La performance énergétique des bâtiments résidentiels en utilisant des outils d'apprentissage

Encadré par :

Mr.Youssef lamrani

Présenté par :

ARFAOUI Abderrahim

Plan:

Présentation du projet

Ce projet vise à prédire les charges de chauffage (y1) et de refroidissement (y2) pour des bâtiments résidentiels en fonction de huit caractéristiques architecturales. Le jeu de données utilisé pour cette analyse a été créé par Angeliki Xifara, ingénieure civile et en structures

Utilité:

1)Efficacité Énergétique

2)Gestion des Systèmes de CVC(les systèmes de chauffage, ventilation et climatisation)

3)Conception de Bâtiments Durables

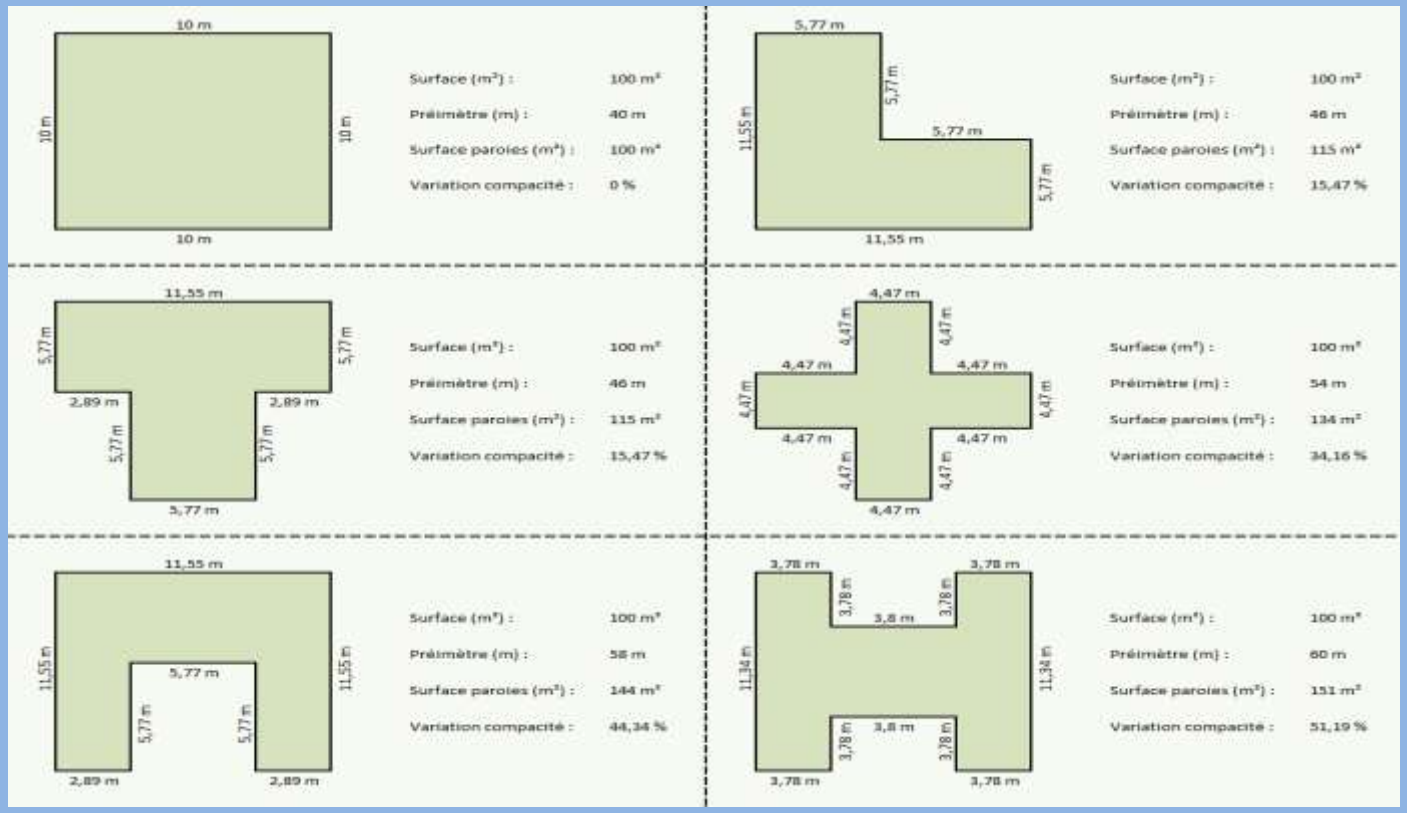
4)Systèmes de contrôle automatisés

