Documentation de la page « Beta Forecast »

Gabriel Pézennec

29 décembre 2024

Table des matières

1 Introduction

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

- **Sélectionner** une entreprise parmi une liste prédéfinie (fichier CSV).
- **Choisir** un *horizon de prévision* (de 30 à 365 jours).
- **Exécuter** un modèle de prédiction via la bibliothèque `Prophet` pour estimer les prix futurs.
- **Visualiser** l'historique et les prévisions sous forme d'un *graphique interactif*, avec intervalles de confiance.
- **Interpréter** rapidement la tendance via un *tableau de métriques* et une *jauge de recommandation*.
- **Exporter** les résultats (DataFrame de prévisions) en CSV pour un usage ultérieur.

2 Explications détaillées

2.1 Importations

- `streamlit` : Bibliothèque clé pour créer des applications web en Python. *Utilité* : définir des widgets (slider, selectbox, etc.) et afficher des graphiques ou des tables.
- `yfinance` : Librairie facilitant l'extraction de données boursières depuis Yahoo Finance.
- `pandas` : Module indispensable pour la manipulation de `DataFrame` (lecture/écriture CSV, sélection, filtrage, etc.).
- `prophet` : Librairie de Facebook (Meta) pour la prévision de séries temporelles (prix, ventes, taux, etc.).
- `plotly.graph_objects` : Sous-module `Plotly` servant à générer des graphiques (scatters, chandeliers, jauges...).
- `datetime` (`dt`) : Fonctions utiles pour la gestion des dates (`dt.date.today()`, etc.).
- `numpy` : Librairie de calcul scientifique, permettant ici de gérer la palette de couleurs du *gauge* et les manipulations numériques.

2.2 `main()`

— **CSS pour ajuster la zone de contenu**

```
st.markdown(
    """
    <style>
    div.block-container {
        max-width: 90%;
        margin: auto;
        padding: 1rem;
    }
    </style>
    """,
    unsafe_allow_html=True,
)
```

— *Objectif* : Ajuster la mise en page par défaut de Streamlit.
— `max-width: 90%` : permet de limiter la largeur, rendant l'interface plus agréable et centrée.
— `unsafe_allow_html=True` : autorise le code HTML/CSS directement dans la page.

— **Titre et description**

```
st.title("Beta␣  ␣Trading")

description = (
  "Beta␣Forecast␣est␣une␣fonctionnalit␣avance␣..."
)
justified_description = f"""
<div style='text-align: justify; text-justify: inter-word;'>
    {description}
</div>
"""
st.markdown(justified_description, unsafe_allow_html=True)
```

— `st.title(...)` : Place un titre principal (« Beta π^2 Trading ») au sommet de la page.
— `description` : texte de présentation décrivant les grandes lignes de la fonctionnalité « Beta Forecast ».
— `justified_description` : Emballe le texte dans un bloc HTML pour obtenir un alignement justifié, plus propre à la lecture.
— `st.markdown(..., unsafe_allow_html=True)` : injection du HTML, avec mise en forme intégrée.

— **Chargement du CSV `data_pisquared.csv`**

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

— Lecture du fichier CSV contenant au moins les colonnes `Company` (nom complet de l'entreprise) et `Ticker` (symbole boursier).
— `df` est un `DataFrame` manipulable (filtrage, sélection...).

— **Sélection d'une entreprise**

```
entreprise = st.selectbox("Choisissez␣une␣entreprise␣:", df['Company'])
```

— `st.selectbox(...)` : Widget Streamlit permettant de choisir une seule valeur dans une liste.
— Ici, la liste est `df['Company']` : toutes les entreprises disponibles dans le CSV.
— La valeur choisie par l'utilisateur est stockée dans `entreprise`.

— **Horizon de prévision (slider)**

```
horizon = st.slider("Horizon de prvision (en jours) :",
                    min_value=30, max_value=365, value=90)
```

- `st.slider` : Propose un curseur pour sélectionner une valeur entière comprise entre 30 et 365.
- `value=90` : Valeur initiale par défaut (3 mois).
- `horizon` : Définit le nombre de jours pour lesquels on projette le prix futur.

