Documentation de la page « Portfolio Visualizer »

Gabriel Pézennec

29 décembre 2024

Table des matières

1 Introduction

Cette page **Portfolio Visualizer** vise à aider l'utilisateur à :

- **Créer** un portefeuille d'investissement personnalisé en choisissant les *tickers* et en leur attribuant un poids.
- **Filtrer** les entreprises par indice boursier (ex. S&P 500, CAC 40, etc.).
- **Inclure** des actifs provenant de sa *watchlist* personnelle.
- **Contrôler** la répartition des poids (somme à 100%) et éviter les doublons.
- **Visualiser** la performance historique du portefeuille, sa répartition par industrie, ainsi que ses principales métriques (rendement, volatilité, ratio de Sharpe...).
- **Sauvegarder** le portefeuille (dans `st.session_state`) pour exploitation ultérieure (ex. Portfolio Optimizer).

Grâce à `streamlit` et `yfinance`, la page propose une interface **interactive** et **intuitive** pour ajouter des actifs et visualiser aussitôt les indicateurs clés.

2 Explications Détaillées

2.1 Importations

```
import pandas as pd
import streamlit as st
import yfinance as yf
import plotly.graph_objs as go
```

```
import numpy as np
from utils.graph_utils import plot_performance, plot_pie
```

- `pandas` : Manipulation de données tabulaires. Ici, utile pour la lecture/écriture de CSV, la création de DataFrames, etc.
- `streamlit` : Bibliothèque permettant de créer des applications web interactives (widgets, session state).
- `yfinance` : Récupération des données de marché (cours, informations sur les entreprises).
- `plotly.graph_objs` : Génération de graphiques interactifs (tables, pie charts, etc.).
- `numpy` : Calcul numérique (opérations vectorisées, statistiques).
- `plot_performance, plot_pie` : Fonctions utilitaires (définies dans `utils/graph_utils.py`) pour l'affichage du *cumul de performance* et des camemberts (`pie charts`).

2.2 `get_company_name(ticker)`

```
def get_company_name(ticker):
    try:
        company = yf.Ticker(ticker).info.get('longName', 'N/A')
    except:
        company = 'N/A'
    return company
```

- Reçoit un `ticker` (ex. `"AAPL"`).
- Tente de récupérer son *longName* (ex. : « Apple Inc. ») via `yfinance`.
- Retourne « N/A »si indisponible ou en cas d'erreur.

2.3 `calculate_portfolio_performance(. . .)`

```
def calculate_portfolio_performance(tickers, weights, period='1y'):
    data = yf.download(tickers, period=period)['Close']
    if isinstance(data, pd.Series):
        data = data.to_frame()  # CAS OU IL Y A UN SEUL TICKER
    returns = data.pct_change().dropna()
    weighted_returns = returns.multiply(weights, axis=1)
    portfolio_returns = weighted_returns.sum(axis=1)
    portfolio_cumulative = (1 + portfolio_returns).cumprod()
    return portfolio_cumulative, portfolio_returns
```

- `yf.download(..., period='1y')` : Télécharge les prix de clôture (`Close`) sur 1 an (modifiable).
- `data.to_frame()` : Si un seul `ticker`, `yfinance` renvoie une *Series*, on convertit pour rester cohérent.
- `returns = data.pct_change().dropna()` : Calcule le rendement quotidien (pour chaque ticker).
- `weighted_returns = returns.multiply(weights, axis=1)` : Applique les poids du portefeuille à chaque colonne (ticker).
- `portfolio_returns` : Somme de ces rendements pondérés, pour obtenir le rendement global du portefeuille.
- `portfolio_cumulative` : Produit cumulatif $(1 + r_t)$, utile pour tracer la performance historique.
- **Retourne** :
  - `portfolio_cumulative` (traçable en courbe de performance),
  - `portfolio_returns` (pour calculer volatilité, rendement moyen, etc.).

2.4 Initialisation des `session_state`

```
if 'tickers' not in st.session_state:
    st.session_state.tickers = ['']
if 'weights' not in st.session_state:
    st.session_state.weights = [0.0]
if 'watchlist' not in st.session_state:
    st.session_state.watchlist = []
if 'current_index' not in st.session_state:
    st.session_state.current_index = 0
```

- **Objectif** : S'assurer que certaines listes (`tickers`, `weights`, `watchlist`) sont prêtes à l'emploi.
- `st.session_state` : Stocke des variables persistantes pendant la session Streamlit.
- `tickers`, `weights` : On commence avec un seul actif « vide »(`""`) et un poids 0.0.
- `current_index` : Peut servir pour la navigation, initialisé à 0.

