Blue Print sur le suivi de la consommation énergétique et de l'émission de CO2 de la ville de Seattle

Objectif global

devenir neutre en émissions de carbone d'ici 2050 pour la ville de Seattle.

Le besoin des utilisateurs

le besoin des utilisateurs est de pouvoir visualiser la consommation énergétique de la ville de Seattle et les émissions de CO2 en fonction de différents critères :

- type d'habitation
- lieu
- type d'énergie

Outils utilisés

Pour réaliser notre tableau de bord et présentation, nous avons utilisé:

Figma est un choix populaire pour la conception d'interfaces utilisateur en raison de sa facilité de collaboration, de son accessibilité et sa capacité à faciliter le travail en équipe. Pour la réalisation de notre mockup nous avons utilisé Figma.

Pour la réalisation d'un tableau de bord interactif nous avons décidé de choisir streamlit. En effet, Il permet aux développeurs de créer rapidement des applications web en utilisant des scripts Python simples. Il offre une approche déclarative où vous décrivez simplement les

éléments de votre application dans un script Python, et Streamlit se charge de la conversion en une interface utilisateur interactive.

Pour la présentation de notre projet nous avons décidé d'utiliser Canva car c'est une solution facile à utiliser et qui permet de faire de belles présentations en ajoutant des images, logos rendant la présentation plus ludique.

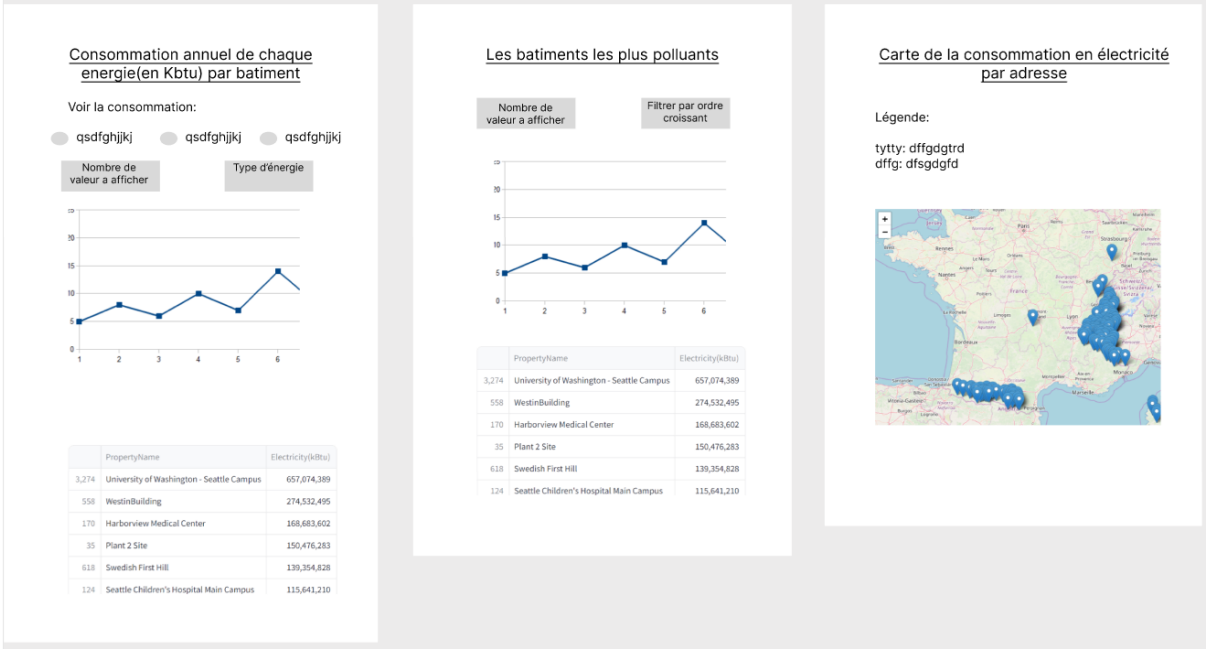
Pour l'hébergement de notre solution nous avons choisi github notamment pour stocker et partager notre code source, collaborer avec d'autres développeurs, et suivre les modifications apportées à notre projet au fil du temps. C'est un outil essentiel pour le travail en équipe et le suivi des versions, ce qui est particulièrement important dans le développement de projets logiciels.