— **Correspondance *nom d'entreprise* → Ticker**

```
ticker = df.loc[df['Company'] == entreprise, 'Ticker'].values[0]
```

- On filtre `df` sur la colonne `Company` pour égaliser à `entreprise`.
- On récupère la colonne `Ticker` de la ligne correspondante.
- `.values[0]` : Extraire la première valeur (unique) du résultat.

— **Téléchargement des données historiques (Yahoo Finance)**

```
current_date = str(dt.date.today())
data = yf.download(ticker, start="2015-01-01", end=current_date)
```

- `current_date = str(dt.date.today())` : Convertit la date du jour en `"YYYY-MM-DD"`.
- `yf.download(...)` : Charge les cours (`Open, High, Low, Close, Volume, ...`) depuis 2015 jusqu'à `current_date`.
- `data` est alors un `DataFrame` Pandas.

— **Préparation pour Prophet**

```
data = data[['Close']].reset_index()
data.columns = ['ds', 'y']
data['ds'] = data['ds'].dt.tz_localize(None)
```

- `[['Close']]` : On ne garde que la colonne `Close`, correspondant à la valeur que l'on cherche à prédire.
- `reset_index()` : Transforme l'index en colonne (notamment la date).
- `data.columns = ['ds', 'y']` : **Obligation** de la librairie `Prophet` :
  - `ds` = Date
  - `y` = Valeur ciblée (ici, cours de clôture)
- `tz_localize(None)` : Retire le fuseau horaire pour éviter tout conflit interne avec `Prophet`.

— **Initialisation et entraînement du modèle Prophet**

```
m = Prophet()
m.fit(data)
```

- `m = Prophet()` : Crée une instance du modèle Prophet avec ses paramètres par défaut (saisonnalité, etc.).
- `m.fit(data)` : Entraîne le modèle sur l'historique, c'est-à-dire qu'il ajuste ses paramètres pour prédire la colonne `y` en fonction de `ds`.

— **Création de la DataFrame de prévisions et prédictions**

```
future = m.make_future_dataframe(periods=horizon)
forecast = m.predict(future)
```

- `m.make_future_dataframe` : Génère des dates futures sur `horizon` jours à partir de la dernière date disponible dans `data`.
- `m.predict(future)` : Calcule la prévision pour toutes les dates (y compris l'historique pour comparaison).
- `forecast` comporte les colonnes `ds`, `yhat`, `yhat_lower`, `yhat_upper`, etc.

— **Construction d'un *Figure* Plotly**

```
fig = go.Figure()
```

- `go.Figure()` : Création d'une figure Plotly vide.
- Sur cette figure, on va ajouter différentes *traces* (Scatters) pour tracer l'historique, la prédiction et l'intervalle de confiance.

— **Affichage des données historiques (noir)**

```
fig.add_trace(go.Scatter(
    x=data['ds'], y=data['y'],
    mode='lines',
    name="Donnes historiques",
    line=dict(color="black")
))
```

- `data['ds']` : Les dates en abscisse.
- `data['y']` : Les cours de clôture en ordonnée.
- `mode='lines'` : On dessine une courbe continue.
- `name="Données historiques"` : Légende qui apparaît sur le graphique.

— **Affichage de la prédiction (bleu)**

```
fig.add_trace(go.Scatter(
    x=forecast['ds'], y=forecast['yhat'],
    mode='lines',
    name="Prdictions",
    line=dict(color="blue")
))
```

- `forecast['ds']` : Les dates (historiques et futures) générées par Prophet.
- `forecast['yhat']` : La prévision du modèle (valeur médiane).
- Couleur bleue pour la distinguer de l'historique.

