行います [12]。計算手順:

6. 現在のポートフォリオの総評価額を計算します（円建て総額。既に `portfolio_calculator.py` の出力に各銘柄の `value_jp` があり、全銘柄合計が総額となります）。
7. **目標配分における各銘柄のターゲット評価額** = 総評価額 × w_i（上で計算した新割合）。
8. 各銘柄について、ターゲット評価額と現在評価額との差分を求めます。この差がその銘柄で売買すべき金額になります。例えば、ある銘柄Aの現在評価額が100万円で新割合が20%（新ターゲット評価額200万円）なら、+100万円分の買い増しが必要です。逆に新割合が5%（ターゲット50万円）なら、-50万円、すなわち50万円分を売却する提案になります。

9. **具体的な株数への変換:** 上記金額の差分をそれぞれの銘柄の株数に換算します。銘柄ごとの現在株価と通貨を考慮してください。既存のUSD/JPYレート（`usd_jpy_rate`）を使用し、米国株は円換算した株価、または日本株はそのまま円株価を用いて、必要売買額÷現在株価で株数差分を計算します。例えば:

```
price_jpy = price_usd * usd_jpy_rate if 米国株 else price_yen
diff_value = target_value_jpy - current_value_jpy
diff_shares = diff_value / price_jpy
```

   得られた `diff_shares` が**追加購入すべき株数**（正の値なら買い、負の値なら売り）です。必要に応じて整数に丸めます（端数株は買えないため）。この計算は既存の効率的フロンティアのリバランス計算ロジックと類似しています [12]。現在保有株数は入力CSVや計算結果データから取得できます。
   注: 日本株と米国株を両方含む場合、それぞれ異なる通貨で計算する代わりに円に統一して計算していますので、diff_shares算出時に価格も円建てに統一する点に留意してください。

10. **リバランス提案結果の表示:** 計算した各銘柄の売買指示（何株を売買すべきか、または金額ベースでいくら売買か）をユーザーに提示します。分かりやすい形式で一覧表示してください。以下のいずれかの方法が考えられます。

11. **テーブル形式:** 銘柄コード、現在保有額(比率)、目標比率、目標額、差分(金額/株数) といった列を持つデータフレームを作成し、`st.table` や `st.dataframe` で表示します。例えば「AAPL | 現在10% | 目標15% | +5% (約+XX万円, +YY株)」のような行を並べます。
12. **文章形式:** `st.write` で銘柄ごとに「**AAPL**: 約YY株を新規購入 (＋XX万円)」や「**7203.T**: ZZ株を売却 (-YY万円)」のような文章を箇条書き表示します。
    いずれにせよ、ユーザーが**具体的なアクション**を理解できるよう、「買い増し or 売却」「おおよその株数または金額」を明示してください。既存の効率的フロンティア機能では「Required Trades」として類似の出力を行っているはずなので、それに倣ったUI配置・デザインにすると統一感があります [12]。

以上の手順により、ユーザーはSharpe　Ratioに着目したポートフォリオ最適化の結果（新しい配分比率）と、その実現のための具体的な売買提案を得られます。これにより、自身のポートフォリオをリスク対効果の高い構成に改善するインサイトを提供できます。

3. Sharpe Optimized タブの新設とUI統合

目的:　上記で実装したSharpe最適化によるリバランス提案機能を、既存の「Conservative / Balanced / Aggressive」プロファイルとは独立した新しいタブ（またはセクション）としてアプリに組み込みます。ユーザーが他の分析ビューと切り替えながらこのSharpe Optimized提案を見ることができるようにします。以下の手順でUI側の改修を行ってください。

1. **新タブ/セクションの追加:** Streamlitのマルチページまたはタブ機能を利用し、**Sharpe Optimized** 用の独立したビューを作成します。具体的な実装方法は以下のいずれかになります。
2. **`st.tabs` を使用:** 例えば `tab1, tab2 = st.tabs(["Efficient Frontier", "Sharpe Optimized"])` のように既存の効率的フロンティアビューとSharpe最適化ビューをタブで分割します。そして `with tab2:` ブロック内にSharpe OptimizedのUI要素を配置します。これにより画面上部に2つのタブが表示され、ユーザーは「Sharpe Optimized」タブをクリックして本機能の画面に切り替えられます。

