Université de Strasbourg

Recherche d'information

Interface de recherche avec Streamlit

Enzo Doyen

2025 - LGC6KM43 - M2

Principe de Streamlit

Streamlit est un framework Python qui permet de créer simplement et rapidement des applications Web interactives en Python.

Ce framework est particulièrement adapté pour les applications de visualisation de données, de machine learning et de TAL en raison de l'intégration facile de code Python existant.

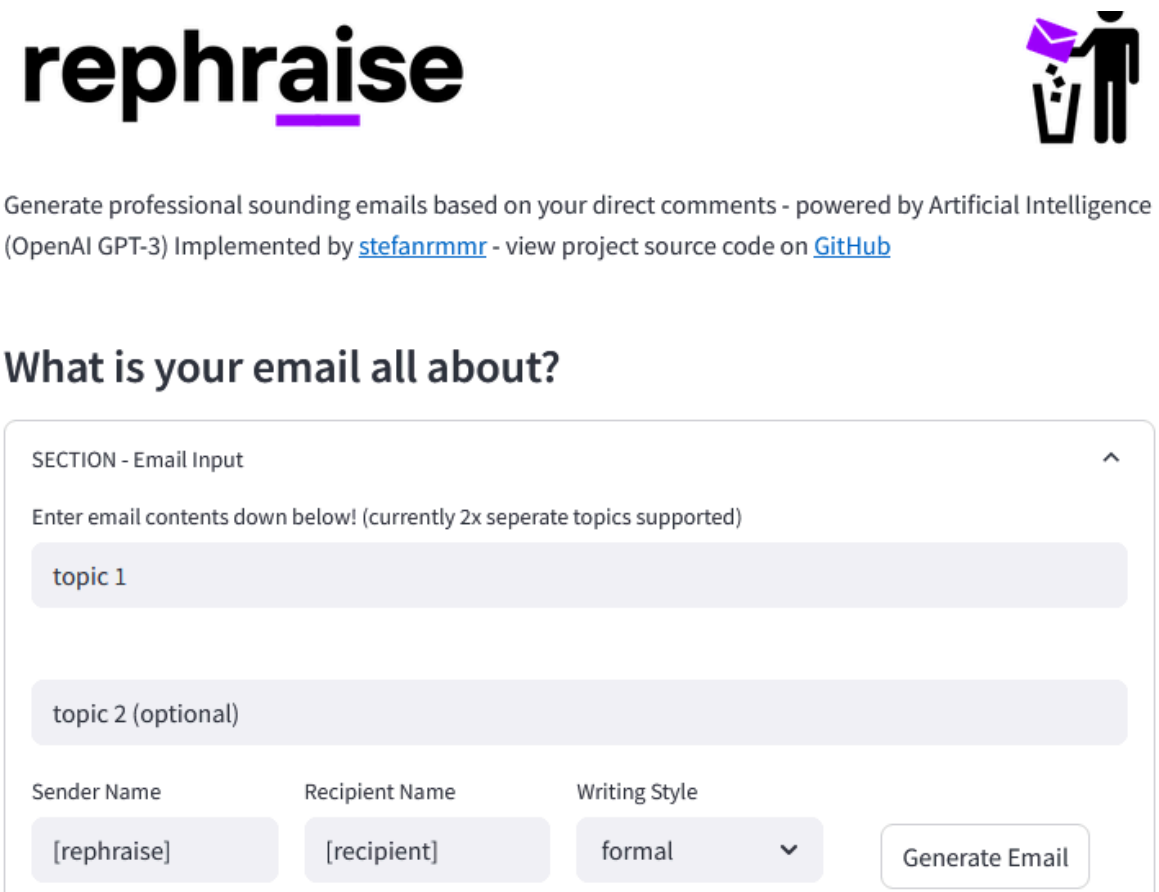
Principe de Streamlit

Streamlit est un framework Python qui permet de créer simplement et rapidement des applications Web interactives en Python.