Etapes

Les étapes d'un projet peuvent varier en fonction de sa nature, de sa taille et du domaine dans lequel il s'inscrit. Pour la réalisation de notre projet voici les différentes étapes qui ont été réalisées :

1. Mockup:

Il s'agit d'une maquette ou d'un prototype visuel qui donne un aperçu de l'apparence et de la disposition de notre interface. En réalisant le mockup nous avons eu à faire le choix de nos visualisations (diagramme de scatter, map).



2. Collecte de données:

Les données ont été collectées sur:
https://data.seattle.gov/dataset/2016-Building-Energy-Benchmarking/2bpz-gwpy
En format CSV.

3. Nettoyage des données:

Le nettoyage des données est une étape cruciale dans le processus de préparation des données avant leur analyse ou leur utilisation pour les visuels.
Le nettoyage c'est fait avec le langage python à l'intérieur d'un module "mod_data".

```
def pollution():
    po = pd.read_csv('pollution.csv', sep=";")
```

- Lecture du fichier CSV : Le fichier 'pollution.csv' est lu dans un DataFrame Pandas appelé 'po', en utilisant ';' comme séparateur.

```
    po = po.iloc[:, :-1]
                                                                    po =
po.drop(columns=['City','State','ZipCode','TaxParcelIdentificationNumber','CouncilDistrictCode'
,'Neighborhood',
                                                    'DefaultData', 'Comments','ComplianceStatus',
'Outlier','DataYear','NumberofFloors','PropertyGFATotal'

,'PropertyGFAParking','PropertyGFABuilding(s)','ListOfAllPropertyUseTypes','LargestPropertyU
seTypeGFA',

'SecondLargestPropertyUseTypeGFA','ThirdLargestPropertyUseTypeGFA','YearsENERGYSTAR
Certified',

'ENERGYSTARScore','LargestPropertyUseType','NaturalGas(therms)','SiteEUI(kBtu/sf)','SiteEUI
WN(kBtu/sf)',

'SourceEUI(kBtu/sf)','SourceEUIWN(kBtu/sf)','Electricity(kWh)','SecondLargestPropertyUseTyp
e',
                'ThirdLargestPropertyUseType'])
```

- Suppression de colonnes inutiles : Certaines colonnes du DataFrame 'po' sont supprimées pour rendre l'affichage des données plus fluide sur le dashboard. Ces colonnes semblent être celles qui ne seront pas utilisées dans la visualisation.

```
    po['OSEBuildingID'] = po['OSEBuildingID'].apply(str)
    po['YearBuilt'] = po['YearBuilt'].apply(str)
```

- Changement du type de certaines colonnes en chaînes de caractères : Les colonnes 'OSEBuildingID' et 'YearBuilt' sont converties en chaînes de caractères.

```
    po['NumberofBuildings'] = po['NumberofBuildings'].fillna(0)
    po['NumberofBuildings'] = po['NumberofBuildings'].apply(int)
```

- Changement du type de la colonne 'NumberofBuildings' en entier : Les valeurs manquantes dans la colonne 'NumberofBuildings' sont remplies avec zéro, puis la colonne est convertie en type entier.

```
                    po['SiteEnergyUse(kBtu)'].fillna(po[po['SiteEnergyUse(kBtu)']      >=
0]['SiteEnergyUse(kBtu)'].mean(), inplace=True)
```

```
po['SiteEnergyUseWN(kBtu)'].fillna(po[po['SiteEnergyUseWN(kBtu)'] >= 0]['SiteEnergyUseWN(kBtu)'].mean(), inplace=True)
po['SteamUse(kBtu)'].fillna(po[po['SteamUse(kBtu)'] >= 0]['SteamUse(kBtu)'].mean(), inplace=True)
po['Electricity(kBtu)'].fillna(po[po['Electricity(kBtu)'] >= 0]['Electricity(kBtu)'].mean(), inplace=True)
po['NaturalGas(kBtu)'].fillna(po[po['NaturalGas(kBtu)'] >= 0]['NaturalGas(kBtu)'].mean(), inplace=True)
po['TotalGHGEmissions'].fillna(po[po['TotalGHGEmissions'] >= 0]['TotalGHGEmissions'].mean(), inplace=True)
po['GHGEmissionsIntensity'].fillna(po[po['GHGEmissionsIntensity'] >= 0]['GHGEmissionsIntensity'].mean(), inplace=True)
```