3. **プロファイル選択のラジオボタン/セレクトボックスを使用:**　もし既存コードでリスクプロファイル（Conservative/Balanced/Aggressive）を選択するUIがある場合、その選択肢の中に `"Sharpe Optimized"` を追加します [13]。例えば `profile = st.radio("Select Profile", ["Conservative", "Balanced", "Aggressive", "Sharpe Optimized"])` とし、選択肢に応じて表示内容を分岐させます。
   既存コード内にプロファイル概念がまだ実装されていない場合は、前者のタブ方式で実装する方が簡潔です。

4. **Sharpe Optimizedタブ内のUI構築:** 新タブ内に、本機能に必要なインタラクティブ要素と出力を配置します。手順2で実装した**調整パラメータスライダー (a, b)** や、計算結果の表示（新配分比率一覧、リバランス売買提案）をこのタブ内にまとめて配置してください。具体的には:

5. 見出しや説明文: `st.header("Sharpe Optimized Portfolio")` のようなタイトルを付け、併せて本タブの機能説明を短文で記載すると親切です。「各銘柄のSharpeレシオに基づいてポートフォリオを最適化した場合の提案配分と売買プランを表示します。」など。
6. スライダーUI: 機能2のステップ4で導入した a, b パラメータ調整用の `st.slider` をこの中に配置します。ユーザーが操作しやすいようにレイアウトを整えてください（必要なら `st.columns` で横並びにする等）。
7. **「Update」ボタンの要否:** スライダー操作に応じて自動的に再計算・更新させる場合、特別なボタンは不要です。ユーザーが頻繁にスライダーをいじると毎回計算が走るためパフォーマンスが気になる場合は、「Recalculate」などのボタンを設けて明示的に再計算させる方法もあります。計算量自体は大きくないので自動更新でも問題ない想定ですが、必要に応じて検討してください。

8. 出力内容: 機能2で計算した**新配分比率**および**売買提案**の結果を表示します。これらはスライダー操作のたびに動的に更新されるようにします。テーブルまたは文章で一覧表示する実装を、このタブ内に組み込みます。

9. **Sharpe Ratio比較グラフの追加 (リバランス前後):** ユーザーポートフォリオの現状Sharpe Ratioと、Sharpe最適化後ポートフォリオのSharpe Ratioを比較する可視化をこのタブ内に追加します（ユーザーリクエストの「可視化の追加」に対応） [11] 。実装手順:

10. **Sharpe Ratioの再計算:** 現在ポートフォリオおよびSharpe最適化後ポートフォリオのSharpe Ratioを計算します。現状ポートフォリオのSharpeは**機能1**ですでに算出済みの値を使えます。同様に、新ポートフォリオのSharpe Ratioを計算します。計算には各銘柄の期待リターン・ボラティリティと新ポートフォリオの重量を用います。可能であれば銘柄間の相関も考慮して**正確なポートフォリオボラティリティ**を算出してください（既存の相関行列CSV [14] や効率的フロンティアの計算コードが利用できます）。簡易的には、各銘柄の期待超過リターン = Sharpe_i * σ_i と仮定し、ポートフォリオ期待超過リターンを $\sum w_i \times (\text{Sharpe}i \times \sigma_i)$、ポートフォリオボラティリティを $\sqrt{\sum$ を無視すると過大/過小評価につながるので、可能であれば考慮してください。既存の } w_i w_j \sigma_i \sigma_j \rho_{i,j}}$ で求め、Sharpe = (期待リターン - rf) / ボラティリティ を算出します [8] [9] 。※相関 $\rho_{i,j `efficient_frontier.py` に類似の計算実装がある場合、それを参考にするとよいでしょう。
11. **棒グラフの描画:** StreamlitにPlotlyを用いて棒グラフを描画します [11] 。x軸に「Before (現在ポートフォリオ)」と「After (Sharpe Optimized)」の2区分をとり、y軸に各ポートフォリオのSharpe Ratioをプロットします。それぞれ1本のバーで表現し、グラフのタイトルを「Sharpe Ratio Before vs After Rebalance」のように設定してください。例えば:

```
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Bar(name='Sharpe Ratio',
                     x=['Current Portfolio', 'Sharpe Optimized'],
                     y=[current_sharpe, optimized_sharpe]))
fig.update_layout(title="Sharpe Ratio (Before vs After)",
                  yaxis_title="Sharpe Ratio")
st.plotly_chart(fig)
```