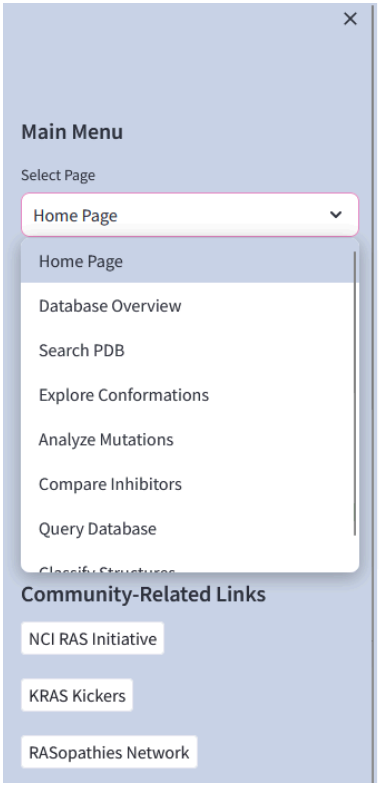
Ce framework est particulièrement adapté pour les applications de visualisation de données, de machine learning et de TAL en raison de l'intégration facile de code Python existant.

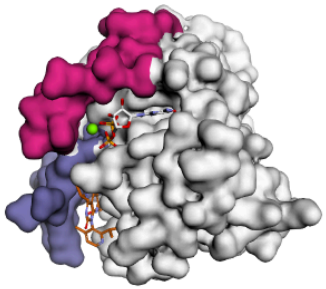
En outre, le framework est conçu pour éviter (par défaut) l'utilisation de code HTML, CSS ou JavaScript en utilisant des modèles prédéfinis, ce qui simplifie le développement.

Exemple de page Streamlit (stefanrmmr-gpt3-email-generator-streamlit-app-ku3fbq.streamlit.app)



Exemple de page Streamlit (rascore.streamlit.app)





Powered by Stmol (PDB: 6OIM).

Rascore

A tool for analyzing RAS protein structures

Created by Mitchell Parker and Roland Dunbrack

Fox Chase Cancer Center

Summary

Rascore is a tool for analyzing structures of the RAS protein family (KRAS, NRAS, and HRAS). The *Rascore* database presents a continually updated analysis of all available RAS structures in t
with their catalytic switch 1 (SW1) and switch 2 (SW2) loops conformationally classified and their molecular contents annotated (e.g., mutation status, nucleotide state, bound protein, inhib

Details of our work are provided in the *Cancer Research* paper, **Delineating The RAS Conformational Landscape**. We hope that researchers will use *Rascore* to gain novel insights into RAS b

I. Fondements de Streamlit

Exemple basique de page Streamlit

Python

```python
import streamlit as st

st.title("Exemple de page Streamlit")

st.markdown(
    "Voici à quoi ressemble une page Streamlit, que vous devrez créer pour le projet de _Recherche d'information_ !"
)
```

