Documentation de la page « Stock Picking »

Gabriel Pézennec

29 décembre 2024

Table des matières

1 Introduction

Cette documentation détaille le fonctionnement et la logique de la page **Stock Picking** de l'application. Elle permet à l'utilisateur de :

- **Filtrer** des entreprises par indice (ex. : CAC 40, S&P 500, etc.).
- **Afficher** la fiche descriptive d'une entreprise : son secteur, son industrie, sa capitalisation, ses ratios financiers.
- **Visualiser** un graphique en chandeliers basé sur l'historique de ses cours.
- **Ajouter/Retirer** le ticker à une *watchlist* persistant dans `st.session.state`.

2 Explications détaillées

2.1 Importations

- `streamlit` : Bibliothèque pour créer des applications web en Python de façon très simple. Permet d'utiliser des widgets (boutons, selectbox, multiselect, etc.) et de définir un *session state*.
- `pandas` : Librairie pour manipuler des données tabulaires (DataFrames). Offre des méthodes utiles (`.unique()`, `.tolist()`, `.isin()`, etc.).
- `yfinance` : Permet de récupérer des données financières depuis Yahoo Finance (cours d'action, ratios, news, etc.) via des objets `yf.Ticker`.
- `plotly.graph_objs` : Modèles d'objets Plotly permettant de générer des graphiques interactifs (ici, un graphique en chandelier).

2.2 `main()`

La fonction `main()` constitue le coeur de la page. C'est là que Streamlit construit l'interface et gère l'interaction utilisateur.

- **Session State**

```
if 'watchlist' not in st.session_state:
    st.session_state.watchlist = []
```

st.session.state maintient l'état entre les différents re-rendus de la page. Ici, on initialise une liste `watchlist` (si elle n'existe pas déjà) pour stocker les tickers que l'utilisateur souhaite suivre. **Avantage** : même si la page est redessinée (à cause de l'interaction avec un widget), la liste reste en mémoire.

— **CSS :**
  — On modifie la mise en forme de la « block-container » pour rendre la page plus large (max-width : 90%).
  — `unsafe_allow_html=True` autorise l'insertion de code HTML/CSS brut dans le markdown de Streamlit.

— **Chargement des données :**

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

  — On utilise la fonction `pd.read_csv` pour lire le contenu du fichier CSV et le charger dans un DataFrame `df`.
  — **DataFrame** : structure de données en colonnes (par exemple, `Company`, `Ticker`, `Ind`, etc.).

— **Titre et description :**
  — `st.title("Stock picking")` : Crée un titre de page en haut de l'interface Streamlit.
  — `description` : Brève explication du concept de stock picking.
  — `justified_description` : On entoure le texte avec du HTML (balise `<div>` et `style`) pour justifier le paragraphe.

— **Filtrer par indice :**

```
indices = df['Ind'].unique().tolist()
selected_indices = st.multiselect(
    "Choisissez un ou plusieurs indices",
    options=indices,
    default=indices
)
```

  — `df['Ind']` : Sélection de la colonne « Ind » dans le DataFrame.
  — `.unique()` : Méthode Pandas retournant un array (ou *Index*) des valeurs uniques de cette colonne (pour éviter les doublons).
  — `.tolist()` : Convertit cet array en *liste* Python exploitable (type `list`).
  — `st.multiselect` : Widget Streamlit proposant de cocher/décocher plusieurs options dans `indices`.
  — `default=indices` : Indique que, par défaut, toutes les options (tous les indices) sont sélectionnées.
  — Le résultat de la sélection est stocké dans `selected_indices`, qui est une liste des indices cochés.

  Pour filtrer :

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

- `df['Ind'].isin(selected_indices)` : Méthode Pandas permettant de renvoyer un masque booléen `True/False` si la valeur dans `df['Ind']` est présente dans la liste `selected_indices`.
- `df[...]` : Sélectionne uniquement les lignes où le masque est `True`.
- `df.copy()` : Si `selected_indices` est vide (cas extrême), on se contente de copier tout le DataFrame d'origine (pas de filtrage).
- `if filtered_companies.empty: st.warning("...")` : On prévient l'utilisateur si aucun résultat n'est trouvé.

— **Sélection d'une entreprise :**

```
companies = filtered_companies["Company"].tolist()
selected_company = st.selectbox("Choisissez une entreprise :", companies)
```

- `filtered_companies["Company"]` : On sélectionne la colonne `Company` (contenant les noms d'entreprises) dans le DataFrame déjà filtré par indices.
- `.tolist()` : On convertit le Series Pandas résultant en une liste Python.
- `st.selectbox(..., companies)` : Permet à l'utilisateur de choisir **une seule** entreprise parmi la liste (contrairement à `multiselect` qui est multiple).

— **Récupération du Ticker :**

```
ticker = filtered_companies[filtered_companies["Company"]
           == selected_company]["Ticker"].values[0]
```

- `filtered_companies["Company"] == selected_company` : On crée un masque booléen pour trouver la ligne où `Company` correspond exactement au nom choisi.
- `filtered_companies[...]["Ticker"]` : Parmi les lignes correspondantes, on sélectionne la colonne `Ticker`.
- `.values[0]` : Accède au premier élément du tableau de résultats (en principe, il y en a qu'un seul si les données sont cohérentes).
- **But** : On obtient ainsi le *ticker* boursier de l'entreprise sélectionnée.

— **Choix de la période :**

- `period_options = [...]` : Liste de périodes lisibles par l'utilisateur, en français (« 1 mois », « 3 mois », etc.).
- `period_mapping` : Correspondance entre l'intitulé utilisateur (ex. « 1 mois ») et la chaîne compréhensible par yfinance (ex. « 1mo »).
- `period_choice = st.selectbox("...", period_options, index=3)` : Permet de sélectionner la période, avec un index par défaut (3 = « 1 an »).
- `period = period_mapping[period_choice]` : On traduit l'option choisie pour en faire un code qui sera compris par `yfinance`.