といったコードで棒グラフを生成できます。上記は一例ですので、実際には計算した `current_sharpe` と `optimized_sharpe` の値を用いてください。グラフ中には目安線としてSharpe=1を引くことも検討できます [11] （例えば `fig.add_hline(y=1, line_dash="dash", line_color="red")` ）。Sharpe=1は一般に優良ポートフォリオの目安とされるため、視覚的指標になります。

12. **表示場所:** このグラフはSharpe Optimizedタブ内の最後部に配置し、リバランス前後でポートフォリオの効率性がどう変化するかを直感的に示します。例えば「Sharpe Ratio (現ポートフォリオ) vs (Sharpe最適化ポートフォリオ)」というサブヘッダを付けてグラフを表示すると分かりやすいでしょう。

13. **他のビューとの統合確認:** 新設したSharpe Optimizedタブが他の機能（例えばCombined ViewやEfficient Frontierビュー）と独立して動作することを確認します。他の部分への影響として考えられる点は以下です。

14. 「Update Data」ボタンを押した際の挙動: 全体データを更新する処理にSharpe Optimized関連のデータも必要なら組み込みます。ただし、本機能の計算は主に既存の出力結果やリアルタイム取得

データを使うため、特別な更新処理は不要な可能性があります。Update Data押下でポートフォリオデータが更新されたら、新配分の計算も最新データに基づき自動更新されることを確認してください。

15. **キャッシュの活用:** 既存コードで `st.cache_data` や `st.cache` を用いてAPI取得結果をキャッシュしている場合、その仕組みをSharpe Optimizedタブでも活用してください。例えば、S&P500や^TNXデータの取得は頻繁に行うとAPI負荷になるため、一度取得したらしばらくキャッシュすることが望ましいです（10年債利回りは日次で更新されますが、数分間に何度も取り直す必要はありません）。実装上は、データ取得処理にデコレータを付与するか、既存のキャッシュ機構に組み込む形で対応してください。

以上で、Sharpe Optimized機能が他の分析プロファイルと並列に提供され、ユーザーはタブ切り替えによって本機能を利用できるようになります。他のビュー（Conservative/Balanced/AggressiveやEfficient Frontier）とは**独立**しているため、相互に干渉せずUI切り替えで比較・検討できるようになります。

4. 既存コード構造への適合と関連箇所の修正

目的: 新機能を追加するにあたり、既存コードベースの整合性を保つよう必要な箇所を修正・拡張します。特にポートフォリオ評価計算部分、リスク分析部分、データ更新処理部分で本機能に関連する変更点を整理します。

- **リスクフリーレート計算の中央管理:** 既存コードではSharpe Ratio算出時のリスクフリーレートを**固定4%**としていました [1]。これを本実装で**動的取得した米国10年債利回り**に置き換えます。具体的には、`portfolio_calculator.py` 内でSharpeを計算している箇所を修正し、`rf_rate` として上記**機能1**で取得した `current_yield` を使用してください。例えば:

```
# 修正前（擬似コード）
sharpe = (annual_return - 0.04) / annual_vol
# 修正後
rf_rate = get_10yr_yield()  # ^TNXから年率利回り(例:0.043)
sharpe = (annual_return - rf_rate) / annual_vol
```

  とします [2]。この `get_10yr_yield()` 関数はyfinanceを用いて^TNXから利回りを取得する処理で、実装が重複しないよう適切な場所に共通化してください（例えば `portfolio_calculator.py` 冒頭やユーティリティ関数として定義）。**注意:** 日本株ポートフォリオ計算(`portfolio_calculator_jp.py`)が別途存在する場合、同様の修正を適用してください。