5)Réduction de la consommation dans les bâtiments commerciaux

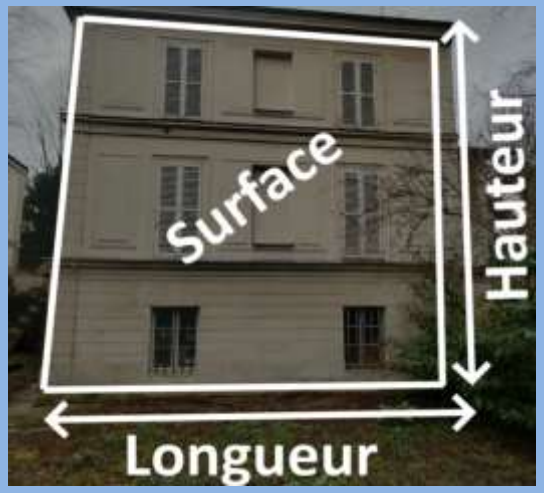
Attributs:

Le jeu de données contient huit attributs (ou **variables prédictives**), notés X1 à X8, et deux **variables cibles** notées y1 et y2.

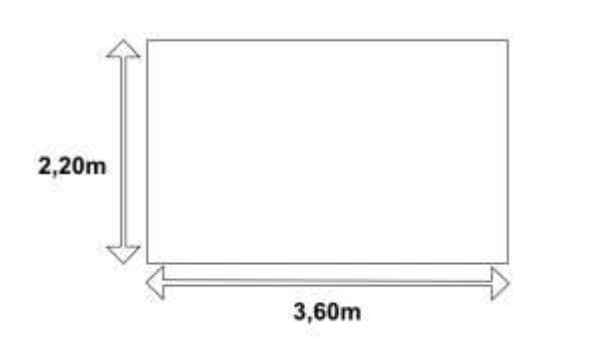
X1:Compacité relative (Relative Compactness) Sans unité



X2 :Surface de la façade (Surface Area) en m²



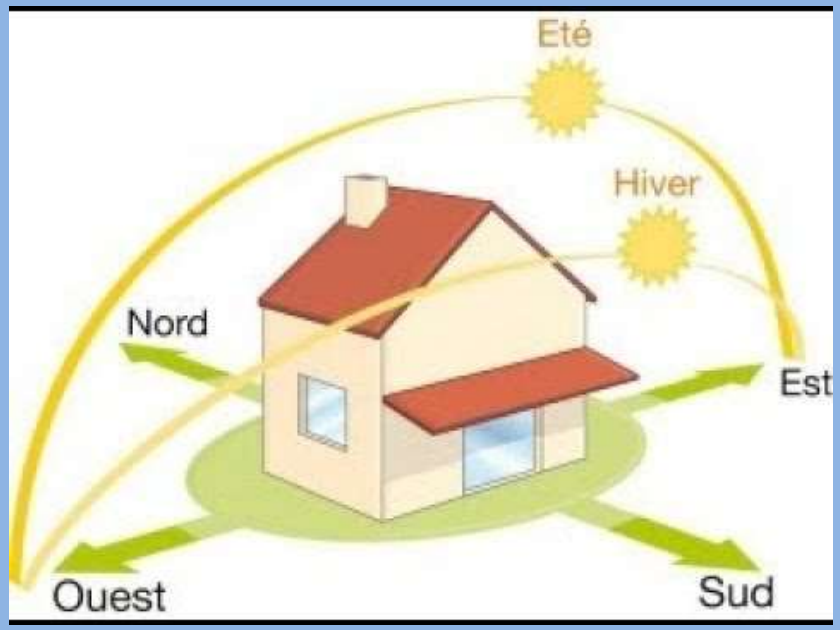
X 3 : Surface des murs (Wall Area) en m²



X 4 : Surface du toit (Roof Area) en m^2
La surface du toit détermine la quantité de chaleur qui peut être absorbée ou perdue par le toit. Un toit plus grand peut accroître les pertes de chaleur ou les gains thermiques selon la saison.