— **Récupération des données YFinance :**

```
stock = yf.Ticker(ticker)
data = stock.history(period=period)
info = stock.info
```

- `yf.Ticker(ticker)` : Crée un objet YFinance associé à ce ticker. Cet objet peut donner accès à `history()`, `info`, `news`, etc.
- `stock.history(period=period)` : Récupère l'historique de prix (open, high, low, close) et d'autres informations (volume) sur la période choisie.
- `stock.info` : Renvoie un dictionnaire contenant une foule de métadonnées (cap boursière, secteur, ratio P/E, etc.).
- **try/except** : Si yfinance rencontre un souci (ticker inconnu, connexion), on affiche un message d'erreur `st.error(...)` au lieu de faire crasher l'app.

— **Vérification `if not data.empty` :**
  — Si `data` est vide (`data.empty == True`), on prévient l'utilisateur que « Les données pour ce ticker ne sont pas disponibles. »
  — Sinon, on continue pour afficher les informations relatives à ce ticker.
— **Métriques principales (colonnes `col1, col2, col3, col4`) :**
  — `st.columns([2,2,2,1])` : Crée 4 colonnes de largeurs relatives 2, 2, 2 et 1.
  — Dans chaque colonne, on appelle `st.metric(...)` pour afficher une *métrique* (ex. « Entreprise : Apple Inc », « Secteur : Technology », etc.).
  — **Exemples de clés `info` :**
    — `longName` : nom complet de l'entreprise (ex. « Apple Inc. »).
    — `sector` : Secteur d'activité (ex. « Technology »).
    — `industry` : Industrie (ex. « Consumer Electronics »).
    — `marketCap` : Capitalisation boursière (valeur de la société en Bourse).
    — `trailingPE` : Ratio `P/E` basé sur les bénéfices passés.
    — `dividendYield` : Rendement du dividende.
  — **Formatting de la capitalisation (`marketCap`)** : on vérifie si c'est au-dessus de 10^{12} (on affiche des « T »), etc.
— **Bouton Watchlist :**

```
if ticker.upper() in st.session_state.watchlist:
    watchlist_label = " "
else:
    watchlist_label = " "

if st.button(watchlist_label, key=f"watchlist_{ticker.upper()}"):
    ...
```

  — On choisit l'icône à afficher ("étoile pleine" si déjà dans la watchlist, "étoile vide" sinon).
  — Au clic, on ajoute ou on retire `ticker.upper()` de la liste `watchlist`.
  — `st.success(...)` : On affiche un petit encart vert pour confirmer l'ajout ou la suppression.
— **Résumé de l'entreprise :**
  — `info.get('longBusinessSummary', 'Aucun résumé disponible.')` : Récupère la description de l'entreprise. S'il n'y a pas de clé `longBusinessSummary`, on utilise « Aucun résumé disponible. »
  — *Justification* du texte : on encadre la variable `summary` dans un `<div>` avec des styles CSS.
— **Graphique en Chandeliers :**
  — `go.Candlestick(...)` : Objet Plotly permettant de tracer un chandlier.
    — `x=data.index` : L'axe des X est la série de dates (Index du DataFrame).
    — `open=data['Open']` : Ouverture.
    — `high=data['High']` : Haut de la journée.
    — `low=data['Low']` : Bas de la journée.
    — `close=data['Close']` : Clôture de la journée.
    — Couleurs personnalisées pour les bougies haussières ou baissières (`increasing_line_color` / `decreasing_line_color`).
  — `fig.update_layout(...)` : Personnalise le titre, l'axe X, l'axe Y, la taille du graphe, le `template` (fond blanc), etc.
  — `st.plotly_chart(fig, use_container_width=True)` : Affiche le graphique de façon interactive, avec *zoom*, *hover* et *rangeslider*.
— **Ratios Financiers :**

- On construit un dictionnaire `ratios` associant un nom (ex. « P/E Ratio ») à un objet contenant :
  - `"value"` : la valeur du ratio, typiquement `info.get('trailingPE', 'N/A')`.
  - `"description"` : une explication courte de ce que signifie ce ratio.
- `st.columns(len(ratios))` : Crée autant de colonnes que de ratios, pour les présenter côte à côte.
- Dans chaque colonne, on appelle `st.metric(label=..., value=...)` pour afficher la valeur, puis `st.caption(...)` pour afficher la description.

— **(Optionnel) Actualités** :

- `stock.news` : YFinance peut fournir un *flux* d'actualités reliées au ticker, si disponible.
- On peut itérer sur `news[:5]` pour afficher les 5 derniers articles (titre, lien, résumé).
- Dans le code, c'est commenté, donc non actif par défaut.

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

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

- En Python, ce bloc s'exécute uniquement si le script est lancé directement (par exemple, `python 2_Recherche_Ticker.py`), et non quand le fichier est importé comme module dans un autre script.
- Il appelle la fonction `main()`, qui enclenche l'ensemble de la logique Streamlit décrite ci-dessus.

3 Conclusion

La page *Stock Picking* a pour objectif de :

1. Proposer un **filtrage dynamique** des entreprises par indice.
2. **Faciliter la recherche** d'une entreprise via un `selectbox`.
3. **Afficher des informations clés** : secteur, industrie, capitalisation boursière, ratios financiers, historique des cours sous forme de graphique en chandeliers.
4. **Gérer une watchlist** : l'utilisateur peut marquer (étoile) ou dé-marquer l'entreprise pour la retrouver plus tard.