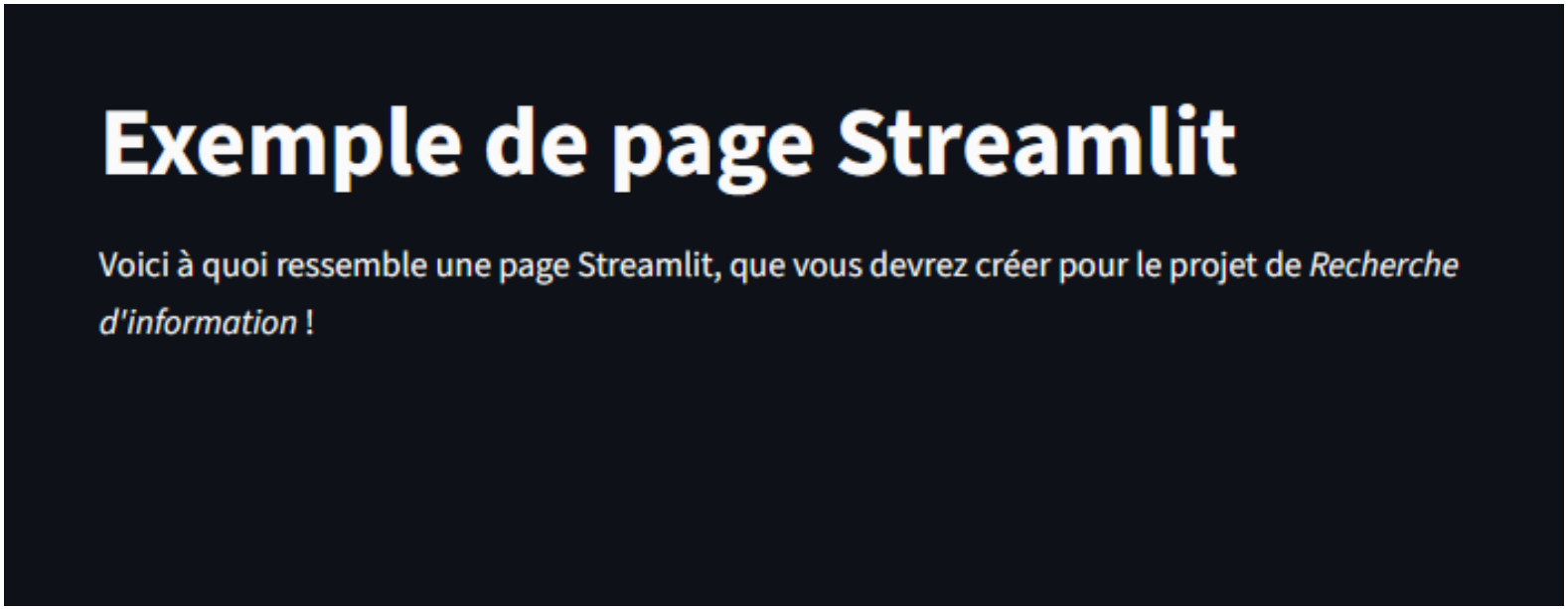
Exemple basique de page Streamlit

On peut lancer un serveur local depuis la ligne de commande pour accéder à notre page Streamlit :

```bash
streamlit run app.py
```

Où `app.py` est le nom du fichier Python contenant le code Streamlit.