X 5 : Hauteur totale (Overall Height) en m

X 6 : Orientation (Orientation) Catégorique Nord, Sud, Est, Ouest



X 7: Surface vitrée (Glazing Area) en m^2
Une plus grande surface vitrée peut entraîner une augmentation des gains solaires mais aussi des pertes thermiques la nuit.

X 8 : Répartition de la surface vitrée (Glazing Area Distribution)
par exemple, répartie uniformément ou concentrée sur une seule façade

Les variables à prédire:

Y1:Charge de chauffage (Heating Load)

l'énergie nécessaire pour maintenir une température intérieure confortable(souvent fixée entre 20 et 22°C) en hiver

Y2:Charge de refroidissement (Cooling Load)

l'énergie nécessaire pour maintenir une température intérieure confortable (souvent fixée entre 20 et 22°C) en été

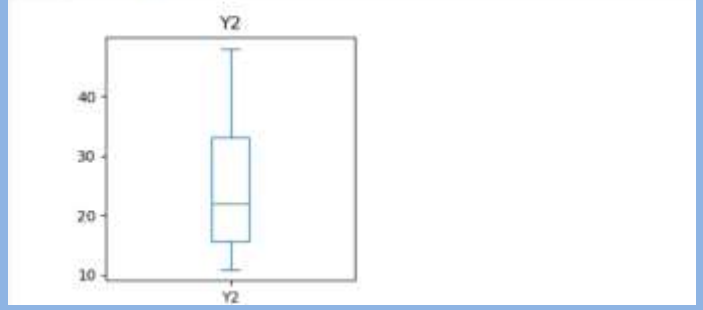
2-Entraînement d'un modèle

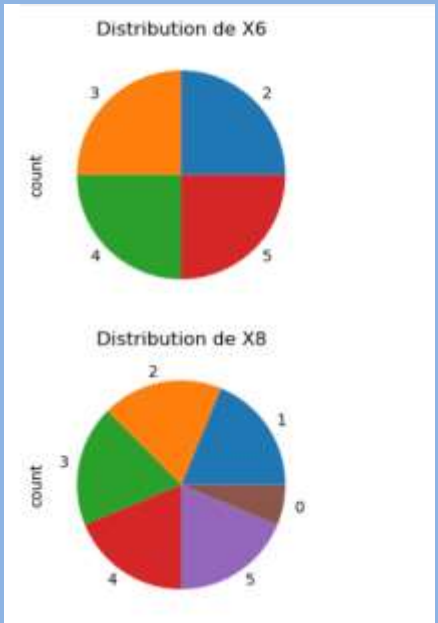
2-1-Préparation des Données

```
[12]: # Vérifier s'il y a des données manquantes
      data.isna().sum()
```

```
[12]: X1    0
      X2    0
      X3    0
      X4    0
      X5    0
      X6    0
      X7    0
      X8    0
      Y2    0
      dtype: int64
```

```
[22]: #Décrire les variables quantitative: identification des données aberrantes
      vars=['X1', 'X2', 'X3', 'X4', 'X5', 'X7',"Y2"]
      for col in vars:
          plt.figure(figsize=(3,3))
          plt.title(f"{col}")
          data[col].plot(kind='box')
          plt.show()
```





La variable X6 présente quatre catégories (nord, sud, est, ouest) avec des pourcentages égaux