— **Intervalles de confiance (`yhat_upper` et `yhat_lower`)**

```
fig.add_trace(go.Scatter(
    x=forecast['ds'], y=forecast['yhat_upper'],
    fill=None,
    mode='lines',
    line=dict(color='lightblue', dash='dot'),
    name="Intervalle suprieur"
))
fig.add_trace(go.Scatter(
    x=forecast['ds'], y=forecast['yhat_lower'],
    fill='tonexty',
    mode='lines',
    line=dict(color='lightblue', dash='dot'),
    name="Intervalle infrieur"
))
```

- `yhat_upper` / `yhat_lower` : Limites supérieure et inférieure de l'intervalle de confiance prédictif.
- `fill='tonexty'` : Remplit la zone entre la courbe `yhat_upper` (ajoutée juste avant) et `yhat_lower`.
- `dash='dot'` : Style de ligne en pointillés, pour mettre en évidence les bornes.

— **Personnalisation de la figure**

```
fig.update_layout(
    title="Prdiction des Prix Futures",
    height=600,
    width=1000,
```

```
    xaxis_title="Date",
    yaxis_title="Prix ($)",
    template="plotly_white"
)
st.plotly_chart(fig, use_container_width=True)
```

- `title` : Donne un titre explicite au graphe.
- `xaxis_title, yaxis_title` : Labels des axes.
- `template="plotly_white"` : Fond blanc standard.
- `use_container_width=True` : Le graphe s'adapte à la largeur du conteneur Streamlit.

— **Récupération des valeurs clés sur la dernière date de prévision**

```
last_date = forecast['ds'].iloc[-1]
last_date_str = last_date.strftime('%Y-%m-%d')
last_adjclose = data['y'].iloc[-1]

borne_inf = forecast.loc[forecast['ds'] == last_date_str]['yhat_lower'].values[0]
borne_sup = forecast.loc[forecast['ds'] == last_date_str]['yhat_upper'].values[0]
prediction = forecast.loc[forecast['ds'] == last_date_str]['yhat'].values[0]
```

- `last_date` : Dernière date du `forecast` (soit la date la plus loin dans l'horizon).
- `last_adjclose` : Le *dernier cours réel* de la série historique (pour comparer).
- `borne_inf` (`yhat_lower`) et `borne_sup` (`yhat_upper`) : Intervalle de confiance sur la prédiction finale.
- `prediction` (`yhat`) : Valeur centrale de la prévision pour la date la plus lointaine.

— **Statistiques de la prédiction**

```
st.write(f"### Statistiques de la prdiction  {horizon} jours")
metrics_labels = [
    "Prix actuel",
    "Prix prdit",
    "Prix prdit minimum",
    "Prix prdit maximum"
]
metrics_values = [
    f"{last_adjclose:.2f}",
    f"{prediction:.2f}",
    f"{borne_inf:.2f}",
    f"{borne_sup:.2f}"
]
...
```

- On prépare un petit tableau de 4 valeurs : Prix actuel, prédiction médiane, borne inférieure et borne supérieure.
- `:.2f` : Format numérique à 2 décimales.
- **Objectif** : Synthétiser les éléments importants pour l'investisseur.

— **Affichage en colonnes via `st.columns()`**

```
cols = st.columns(num_metrics)
for col, label, value in zip(cols, metrics_labels, metrics_values):
    with col:
        st.metric(label, value)
```

- `st.columns(num_metrics)` : Crée 4 colonnes, car on a 4 métriques.
- `st.metric(label, value)` : Widget Streamlit affichant un gros « chiffre »avec un label au-dessus.
- Très utile pour donner un aperçu rapide (KPI).

— **Calcul d'une variation en pourcentage et conversion en jauge**

```
percentage_delta = ((prediction - last_adjclose) / last_adjclose) * 100
niveau = max(min(50 + percentage_delta * 5, 100), 0)
```

- `percentage_delta` : Mesure l'écart relatif entre la prévision et le cours actuel en %.
- `niveau` : Transforme cet écart en un score sur 0–100 :
  - `50 + percentage_delta * 5` : $\Delta\% \times 5$ partant d'une base de 50.
  - `min(..., 100)` et `max(..., 0)` : On borne ce niveau entre 0 et 100.
- **Logique** : Si la prévision est bien plus élevée que l'actuel, `niveau` se rapprochera de 100. Si elle est beaucoup plus faible, on tendra vers 0.