2.5 Fonctions `add_asset` et `remove_asset`

```
def add_asset():
    st.session_state.tickers.append('')
    st.session_state.weights.append(0.0)

def remove_asset(index):
    del st.session_state.tickers[index]
    del st.session_state.weights[index]
```

- **add_asset()** : Ajoute une ligne vide (`""`) et un poids nul pour un *nouvel* actif.
- **remove_asset(index)** : Supprime l'actif (et son poids) à l'indice donné.
- **But** : Permettre d'ajouter/supprimer dynamiquement des lignes d'actifs dans l'interface.

2.6 Injection HTML/CSS/JS pour AOS (Animation on Scroll)

```
st.markdown("""
<link href="https://unpkg.com/aos@2.3.4/dist/aos.css" rel="stylesheet">
<script src="https://unpkg.com/aos@2.3.4/dist/aos.js"></script>
<script>
  AOS.init({
    duration: 1200,
  });
</script>
""", unsafe_allow_html=True)
```

- Ajoute des liens externes pour gérer des animations (`AOS`), animant potentiellement l'apparition d'éléments sur la page.
- `unsafe_allow_html=True` : Permet l'insertion directe de ressources HTML/JS.

2.7 Lecture du `df_companies`

```
df_companies = pd.read_csv("/home/onyxia/work/Pi.Squared.Trading/Data/data_pisquared.csv")
```

- `df_companies` : Contient les `Company`, `Ticker`, `Ind` (indice), etc.
- Permet de filtrer/choisir les entreprises lors de la construction du portefeuille.

2.8 main()

```
def main():
    #CSS pour ajuster la largeur de la zone de contenu
    st.markdown(
        """
        <style>
        div.block-container {
            max-width: 90%;
            margin: auto;
            padding: 1rem;
        }
        </style>
        """,
        unsafe_allow_html=True,
    )
    st.title("Portfolio visualizer")
```

— **Mise en page Streamlit** : `max-width: 90%`, alignement, etc.
— `st.title("Portfolio visualizer")` : Titre en haut de la page.

```
    description = "Portfolio Visualizer permet aux utilisateurs de crer ..."
    justified_description = f"""
    <div style='text-align: justify; text-justify: inter-word;'>
        {description}
    </div>
    """
    st.markdown(justified_description, unsafe_allow_html=True)
```

— `description` : Présentation générale.
— Alignement justifié du texte pour un rendu plus professionnel.

```
    st.write("")
    st.write("Veuillez entrer les actifs que vous souhaitez ...")

    if 'portfolio' not in st.session_state:
        st.session_state.portfolio = pd.DataFrame(columns=['Actions', 'Nom de l\'Entreprise', 'Poids (%)']
```

— **Message d'intro** pour expliquer comment procéder.
— `st.session_state.portfolio` : DataFrame final regroupant les champs `Actions`, `Nom de l'Entreprise`, `Poids (%)`.

```
    #Filtre d'indice
    st.header("Filtrer par Indice")
    indices = df_companies['Ind'].unique().tolist()
    selected_indices = st.multiselect(
        "Choisissez un ou plusieurs indices",
        options=indices,
        default=indices
    )

    if selected_indices:
        filtered_companies = df_companies[df_companies['Ind'].isin(selected_indices)]
    else:
        filtered_companies = df_companies.copy()

    if filtered_companies.empty:
        st.warning("Aucune entreprise trouve pour les indices slectionns.")
```

— `st.multiselect` : permet de sélectionner plusieurs indices (ex. S&P 500, CAC40, etc.).
— `filtered_companies` : DataFrame restreint aux indices choisis.
— `st.warning` : avertit si aucun résultat n'est trouvé.

```
#Fonctions callback pour ajouter/supprimer un actif
def remove_asset_callback(index):
    if 0 <= index < len(st.session_state.tickers):
        remove_asset(index)

def add_asset_callback():
    add_asset()

st.button(" ␣Ajouter␣un␣actif", on_click=add_asset_callback)
```