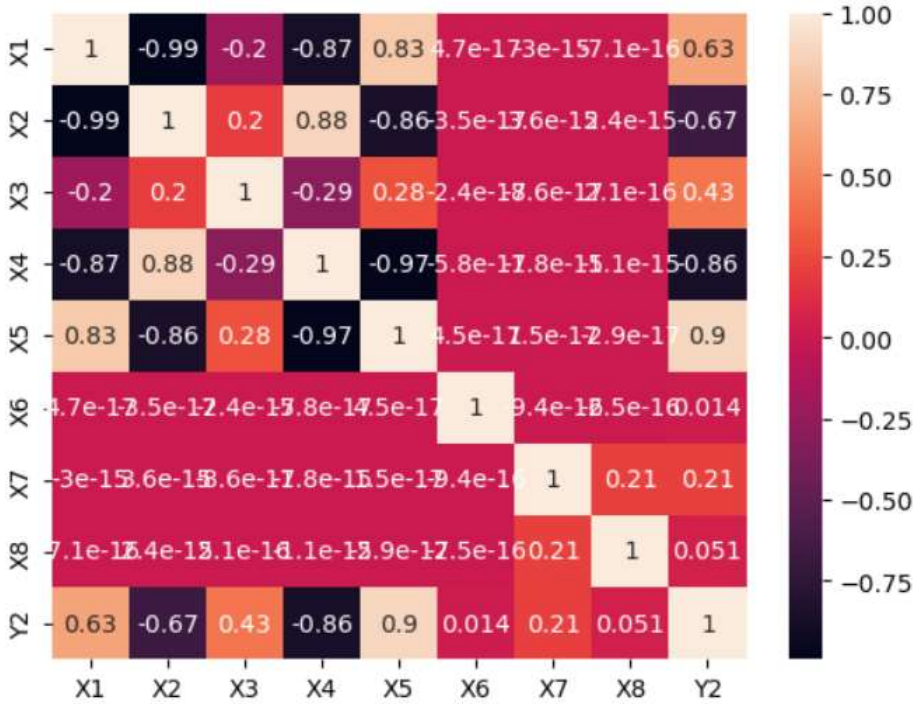
-Les deux variables X1 et X4 sont relativement symétrique (La moyenne et la médiane sont très proches)

```
## Afficher un résumé des données
data.describe(include="all")
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y2
count	768.000000	768.000000	768.000000	768.000000	768.00000	768.000000	768.000000	768.00000	768.000000
mean	0.764167	671.708333	318.500000	176.604167	5.25000	3.500000	0.234375	2.81250	24.587760
std	0.105777	88.086116	43.626481	45.165950	1.75114	1.118763	0.133221	1.55096	9.513306
min	0.620000	514.500000	245.000000	110.250000	3.50000	2.000000	0.000000	0.00000	10.900000
25%	0.682500	606.375000	294.000000	140.875000	3.50000	2.750000	0.100000	1.75000	15.620000
50%	0.750000	673.750000	318.500000	183.750000	5.25000	3.500000	0.250000	3.00000	22.080000
75%	0.830000	741.125000	343.000000	220.500000	7.00000	4.250000	0.400000	4.00000	33.132500
max	0.980000	808.500000	416.500000	220.500000	7.00000	5.000000	0.400000	5.00000	48.030000

```
#afficher la matrice de corrélation
sns.heatmap(data.corr(),annot=True)
##on remarque qu il y a une faible corrélatin entre X6,X8 et Y2
##une corrélation négative enter X2,X4 et Y1
##fort corrélation entre X5,X4 et Y1
```