Exemple basique de page Streamlit



Affichage de texte

Python

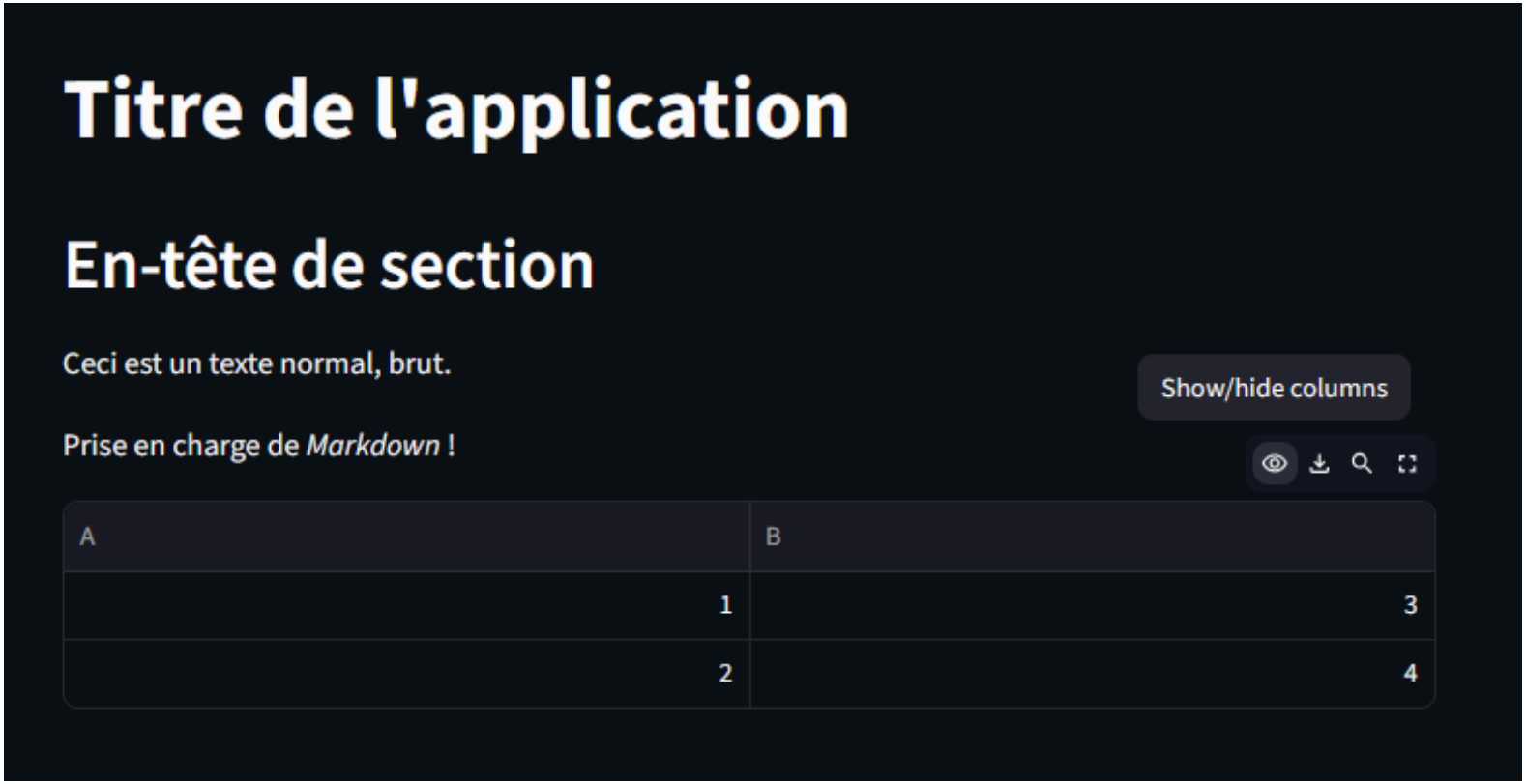
```python
st.title("Titre de l'application")
st.header("En-tête de section")
st.text("Ceci est un texte normal, brut.")
st.write("Prise en charge de *Markdown* !")
```

Affichage de données tabulées

Il est possible d'utiliser soit `st.dataframe()` pour afficher les données sous forme de tableau interactif, soit `st.table()` pour afficher les données sous forme de tableau statique.

`st.dataframe()` offre plusieurs options de personnalisation, telles que le tri, le filtrage, le téléchargement des données ou encore le redimensionnement des colonnes.

```python
st.dataframe({"A": [1, 2], "B": [3, 4]})
st.table({"A": [1, 2], "B": [3, 4]})
```


Titre de l'application
En-tête de section
Ceci est un texte normal, brut.
Prise en charge de *Markdown* !
Show/hide columns
A
B
1
3
2
4

Saisie et variables

Python

```python
name = st.text_input("Nom")
age = st.number_input("Âge", min_value=0, max_value=120, step=5)
color = st.selectbox("Choisissez une couleur", ["Rouge", "Vert", "Bleu"])

st.write(f"Bonjour {name} ! Tu as {age} ans et tu as choisi la couleur {color}.")
```

Saisie et variables : conditions

Python

```python
name = st.text_input("Nom")
age = st.number_input("Âge", min_value=0, max_value=120, step=5)
color = st.selectbox("Choisissez une couleur", ["Rouge", "Vert", "Bleu"])
agree = st.checkbox("Faire apparaitre mes infos")
if agree:
    st.write(f"Bonjour {name} ! Tu as {age} ans et tu as choisi la couleur {color}.")
```

Agencement de la page : colonnes

```python
col1, col2 = st.columns(2)
col1.write("Texte dans la colonne de gauche")
col2.write("Texte dans la colonne de droite")
```

Agencement de la page : barre latérale

```python
with st.sidebar:
    st.header("Paramètres")
    st.selectbox("Choisissez une option", ["A", "B", "C"])
```