- **キャッシュデータの更新:** リスクフリーレートを固定値から動的取得に変えたことで、キャッシュしているSharpe Ratio値との不整合が生じないようにします。既存実装では銘柄ごとのボラティリティ・Sharpeを24時間キャッシュしています [15]。新実装では、利回りの変化によってSharpe計算結果が変わり得るため、キャッシュのキーにリスクフリー利率も含めるか、またはUpdate Data時にキャッシュを無視するようにします。例えば、`ticker_cache.json` に格納するSharpe値は利回りが変動したら再計算が必要です。シンプルな対応としては、**Update Data**（フルリフレッシュ）時にキャッシュをクリアすることで最新利回りを反映できます。「Force full refresh」オプションが既に実装されているので [16]、それを利用してキャッシュを無視させる流れにSharpe計算も従わせてください。

- **データ更新処理との統合:** 新機能で使用するS&P500指数や^TNX利回りは、基本的にユーザー操作時（期間選択やタブ選択時）にオンデマンドで取得する設計です。そのため「Update Data」ボタンを押した際に必ずしも取得し直す必要はありません。ただし、アプリ起動直後にSharpe Optimizedタブを開いた場合など、必要なデータがまだ無い場合があります。堅牢性のために、**Update Data処理完**

了後にSharpe　Optimized表示内容も最新データで更新されるようにしてください。例えばUpdate Dataボタンハンドラ内でグローバルなデータ状態を更新している場合、S&P500や^TNXのデータもリセットするか、次回表示時に再取得させる措置を講じます。
実装方法の一例として、Update　Data実行時に `st.session_state` にフラグ（例: `session_state['data_updated'] = True`）をセットし、Sharpe Optimizedタブ側でそのフラグを検知したら内部のキャッシュをクリアして再計算する、などが考えられます。もしくはUpdate　Dataボタンが押されたらSharpe　Optimized用のデータ取得関数に `force_refresh=True` を渡し、キャッシュ無視で再取得する方法もあります。いずれにせよ、ユーザーがデータ更新後に古い指標を見続けることのないよう配慮してください。

- **コード構造の分離/共通化:** Sharpe Optimizedの計算処理は、可能であればUIロジックから分離して関数化しておくと保守性が高まります。例えば `sharpe_optimized.py` という新モジュールを作成し、以下のような関数を実装することも検討してください（必須ではありませんが推奨されます）:

- `calculate_sharpe_scores(portfolio_data, a=1, b=1) -> dict[ticker, score]`: 銘柄ごとのSharpeとσからscoreを計算し返す関数
- `calculate_target_weights(scores: dict) -> dict[ticker, float]`: スコア辞書から正規化された重みを返す関数

- `calculate_trade_plan(current_portfolio, target_weights) -> list of trades`: 現在保有株数・評価額と目標重みから、銘柄ごとの売買数量を算出する関数　これらを実装し、Streamlit側ではそれらを呼び出して結果を表示するだけにすると、UIとロジックが分離され読みやすくなります。既存の `efficient_frontier.py` で類似の構造が取られている場合は、それにならってSharpe　Optimized用のコードも整理してください。

- **ドキュメントの更新:** コード実装ではありませんが、READMEやユーザー向け説明がある場合、新機能に関する記述を追加することも忘れないでください。例えば、「Sharpe Optimized」タブでは何ができるのか、Sharpe　Ratio比較機能の使い方などをREADMEに追記すると、利用者に親切です。特に**米国10年債利回りをリスクフリーに使用する**点は重要な変更なので、その旨を明記しておきましょう（「シャープレシオ計算におけるリスクフリーレートを動的に米国債利回りに変更しました」等）[1]。