```
<Axes: >
```



une forte corrélation entre X5 ,X4 et Y2

X1 et X5 influence négativement l'efficacité énergique

X4 et X2 influence positivement l'efficacité énergique

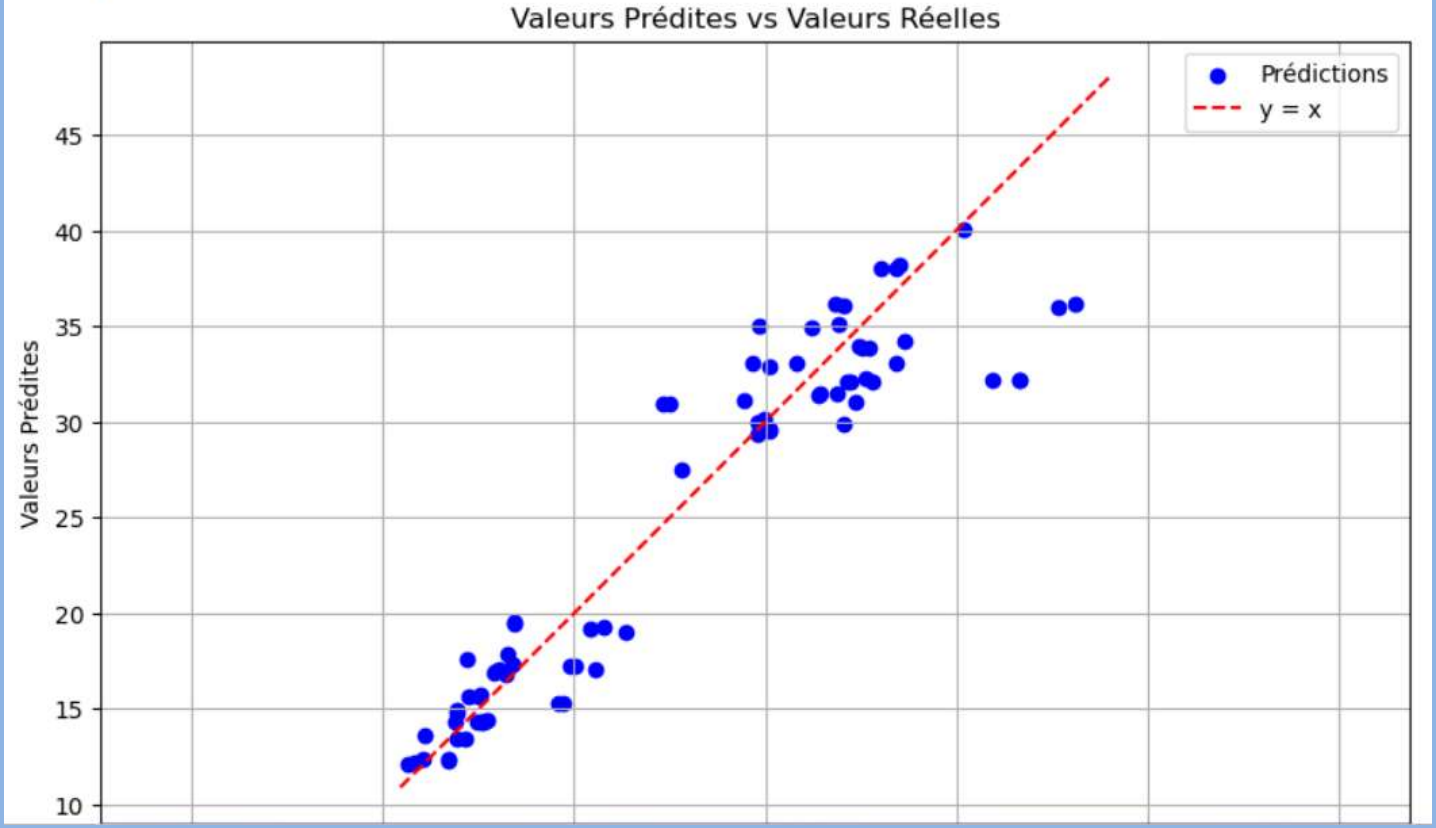
2-2-Entraînement d’un modèle

Plusieurs modèles seront entraînés sur les données. Après l'entraînement, une phase de sélection permettra d’identifier le modèle offrant les meilleures performances selon des métriques d’évaluation appropriées (par exemple, précision, MAE, RMSE, etc

Theil-Sen Regressor