- Remplacement des valeurs manquantes : Les valeurs manquantes dans certaines colonnes sont remplacées par la moyenne des valeurs positives de ces colonnes. Cela se fait pour les colonnes 'SiteEnergyUse(kBtu)', 'SiteEnergyUseWN(kBtu)', 'SteamUse(kBtu)', 'Electricity(kBtu)', 'NaturalGas(kBtu)', 'TotalGHGEmissions', 'GHGEmissionsIntensity'.

```
po = po[po['Electricity(kBtu)'] >= 0]
po = po[po['TotalGHGEmissions'] >= 0]
po = po[po['GHGEmissionsIntensity'] >= 0]
```

- Filtrage des données : Les lignes où les colonnes 'Electricity(kBtu)', 'TotalGHGEmissions', et 'GHGEmissionsIntensity' ont des valeurs négatives sont supprimées du DataFrame 'po'.

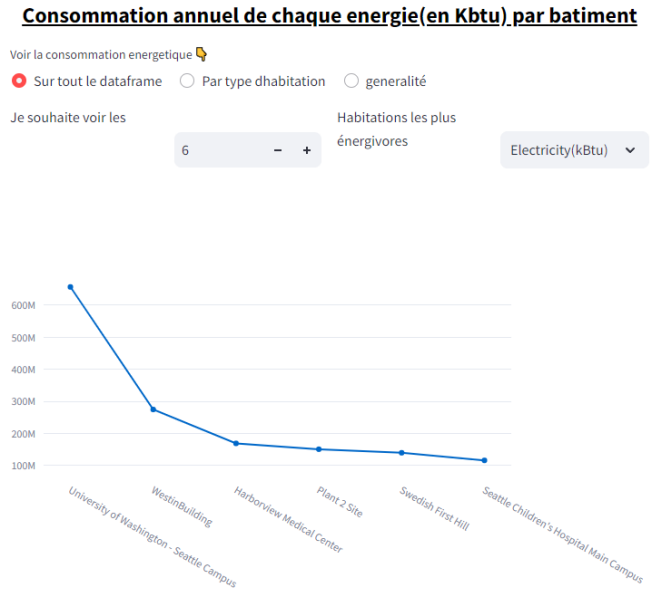
```
return po
```

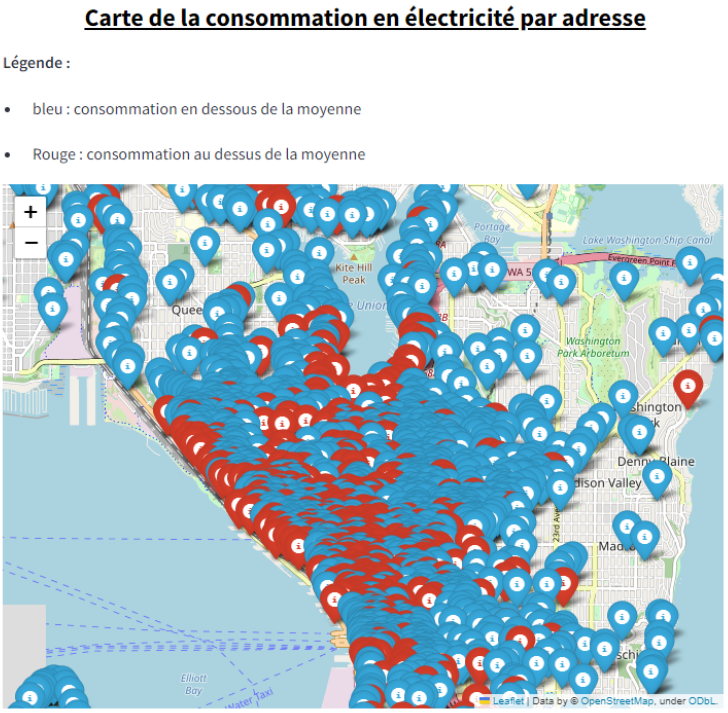
- Retour du DataFrame : La fonction retourne le DataFrame 'po' après toutes les transformations. Cela suggère que la fonction est conçue pour nettoyer et préparer les données contenues dans le fichier 'pollution.csv' pour une utilisation ultérieure dans le cadre d'un dashboard ou d'une analyse.