— **Fonctions callback** : Les boutons Streamlit acceptent des « on_click » paramétrés.
— `remove_asset_callback` : supprime un actif donné.
— `add_asset_callback` : ajoute un actif vide.
— `st.button(" Ajouter un actif", ...)` : Permet à l'utilisateur d'ajouter dynamiquement des lignes pour insérer de nouveaux actifs.

```
for i in range(len(st.session_state.tickers)):
    ticker = st.session_state.tickers[i]
    matching_rows = filtered_companies[filtered_companies["Ticker"] == ticker]

    default_index = 0
    companies_list = [""] + filtered_companies["Company"].tolist()
    if not matching_rows.empty:
        company = matching_rows["Company"].iloc[0]
        if company in companies_list:
            default_index = companies_list.index(company)

    cols = st.columns([2, 2, 1])

    with cols[0]:
        selected_company = st.selectbox(
            f"Entreprise␣{i+1}",
            options=companies_list,
            index=default_index,
            key=f"company_{i}"
        )
        if selected_company:
            new_ticker = filtered_companies.loc[
                filtered_companies["Company"] == selected_company, "Ticker"
            ].values[0]
            st.session_state.tickers[i] = new_ticker

    with cols[1]:
        weight = st.number_input(
            f"Poids␣{i+1}␣(%)",
            key=f"weight_{i}",
            min_value=0.0,
            max_value=100.0,
            value=st.session_state.weights[i],
            step=0.1
        )
        st.session_state.weights[i] = weight

    with cols[2]:
        st.button("      ␣Supprimer", key=f"remove_{i}", on_click=remove_asset_callback, args=(i,))
```

— **Boucle** : Pour chaque actif indexé `i`, on construit une ligne d'interface.
— **Sélection d'entreprise** :
  — `selected_company = st.selectbox(...)` : L'utilisateur choisit « Apple Inc. », « Amazon », etc.
  — On retrouve le *ticker* (`new_ticker`) dans `filtered_companies` pour mettre à jour `st.session_state.tickers[i]`.
— **Poids** (%) :
  — `st.number_input(...)` : Reçoit la valeur d'origine `st.session_state.weights[i]`.
  — `min_value=0.0, max_value=100.0, step=0.1` : Contrôle la plage de saisie.
— **Bouton Supprimer** :
  — `st.button(" Supprimer", on_click=..., args=(i,))` : Appelle `remove_asset_callback` pour l'actif `i`.

```
st.write("### Ajouter des actions depuis votre Watchlist")
if st.session_state.watchlist:
    watchlist_selection = st.selectbox("Slectionnez une action de votre watchlist :", st.session_state

    def add_from_watchlist():
        selected_ticker = watchlist_selection.upper()
        if selected_ticker in st.session_state.tickers:
            st.warning(f"L'action **{selected_ticker}** est dj dans votre portefeuille.")
        else:
            st.session_state.tickers.append(selected_ticker)
            st.session_state.weights.append(0.0)
            st.success(f"L'action **{selected_ticker}** a t ajoute. Veuillez dfinir son poids.")

    st.button(" Ajouter depuis la Watchlist", on_click=add_from_watchlist)
else:
    st.info("Votre watchlist est vide...")
```

— **Ajout direct depuis la watchlist** : L'utilisateur n'a plus besoin de chercher l'entreprise dans le `selectbox`.
— Vérification : évite les doublons.
— **Succès** : On demande ensuite de spécifier le poids.

```
total_weight = sum(st.session_state.weights)
st.markdown(f"**Poids Total Actuel : {total_weight:.2f}%**")
if not np.isclose(total_weight, 100.0):
    st.warning("   La somme des poids doit tre gale   100%.")
else:
    st.success(" La somme des poids est gale  100%.")
```

— Vérifie la cohérence globale (somme à 100%).
— `np.isclose(..., 100.0)` : Contrôle d'une petite marge d'erreur numérique.

```
if st.button(" Valider le Portefeuille"):
    tickers = []
    weights = []
    for i in range(len(st.session_state.tickers)):
        ticker = st.session_state.tickers[i].strip().upper()
        weight = st.session_state.weights[i]
        if ticker:
            tickers.append(ticker)
            weights.append(weight)

    st.session_state.tickers = tickers
```

```
        st.session_state.weights = weights

        duplicates = set([ticker for ticker in tickers if tickers.count(ticker) > 1])
        if duplicates:
            st.error(f"Les␣tickers␣suivants␣sont␣prsents␣plusieurs␣fois␣:␣{',␣'.join(duplicates)}.")
            st.stop()

        df = pd.DataFrame({
            'Ticker': tickers,
            'Poids␣(%)': weights
        })
```