以上の対応により、現在のコード構造を崩さず新機能を組み込むことができます。各変更箇所で既存の関数・クラスを再利用し、重複コードを避けるよう留意してください。また、実装後はユニットテスト（`test_efficient_frontier.py`等）や手動テストを行い、新旧機能が期待通り動作することを確認してください。必要に応じてテストコードも拡充し、Sharpe　Optimized機能の検証を自動化することをおすすめします。

5. 可視化の追加: Sharpe Ratio比較グラフ (リバランス前後)

(※本項目は機能3の中で既に触れましたが、ユーザーからのリクエストに基づき改めて整理します)

目的:　リバランス前後でポートフォリオのSharpe　Ratioがどう向上するかを視覚的に示すグラフを提供します。実装方法は機能3のステップ3で詳述した通りですが、以下に要点をまとめます。

- **計測:** 現在ポートフォリオおよびSharpe Optimizedポートフォリオの Sharpe Ratioを算出します。計算には10年債利回りをリスクフリー率として使用し、期間は基本的に直近1年など最新データに基づきます（ユーザー指定期間ではなく常に最新値で比較するのが分かりやすいでしょう）。Sharpe最適化ポートフォリオの期待Sharpeは、各銘柄の期待リターンと共分散から計算します。簡便的に相関を

無視する場合、結果は概算になりますが、ユーザーに「概算である」断りを入れるか、可能なら相関考慮してください。

- **描画:** Plotlyのbar chartを使い2本の棒を描画します（現在 vs 最適化後）。上記機能3ステップ3のコード例を参照してください。グラフの凡例やタイトルにより、どちらが現状でどちらが提案後か明確にラベル付けします。[11]に示されるようにSharpe=1の基準線を引くのも良い演出です。グラフはStreamlit上で `st.plotly_chart(fig)` により表示します。

- **配置:** このグラフはSharpe Optimizedタブ内の適切な位置（提案内容の下部）に配置し、文章による解説を添えると親切です。例: 「グラフはリバランス前後のポートフォリオSharpe Ratioを比較したものです。提案後のSharpe Ratioが現状より高くなっていることを確認できます。」等。

この可視化により、ユーザーは提案によるリスク調整後リターンの改善度合いを直感的に理解できます。数値だけでなく視覚的に示すことで、Sharpe Ratioの向上がユーザーの意思決定を後押しするでしょう。

以上、GitHubリポジトリ **SenaTaka/Finance_portfolio** に対する新機能の具体的な実装指示をまとめました[12][2]。今回追加するSharpe Ratio比較機能およびSharpe Optimizedリバランス提案機能により、ポートフォリオ分析ツールの有用性が一段と向上するはずです。実装の際は各ステップの目的を踏まえ、既存コードとの一貫性に注意して進めてください。必要に応じて参考文献や既存コード例[1][4]も参照しながら、機能を完成させましょう。

---

[1][12][14][15][16] GitHub - SenaTaka/Finance_portfolio
https://github.com/SenaTaka/Finance_portfolio

[2][5] GitHub - marcelomijas/two_stocks_portfolio: Two stocks minimum-variance portfolio and optimal portfolio with Python.
https://github.com/marcelomijas/two_stocks_portfolio

[3][4][6] A Step-by-Step Guide to Portfolio Rebalancing with Python | by Javier Castillo Guillen | Medium
https://medium.com/@javier.castillo.guillen/a-step-by-step-guide-to-portfolio-rebalancing-with-python-2750b3f88793

[7] Sharpe Ratio Explained: Formula, Calculation in Excel & Python, and Examples
https://blog.quantinsti.com/sharpe-ratio-applications-algorithmic-trading/

[8][9] Sharpe Ratio: Definition, Formula, and Examples
https://www.investopedia.com/terms/s/sharperatio.asp

[10][11] Python Streamlit in Practice; A Use-Case of Visualizing Stock Data | by Ryo Koyajima / 小矢島 諒 | Medium
https://koyaaarr.medium.com/python-streamlit-in-practice-a-use-case-of-visualizing-stock-data-20ec1e1c8478

[13] AI Robo-Advisor - Streamlit Community Cloud
https://adredes-weslee-using-artificial-intelligenc-dashboardapp-juewyb.streamlit.app/