Agencement de la page : barre latérale

Python

```python
with st.sidebar:
    st.header("Paramètres")
    option = st.selectbox("Choisissez une option",
    ["A", "B", "C"])

st.write(f"Option sélectionnée : {option}")
```

Agencement de la page : barre latérale et pages distinctes

```python
page = st.sidebar.selectbox("Choisissez une page", ["Vue d'ensemble", "Analyse", "Paramètres"])
if page == "Vue d'ensemble":
    st.write("Bienvenue sur la page de vue d'ensemble !")
elif page == "Analyse":
    st.write("Page d'analyse des données.")
elif page == "Paramètres":
    st.write("Page des paramètres.")
```

Agencement de la page : accordéon

```python
with st.expander("Voir les détails"):
    st.write("Informations supplémentaires à l'intérieur d'un accordéon.")
```

Agencement de la page : conteneurs

Python

```python
with st.container(width=40, border=True):
    st.write("Élément à l'intérieur d'un conteneur de largeur 40 avec bordure")
    col1, col2 = st.columns([1, 2])
    with col1:
        st.write("Colonne 1")
    with col2:
        st.write("Colonne 2")
```

Intégration de code HTML

Utilisation du paramètre `unsafe_allow_html=True` pour autoriser l'utilisaiton de contenu HTML :

Python

```python
st.markdown("""
    <div style="color: red; font-size: 24px;">
        Texte rouge, de taille 24 pixels.
    </div>
""", unsafe_allow_html=True)
```

Intégration de code HTML

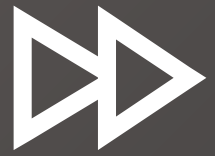
Python

```python
st.markdown("""
    <style>
        .class {
            color: red;
            font-size: 24px;
        }
    </style>
    <div class="class">
        Texte rouge, de taille 24 pixels.
    </div>
""", unsafe_allow_html=True)
```

Intégration de code CSS

En utilisant un fichier CSS externe (p. ex., avec le nom `styles.css`) :

```python
st.markdown('<style>' +
open('styles.css').read() + '</style>',
unsafe_allow_html=True)
```



II. Considérations avancées

États

Par défaut, Streamlit **exécute le code de l'application à chaque interaction** (clic de bouton, texte modifié…).

Ainsi, quand une variable est définie, sa valeur est réinitialisée à chaque interaction.

```python
count = 0
increment = st.button("Augmenter")
if increment:
    count += 1 # count reste à 1
```

États

Par défaut, Streamlit **exécute le code de l'application à chaque interaction** (clic de bouton, texte modifié…).

Ainsi, quand une variable est définie, sa valeur est réinitialisée à chaque interaction.

Pour conserver l'état d'une variable, on peut utiliser des **états**, avec `st.session_state`.

États

`st.session_state` agit de manière très similaire aux dictionnaires Python, avec des clés et des valeurs, qu'il faut initialiser.

```python
if "count" not in st.session_state:
    st.session_state.count = 0

increment = st.button("Augmenter")
if increment:
    st.session_state.count += 1

st.write("Nombre = ", st.session_state.count)
```

États

La valeur d'une clé est accessible soit via `st.session_state.key`, soit via `st.session_state["key"]`.

```python
a = st.session_state["count"]
b = st.session_state.count
```

États

On peut obtenir le même résultat en utilisant des fonctions de rappel (*callbacks*), où une fonction est utilisée comme argument pour certains éléments spécifiques, par exemple `on_click` pour l'élément `button` :

Python

```python
def increment_counter():
    st.session_state.count += 1

increment = st.button("Augmenter",
on_click=increment_counter)
```

Mise en cache

Comme vu précédemment, Streamlit **exécute le code de l'application à chaque interaction** (clic de bouton, texte modifié…).

Pour certaines opérations couteuses (comme le chargement de données ou de modèles), il est possible d'utiliser la mise en cache pour éviter de réexécuter le code à chaque fois quand la valeur attendue ne change pas.