— **Bouton « Valider le Portefeuille »** : Lance l'étape de vérification stricte.
— Nettoie les listes `tickers` et `weights` (enlève les *strings* vides).
— Repère les *duplicates* pour empêcher deux fois le même ticker.
— Construit un `DataFrame` « `df` »avec deux colonnes : `Ticker`, `Poids (%)`.

```
        total_weight = df['Poids␣(%)'].sum()
        if total_weight == 0:
            st.error("La␣somme␣des␣poids␣est␣gale␣␣0%.␣Veuillez␣entrer␣des␣poids␣valides.")
            st.stop()

        try:
            df['Poids␣(%)'] = df['Poids␣(%)'].astype(float)
        except ValueError:
            st.error("Veuillez␣entrer␣des␣valeurs␣numriques␣valides␣pour␣les␣poids.")
            st.stop()

        if (df['Poids␣(%)'] <= 0).any():
            st.error("Tous␣les␣poids␣doivent␣tre␣strictement␣positifs.")
            st.stop()
        else:
            if not np.isclose(total_weight, 100.0):
                st.warning(f"  ␣La␣somme␣des␣poids␣est␣de␣{total_weight:.2f}%.")
                normalize = st.checkbox("Voulez-vous␣normaliser␣les␣poids␣automatiquement␣?", key="normali
                if normalize:
                    df['Poids␣(%)'] = df['Poids␣(%)'] * 100 / total_weight
                    total_weight = df['Poids␣(%)'].sum()
                    st.success("Les␣poids␣ont␣t␣normaliss.")
```

— **Validations** :
  — Poids totaux non nuls.
  — Types numériques valides (`astype(float)`).
  — Poids strictement positifs (exclut 0 ou négatif).
  — Option de **normaliser** automatiquement à 100% (case à cocher).

```
        valid_tickers = []
        invalid_tickers = []
        company_names = []

        for ticker in df['Ticker']:
            if ticker:
                try:
                    company_name = get_company_name(ticker)
                    if company_name != 'N/A':
                        valid_tickers.append(ticker.upper())
                        company_names.append(company_name)
                    else:
                        invalid_tickers.append(ticker)
```

```
            except Exception as e:
                invalid_tickers.append(ticker)
                st.error(f"Erreur : {e}")

    if invalid_tickers:
        st.error(f"Les tickers suivants ne sont pas valides : {', '.join(invalid_tickers)}")
        st.stop()
    elif not valid_tickers:
        st.error("Aucun ticker valide n'a t fourni.")
        st.stop()
    else:
        portfolio = {
            'Actions': valid_tickers,
            'Nom de l\'Entreprise': company_names,
            'Poids (%)': [df.loc[df['Ticker'] == ticker, 'Poids (%)'].values[0] for ticker in valid_t
        }
        portfolio_df = pd.DataFrame(portfolio)
        for i in range(len(portfolio_df)):
            portfolio_df.loc[i, 'Industrie'] = yf.Ticker(portfolio_df.loc[i, 'Actions']).info.get('in
```

— **Validation des tickers** :
  — `get_company_name(ticker)` : Vérifie si on obtient un « longName ».
  — `invalid_tickers` : On stoppe tout si on en détecte.
— **Construction du `portfolio_df`** :
  — `Actions` : la liste de `tickers`.
  — `Nom de l'Entreprise` : la liste correspondante.
  — `Poids (%)` : récupérés depuis `df`.
  — `Industrie` : Collectée via `yf.Ticker(...).info.get('industry', 'N/A')`.

```
        grouped_by_industry = portfolio_df.groupby('Industrie')['Poids (%)'].apply(np.sum).reset_index

        total_weight = portfolio_df['Poids (%)'].sum()
        average_weight = portfolio_df['Poids (%)'].mean()
        max_weight = portfolio_df['Poids (%)'].max()
        min_weight = portfolio_df['Poids (%)'].min()

        portfolio_cumulative, portfolio_returns = calculate_portfolio_performance(
            valid_tickers,
            np.array(portfolio_df['Poids (%)'].tolist()) / 100,
            period='1y'
        )
        expected_return = portfolio_returns.mean() * 252
        volatility = portfolio_returns.std() * np.sqrt(252)
        risk_free_rate = 0.02
        sharpe_ratio = (expected_return - risk_free_rate) / volatility if volatility != 0 else np.nan
```

— `grouped_by_industry` : Poids agrégé par secteur (`Industrie`).
— **Statistiques** du portefeuille :
  — `total_weight, average_weight, max_weight, min_weight`.
  — `expected_return` : Moyenne quotidienne $\times 252$ (jours de Bourse).
  — `volatility` : Écart-type $\times \sqrt{252}$.
  — `sharpe_ratio` : $(r_{port} - r_{rf})/\sigma_{port}$.

```
        st.write("### Votre portefeuille :")
        st.write("#### Tableau Rcapitulatif")

        fig_table = go.Figure(data=[go.Table(
            header=dict(
```

```
                values=list(portfolio_df.columns),
                fill_color='#0611ab',
                font=dict(color='white', size=14),
                align='center',
                height=40
            ),
            cells=dict(
                values=[portfolio_df[col] for col in portfolio_df.columns],
                fill_color=['#f8f9fa']*len(portfolio_df),
                font=dict(color='black', size=12),
                align='center',
                height=30
            )
        )])
        fig_table.update_layout(
            width=600,
            height=200,
            margin=dict(l=0, r=0, t=40, b=0)
        )
        st.plotly_chart(fig_table, use_container_width=True, key="visualizer_portfolio_table")
```