4. Implementation:

Pour implémenter le code nous avons choisi de coder avec Python pour faciliter la prise en main. Ensuite nous avons décomposé le problème en sous problèmes correspondant aux différentes parties de code, pour que cela soit plus gérable. Enfin, après avoir fait des tests et gérer les erreurs nous avons déployé le code via la commande du prompt run mon_application.py .

Voici deux exemples de ce qu'on obtient après avoir déployé le dashboard.:





5. Déploiement:

Le déploiement dans le contexte des logiciels ou des applications se réfère à la mise à disposition opérationnelle d'un produit, d'une application ou d'un service pour qu'il soit utilisé par les utilisateurs finaux. Pour déployer notre application nous avons utilisé Github pour héberger les fichiers et streamlit pour son lancement en ligne.

Lien vers l'app: https://projetfinal.streamlit.app/

Les différentes pages et explication du code:

1) **Accueil**🏠

```
st.markdown(f'<h1 style="color:blue;text-align: center; font-size: 40px;">{"_SCEBEC_"}</h1>', unsafe_allow_html=True)
```

- Affichage d'un titre stylisé avec Markdown : Cela utilise la fonction markdown de Streamlit pour afficher un titre avec le texte "*SCEBEC*" en bleu et centré.

```
st.markdown(f'<p style="color:black;text-align: center; font-size: 20px;">{"SUIVI DE LA CONSOMMATION ÉNERGÉTIQUE DES BÂTIMENTS ET DE ÉMISSION DU CO2"}</p>', unsafe_allow_html=True)
```

- Affichage d'un paragraphe stylisé avec Markdown : Cela utilise markdown pour afficher un paragraphe avec le texte "SUIVI DE LA CONSOMMATION ÉNERGÉTIQUE DES BÂTIMENTS ET DE ÉMISSION DU CO2" en noir et centré.

```
st.markdown(f'<h1 style="text-align: center; text-decoration: underline;font-size:30px; margin:15px">{"Infos clés"}</h1>', unsafe_allow_html=True)
```

- Affichage d'un sous-titre stylisé avec Markdown : Cela utilise markdown pour afficher un sous-titre avec le texte "Infos clés" souligné et centré.

```
st.write("➡️ Les données traitées ont été récupérées dans la ville de SEATTLE en 2016")
st.write("➡️ Le nombre de bâtiments évalués:", po['OSEBuildingID'].count())
st.write("➡️ 3 types d'énergie: gaz, vapeur, électricité")
```

- Affichage d'informations clés : Ces lignes utilisent la fonction write de Streamlit pour afficher des informations clés sur les données, y compris la ville d'origine des données, le nombre de bâtiments évalués, et les types d'énergie considérés.

```
st.markdown(f'<p style="color:blue;text-align: center; font-size: 20px;margin:30px">{"Réalisé par Fabrice et SAMI, IA M2 DA"}</p>', unsafe_allow_html=True)
```

- Affichage d'une mention de réalisation : Cette ligne utilise markdown pour afficher une mention de réalisation en bleu, centrée, avec le texte "Réalisé par Fabrice et SAMI, IA M2 DA".

- En résumé, cette partie du script vise à rendre le dashboard plus esthétique et informatif en ajoutant des éléments stylisés et des informations clés sur les données et les personnes impliquées dans la réalisation du dashboard.