Mise en cache

La mise en cache se fait par l'utilisation de **décorateurs [?]** dans les fonctions Python concernées. Il en existe deux types : `@st.cache_data` et `@st.cache_resource`.

`@st.cache_data` est privilégié pour les données pures fixes (nombres, listes, DataFrames...), par exemple les données textuelles. Le décorateur crée une copie de l'objet.

`@st.cache_resource` est utilisé pour les ressources qui doivent être créées une seule fois, comme les modèles de machine learning ou les objets lourds. Le décorateur sauvegarde l'objet original en mémoire ainsi que son état.

Mise en cache : exemples

```python
@st.cache_data
def load_data(path):
    return pd.read_csv(path)
```

```python
@st.cache_resource
def load_vectorizer_and_matrix(docs):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(docs)
    return vectorizer, tfidf_matrix
```

Chargement de données pour la RI

Les modèles que nous avons vus dans le cours (notamment les modèles denses, à base de plongements) peuvent nécessiter beaucoup de ressources, et il n'est pas forcément possible de les charger en local pour les faire fonctionner tels quels avec Streamlit.

Dans ce cas, il peut être préférable de calculer au préalable les scores de similarité entre plongements requêtes/documents sur une machine plus puissante (p. ex., une instance Google Colab), et de charger ces données dans Streamlit.

Chargement de données pour la RI

Cela implique de définir une liste statique de requêtes pour lesquelles calculer les plongements :

```python
# (Sur Colab ou une machine plus puissante)
QUERIES: list[str] = []
```

Chargement de données pour la RI

Définissons ensuite un dictionnaire *query_results* qui sera utilisé pour stocker les résultats de chaque modèle au format JSON :

```python
# (Sur Colab ou une machine plus puissante)
QUERIES: list[str] = []
query_results: dict[str, list[dict[str | int]]] = {}
```

Chargement de données pour la RI

Notre format JSON pourrait prendre la forme suivante :

Python

```
{
    'document' …
    'sparse_score' …
    'dense_score' …
    'hybrid_score' …
    'rank' …
}
```

Chargement de données pour la RI

Création du dictionnaire au format JSON, où chaque clé correspond à une requête prédéfinie :

```python
for i, query in enumerate(QUERIES):
  # Calculs modèles épars, dense, hybride…, puis :
  query_results[query.lower()] = [{
          'document': documents[idx],
          'sparse_score': float(sparse_scores[idx]),
          'dense_score': float(dense_scores[idx]),
          'hybrid_score': float(hybrid_scores[idx]),
          'rank': rank + 1
      } for rank, idx in enumerate(sorted_indices[:top_k])]
```

Chargement de données pour la RI

Enregistrement au format JSON :

Python

```python
import json
with open('precomputed_results.json', 'w', encoding='utf-8') as f:
    json.dump(query_results, f, indent=2, ensure_ascii=False)
```

Chargement de données pour la RI

Ensuite, dans notre application Streamlit, on peut charger ces résultats avec le module `json` et le décorateur `@st.cache_data` :

```python
# En local, même sur une machine peu puissante
@st.cache_data
def load_precomputed_results():
    with open("precomputed_results.json", "r", encoding="utf-8") as f:
        return json.load(f)
```

⊕ Ressources complémentaires

La documentation de Streamlit est très complète et propose de nombreux exemples ; n'hésitez pas à la consulter : **https://docs.streamlit.io/**

- Documentation sur les états : **https://docs.streamlit.io/develop/concepts/architecture/session-state**
- Documentation sur la mise en cache : **https://docs.streamlit.io/develop/concepts/architecture/caching**
- Exemples de projets Streamlit : **https://github.com/jrieke/best-of-streamlit**

Bibliographie

Manning, C. D., Raghavan, P., et Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Mitra, B. (2018). *An Introduction to Neural Information Retrieval* (Numéro v.41). Now Publishers.

Zhai, C., et Massung, S. (juin 2016). *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. Association for Computing Machinery and Morgan & Claypool. **10.1145/2915031**