— **Table récapitulative** `Plotly` (`go.Table`) :
  — `header` : ligne d'en-tête (fond bleu #0611ab).
  — `cells` : données du `DataFrame`, fond gris clair.
— `st.plotly_chart` : Affiche la table.

```
        st.write("### Statistiques du Portefeuille")
        metrics_labels = [
            "Poids Total", "Poids Moyen", "Poids Maximum", "Poids Minimum",
            "Rendement Attendu", "Volatilit", "Ratio de Sharpe"
        ]
        metrics_values = [
            f"{total_weight:.2f}%", f"{average_weight:.2f}%", f"{max_weight:.2f}%", f"{min_weight:.2f
            f"{expected_return*100:.2f}%", f"{volatility*100:.2f}%", f"{sharpe_ratio:.2f}"
        ]
        metrics_descriptions = [
            "La somme des poids doit tre de 100%.", "", "", "",
            "Rendement attendu annualis (moyenne journalire x 252).",
            "Volatilit annualise ( cart -type x  252 ).",
            "Ratio de Sharpe (rendement ajust au risque)."
        ]

        num_metrics = len(metrics_labels)
        cols = st.columns(num_metrics)

        for col, label, value, description in zip(cols, metrics_labels, metrics_values, metrics_descr
            with col:
                st.metric(label, value)
                if description:
                    st.caption(description)
```

— **Affichage des métriques** sous forme de `st.metric(...)` alignées en colonnes.
— `st.caption(...)` : brève explication au-dessous (ex. : signification du Ratio de Sharpe).

```
        graph_cols = st.columns(2)

        with graph_cols[0]:
            st.write("### Rpartition du portefeuille par industrie")
            grouped_sorted = grouped_by_industry.sort_values(by='Poids (%)', ascending=False)
```

```
            plot_pie(grouped_sorted, 'Industrie')

        with graph_cols[1]:
            st.write("### Rpartition des poids dans le portefeuille")
            portfolio_df_sorted = portfolio_df.sort_values(by='Poids (%)', ascending=False)
            plot_pie(portfolio_df_sorted, 'Actions')
```

— Deux camemberts (pie charts) côte à côte :
  — `grouped_sorted` : regroupé par *Industrie*.
  — `portfolio_df_sorted` : basé sur chaque action du portefeuille.
  — `plot_pie(...)` : fonction *custom* (depuis `graph_utils`) affichant un `pie chart`.

```
        st.write("### Performance historique du portefeuille")
        plot_performance(portfolio_cumulative)
```

— **Historique** : La fonction `plot_performance(...)` trace la performance cumulative (`portfolio_cumulati`
sur la période spécifiée (1 an).

```
        st.session_state['portfolio'] = portfolio_df
        st.success(" Le portefeuille a  t  enregistr ...")
```

— **Enregistrement** du `portfolio_df` dans `st.session_state` pour un usage ultérieur.
— Message de succès pour confirmer que tout s'est bien passé.

2.9 `if __name__ == "__main__": main()`

```
if __name__ == "__main__":
    main()
```

— En Python, n'exécute la fonction `main()` que si le fichier est lancé directement, et non importé.
— Dans le contexte Streamlit, on fera `streamlit run Portfolio_visualizer.py`.

3 Conclusion

La page **Portfolio Visualizer** :

1. **Centralise** la construction d'un portefeuille (import dynamique de tickers, gestion des poids).
2. **Filtre** les titres par indice pour une sélection plus rapide.
3. **Incorpore** la *watchlist* (possibilité d'ajouter un ticker en un clic).
4. **Vérifie** la somme des poids, la validité des tickers, et bloque la validation en cas d'erreur.
5. **Affiche** un tableau complet des actifs retenus, leurs poids, leurs industries, et en parallèle :
   — La performance historique du portefeuille (sur 1 an par défaut).
   — Les principales métriques (rendement, volatilité, ratio de Sharpe).
   — Deux *pie charts* illustrant la répartition par *industrie* et par *ticker*.
6. **Sauvegarde** le portefeuille final dans `st.session_state`, facilitant son accès (ex. : pour une optimisation plus poussée).