— **Attribution du libellé de recommandation**

```
if 80 <= niveau <= 100:
    recommendation = "Strong Buy"
    text_color = "darkgreen"
elif 60 <= niveau < 80:
    recommendation = "Buy"
    ...
```

- Selon la tranche dans laquelle se situe `niveau`, on affiche « Strong Buy », « Buy », « Hold », « Sell », etc.
- `text_color` : Couleur associée au texte, pour le mettre en évidence (vert = achat, rouge = vente...).
- **Remarque** : Les seuils (20, 40, 60, 80) sont arbitraires et modifiables selon la stratégie de l'application.

— **Construction de la jauge (gauge) Plotly**

```
num_levels = 100
fig_gauge = go.Figure(go.Indicator(
    mode="gauge",
    value=niveau,
    title={'text': "Recommandation"},
    gauge={
        'axis': {'range': [0, 100], 'tickwidth': 1, 'tickcolor': "black"},
        'bar': {'color': "black", 'thickness': 0.2},
        'steps': [
            {'range': [i, i + 100 / num_levels],
             'color': f"rgb({int(255 - (i * 2.55))},{int(i * 2.55)},0)"}
            for i in np.linspace(0, 100, num_levels)
        ],
        'threshold': {
            'line': {'color': "black", 'width': 4},
            'thickness': 0.75,
            'value': niveau
        }
    }
))
```

- `go.Indicator(mode="gauge")` : Type de graphique dédié à l'affichage d'une jauge circulaire.
- `value=niveau` : La position de l'aiguille sur l'échelle 0–100.
- `'steps'` : Génère un dégradé de couleurs du rouge au vert (ou inversement) sur l'intervalle [0, 100].
- `'threshold'` : Affiche une ligne indiquant la valeur courante du `niveau`.

— **Ajout du texte de recommandation et affichage**

```
fig_gauge.add_trace(go.Scatter(
    x=[0.5],
```

```
    y=[-1.2],
    text=[f"<b>{recommendation}</b>"],
    mode="text",
    textfont=dict(size=50, color=text_color),
    showlegend=False
))

fig_gauge.update_layout(
    xaxis=dict(visible=False),
    yaxis=dict(visible=False),
    paper_bgcolor="white"
)

st.plotly_chart(fig_gauge, use_container_width=True)
```

- `go.Scatter(..., mode="text")` : On place un élément de texte en dessous de la jauge (coordonnées `x=[0.5], y=[-1.2]`).
- `textfont=dict(...)` : Paramètre la taille et la couleur du texte (Strong Buy, Sell, etc.).
- `paper_bgcolor="white"` : Fond blanc pour simplifier la lecture.

— **Bouton de téléchargement CSV**

```
csv = forecast.to_csv(index=False)
st.download_button(
    label=" Tlcharger ␣les␣prvisions␣en␣CSV",
    data=csv,
    file_name=f"previsions_{entreprise}.csv",
    mime='text/csv'
)
```

- `forecast.to_csv(...)` : Convertit tout le DataFrame de prévision en texte CSV.
- `st.download_button(...)` : Propose un bouton « Télécharger », permettant à l'utilisateur de sauvegarder localement les prévisions.

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

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

- Bloc standard en Python.
- *But* : Exécuter la fonction `main()` si le script est lancé directement, et non importé.
- Dans le contexte de Streamlit, cela garantit que l'application est construite et affichée lorsque l'on fait `streamlit run Beta_Forecast.py`.

3 Conclusion

La page *Beta Forecast* fournit :

1. **Une interface intuitive** : Sélection de l'entreprise, choix de la période, visualisation interactive.
2. **Une prévision de prix** : S'appuie sur `Prophet`, reconnu pour sa gestion des tendances, saisonnalités et incertitudes.
3. **Un aperçu rapide** : Les métriques clés (prix actuel, prédit, min, max) sont affichées de manière synthétique.
4. **Un indicateur de décision** : La jauge « recommandation », colorée et explicite (Strong Buy, Buy, etc.).

5. **Un export des résultats** : Possibilité de récupérer les prévisions au format CSV pour les analyser dans d'autres outils.