2) Affichage des données

```
import streamlit as st
```

- importation de Streamlit : Streamlit est une bibliothèque Python qui facilite la création d'applications web pour l'analyse de données.

```
import json
import requests
```

- Importation de bibliothèques supplémentaires : Ces bibliothèques pourraient être utilisées pour des fonctionnalités supplémentaires ou des interactions avec des API externes, bien qu'elles ne soient pas directement utilisées dans le code que vous avez partagé.

```
import pandas as pd
```

- Importation de Pandas : Pandas est une bibliothèque Python utilisée pour la manipulation et l'analyse de données.

```
from mod_data import pollution
```

- Importation de la fonction pollution du module mod_data : Cette ligne importe la fonction pollution() du module mod_data. Cela suggère que vous avez une structure modulaire où le traitement des données est séparé dans un module distinct (mod_data).

```
st.set_page_config(
   page_title="SCEBEC",
   page_icon="🏭",
)
```

- Configuration de la page Streamlit : Cette partie configure le titre de la page (SCEBEC) et l'icône de la page (🏭).

```
po = pollution()
```

- Appel de la fonction pollution : Vous appelez la fonction pollution() pour récupérer le DataFrame po contenant les données nettoyées et prétraitées.

st.markdown(f'<h1 style="color:black;font-size:30px; text-align: center;text-decoration: underline">{"Les données utilisées pour le dashboard"}</h1>', unsafe_allow_html=True)

- Affichage d'un titre avec Markdown : Cela utilise la fonction markdown de Streamlit pour afficher un titre stylisé avec le texte "Les données utilisées pour le dashboard".

st.write(po)

- Affichage des données avec st.write : Les données nettoyées (po) sont affichées dans le dashboard avec la fonction write de Streamlit.

st.write("le TotalGHGEmissions est mesuré en tonnes métriques et c'est l'équivalent du dioxyde de carbone (CO2).")

- Affichage d'une information supplémentaire : Une note supplémentaire est ajoutée pour expliquer que la colonne 'TotalGHGEmissions' est mesurée en tonnes métriques et représente l'équivalent du dioxyde de carbone (CO2).

- En résumé, ce script utilise Streamlit pour créer un dashboard simple qui affiche les données nettoyées et prétraitées, avec une explication sur l'unité de mesure d'une colonne spécifique.

3) Statistiques sur les données📈

st.markdown(f'<h1 style="color:black;font-size:30px; text-align: center;text-decoration: underline">{"Statistiques sur les données"}</h1>', unsafe_allow_html=True)

- Affichage d'un titre avec Markdown : Cette ligne utilise la fonction markdown de Streamlit pour afficher un titre stylisé avec le texte "Statistiques sur les données"

st.write(po.describe())

- Affichage des statistiques descriptives : Les statistiques descriptives du DataFrame po sont affichées à l'aide de la fonction describe() de Pandas. Cela inclut des informations telles que la moyenne, l'écart-type, le minimum, les quartiles et le maximum pour chaque colonne numérique du DataFrame.

st.write("NB: std c'est l'écart-type, mesure de la dispersion des valeurs")

- Affichage d'une note d'information : Une note supplémentaire est ajoutée pour expliquer que "std" représente l'écart-type, qui est une mesure de la dispersion des valeurs.

- En résumé, cette partie du script enrichit le dashboard en affichant des statistiques descriptives sur les données nettoyées et prétraitées, permettant aux utilisateurs d'obtenir un aperçu plus approfondi de la distribution et de la variabilité des données.

4) Consommation énergie

```
st.markdown(f'<h1 style="color:black;font-size:24px; text-align: center;text-decoration: underline">{"Consommation annuelle de chaque énergie (en Kbtu) par bâtiment"}</h1>', unsafe_allow_html=True)
```

- Affichage d'un titre avec Markdown : Utilise la fonction markdown de Streamlit pour afficher un titre stylisé avec le texte "Consommation annuelle de chaque énergie (en Kbtu) par bâtiment".

```
choix = st.radio(
   "Voir la consommation énergétique 👇 ",
   ["Sur tout le dataframe", "Par type d'habitation","Généralité"],
   key="visibility",
   horizontal=True,
)
```

- Utilisation d'un widget Radio : La fonction radio crée un widget radio permettant à l'utilisateur de choisir entre trois options : "Sur tout le dataframe", "Par type d'habitation" et "Généralité".

```
if choix == 'Par type dhabitation':
   valeurs_uniques = po['BuildingType'].unique()
   val = ['Electricity(kBtu)','NaturalGas(kBtu)','SteamUse(kBtu)']
   selected_option = st.selectbox('Selectionner un type dhabitation pour voir les les batiments les plus energivores', valeurs_uniques)

   col1, col2, col3, col4 = st.columns(4)
   with col1:
      st.write('Je souhaite voir les')
   with col2:
      n = st.number_input("", value=6, step=1, format="%d")
   with col3:
      st.write("Habitations les plus énergivores")
   with col4:
      selected_option2 = st.selectbox('', val)

   filtre = po[po['BuildingType'] == selected_option]
   first=filtre.sort_values(by=selected_option2, ascending=False)[:n]

   data2 = [
```

```
    go.Scatter(x = first['PropertyName'], y=first[selected_option2], name='Quantité électricité utilisé', text=first[selected_option2])
  ]
  fig2 = go.Figure(data=data2)
  st.plotly_chart(fig2)

  st.write(first.loc[:,['PropertyName',selected_option2]])
```

- Affichage d'un graphique interactif pour les types de bâtiments spécifiques : En fonction du choix "Par type d'habitation", le code crée un graphique interactif avec Plotly Express montrant la quantité d'énergie consommée (choisie par l'utilisateur) pour les bâtiments du type sélectionné. Les utilisateurs peuvent également choisir le nombre de bâtiments les plus énergivores à afficher.

```
elif choix == 'Généralité':
  val = ['Electricity(kBtu)','NaturalGas(kBtu)','SteamUse(kBtu)']
  col1, col2, col3, col4 = st.columns(4)
  with col1:
    st.write('Je souhaite voir les')
  with col2:
    m = st.number_input("", value=6, step=1, format="%d")
  with col3:
    st.write("Habitations les plus énergivores")
  with col4:
    selected_option2 = st.selectbox('', val)

  first=po.sort_values(by=selected_option2, ascending=False)[:m]
  data2 = [
    go.Scatter(x = first['PropertyName'], y=first[selected_option2], name='Quantité électricité utilisé', text=first[selected_option2])
  ]
  fig2 = go.Figure(data=data2)

  st.plotly_chart(fig2)

  st.write(first.loc[:,['PropertyName',selected_option2]])
```

- Affichage d'un graphique interactif pour l'ensemble des bâtiments : En fonction du choix "Généralité", le code crée un graphique interactif avec Plotly Express montrant la quantité totale d'énergie consommée (choisie par l'utilisateur) pour chaque type d'énergie, pour l'ensemble des bâtiments. Les utilisateurs peuvent également choisir le nombre de bâtiments les plus énergivores à afficher.
- En résumé, ce script ajoute une fonctionnalité interactive au dashboard, permettant aux utilisateurs de choisir entre différentes vues de la consommation énergétique des bâtiments. Ils peuvent explorer la consommation par type d'habitation, voir la

consommation globale par type d'énergie, et visualiser les bâtiments les plus énergivores en fonction de certains critères.

5) Pollution

```
st.markdown(f'<h1 style="color:black;font-size:24px; text-align: center;text-decoration: underline">{"Les bâtiments les plus polluants"}</h1>', unsafe_allow_html=True)
```

- Affichage d'un titre avec Markdown : Utilise la fonction markdown de Streamlit pour afficher un titre stylisé avec le texte "Les bâtiments les plus polluants".

```
val = [False,True]
col1, col2, col3, col4 = st.columns(4)
with col1:
   st.write('Je souhaite voir les')
with col2:
   m = st.number_input("", value=6, step=1, format="%d")
with col3:
   st.write("Habitations les plus polluantes")
with col4:
   selected_option2 = st.selectbox('Trier par ordre croissant', val)
```

- Utilisation d'un widget Sélecteur : La fonction selectbox crée un widget de sélection permettant à l'utilisateur de choisir entre deux options : "False" (ordre décroissant) et "True" (ordre croissant). L'utilisateur peut également spécifier le nombre de bâtiments les plus polluants à afficher à l'aide du widget de saisie numérique.

```
first = po.sort_values(by='TotalGHGEmissions', ascending=selected_option2)[:m]
```

- Tri et sélection des bâtiments : Les données sont triées en fonction des émissions totales de gaz à effet de serre (TotalGHGEmissions) dans l'ordre spécifié par l'utilisateur, et les premiers m bâtiments sont sélectionnés.

```
data2 = [
   go.Scatter(x=first['PropertyName'], y=first['TotalGHGEmissions'], name='Quantité électricité utilisé', text=first['TotalGHGEmissions'])
]
fig2 = go.Figure(data=data2)

st.plotly_chart(fig2)
```

- Affichage d'un graphique interactif : Un graphique interactif est créé avec Plotly Express, montrant la quantité totale d'émissions de gaz à effet de serre pour chaque bâtiment. Le graphique est affiché à l'aide de st.plotly_chart.

```
st.write(first.loc[:,['PropertyName','TotalGHGEmissions','YearBuilt']])
```

- Affichage d'un tableau : Un tableau est affiché avec les colonnes 'PropertyName', 'TotalGHGEmissions' et 'YearBuilt' pour les bâtiments sélectionnés.
- En résumé, ce script permet à l'utilisateur de sélectionner le nombre de bâtiments les plus polluants à afficher et de choisir l'ordre de tri (croissant ou décroissant) en fonction de leurs émissions totales de gaz à effet de serre. Le résultat est affiché à la fois sous forme de graphique interactif et de tableau.

6) Carte de consommation 🗺️

```
def color_marker(electricity):
    if electricity < po['Electricity(kBtu)'].mean():
        return 'blue'
    else:
        return 'red'
```

- Fonction de couleur des marqueurs : Une fonction color_marker est définie pour attribuer une couleur différente aux marqueurs en fonction de la consommation en électricité. Les marqueurs sont bleus si la consommation est inférieure à la moyenne et rouges sinon.

```
st.markdown(f'<h1 style="color:black;font-size:24px; text-align: center;text-decoration: underline">{"Carte de la consommation en électricité par adresse"}</h1>', unsafe_allow_html=True)
```

- Affichage du titre de l'application : Utilise la fonction markdown de Streamlit pour afficher un titre stylisé avec le texte "Carte de la consommation en électricité par adresse".

```
selected_columns = ['Address', 'Electricity(kBtu)']
```

- Sélection des colonnes : Une liste selected_columns est créée pour sélectionner les colonnes nécessaires, à savoir 'Address' et 'Electricity(kBtu)'.

```
m = folium.Map(location=[po['Latitude'].mean(), po['Longitude'].mean()], zoom_start=12)
```

- Création de la carte Folium : Une carte Folium est créée avec une localisation moyenne et un niveau de zoom initial.

```
for index, row in po[selected_columns].iterrows():

    popup_text = f"Adresse: {row['Address']}<br>Consommation en électricité :
{row['Electricity(kBtu)']} kBtu"

    folium.Marker(

        location=[po.at[index, 'Latitude'], po.at[index, 'Longitude']],

        popup=folium.Popup(popup_text, max_width=300),

        icon=folium.Icon(color=color_marker(row['Electricity(kBtu)']))

    ).add_to(m)
```

- Ajout des marqueurs : Une boucle est utilisée pour ajouter des marqueurs à la carte pour chaque emplacement, avec une fenêtre contextuelle affichant l'adresse et la consommation en électricité. La couleur des marqueurs est déterminée par la fonction color_marker.

```
st.markdown("**Légende :**")

st.markdown("- bleu : consommation en dessous de la moyenne")

st.markdown("- Rouge : consommation au dessus de la moyenne")
```

- Ajout d'une légende : Une légende est ajoutée au-dessus de la carte pour expliquer la signification des couleurs des marqueurs.

```
folium_static(m)
```

- Affichage de la carte dans Streamlit : La carte Folium est affichée dans le tableau Streamlit avec folium_static.

```
st.write(f"Type de lieu où on consomme le plus
{po[po['Electricity(kBtu)']==po['Electricity(kBtu)'].max()]['PrimaryPropertyType']}")

st.write(f"Lieu où on consomme le plus
{po[po['Electricity(kBtu)']==po['Electricity(kBtu)'].max()]['Address']}")
```

- Affichage des informations sur la consommation maximale : Les types et adresses des lieux où la consommation d'électricité est maximale sont affichés.
- En résumé, ce script crée une carte interactive avec des marqueurs colorés représentant la consommation en électricité par adresse, et fournit des informations sur les lieux où la consommation est maximale.

Auteurs : Fabrice Laura TATUE KEGOM, Sami DIMACHKIE