```
# Définir la grille des paramètres pour Theil-Sen
param_grid_theil_sen = {
    'max_iter': [1000, 2000],  # Nombre d'itérations pour l'optimisation
    'tol': [1e-4, 1e-3],  # Tolérance pour la convergence
    'fit_intercept': [True, False]  # Si le modèle linéaire doit inclure un intercept
}

# Créer et entraîner le modèle Theil-Sen avec GridSearch
theil_sen = TheilSenRegressor()
grid_theil_sen = GridSearchCV(estimator=theil_sen, param_grid=param_grid_theil_sen, scoring='r2', cv=5)
grid_theil_sen.fit(x_train, y_train)

# Afficher les meilleurs paramètres de Theil-Sen
print("\nMeilleurs paramètres pour Theil-Sen Regressor :")
print(grid_theil_sen.best_params_)

# Évaluer le modèle avec les meilleurs paramètres
print("\nTheil-Sen Regressor:")
model_theil_sen = grid_theil_sen.best_estimator_
evaluate_model(model_theil_sen, x_train, y_train, x_test, y_test)
```

Theil-Sen Regressor:
Mean Absolute Error (MAE): 2.406592
Mean Squared Error (MSE): 11.965231
R-squared (R^2): 0.878909



Y1:

model	RANSAC Regressor	Theil-Sen Regressor	Ridge	Lasso	ElasticNet
MAE	2.348536	2.299419	2.340451	2.341232	2.337733
MSE	10.525493	9.796672	10.047241	10.044940	10.056338
R^2	0.904075	0.910717	0.908433	0.908454	0.908350

Y2:

model	RANSAC Regressor	Theil-Sen Regressor	Ridge	Lasso	ElasticNet
MAE	2.246098	2.406592	2.314548	2.314407	2.308553
MSE	11.656220	11.965231	11.556047	11.553883	11.556186
R^2	0.882036	0.878909	0.883050	0.883072	0.883049

3-Déploiement d'une machine Learning

la création d'une API avec Flask nécessite généralement un fichier Python pour configurer et démarrer l'application. Ce fichier contient le code nécessaire pour définir les routes de l'API, les fonctions de traitement des requêtes, et les réponses de l'API.

Importation des Bibliothèques :

```
from flask import Flask, request, jsonify, render_template
import joblib
import numpy as np
import pandas as pd
```

Création de l'Application et importation de modèles et des scaler

```
app = Flask(__name__)
# Load the Random Forest model
model1 = joblib.load('model1.pkl')
model2 = joblib.load('model2.pkl')

scaler1 = joblib.load('scaler1.pkl')  # Chargez le scaler pour le modèle 1
scaler2 = joblib.load('scaler2.pkl')  # Chargez le scaler pour le modèle 2
```

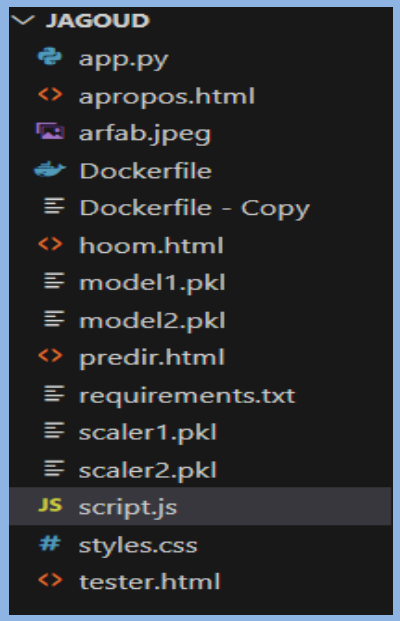
Définition des Routes

```python
@app.route('/')
def home():
    return render_template('hoom.html')
```

Fonction pour faire la prédiction

```python
def predict():
    # Récupérer les données JSON de la requête
    data = request.get_json(force=True)

    # Vérifier quel modèle utiliser
    model_type = data.get('model')
    data.pop('model', None)  # Supprimer le champ 'model' des données

    # Définir les variables binaires (pour les deux modèles)
    binary_variables = ["X6", "X8"]

    # Créer un DataFrame pour les variables non binaires
    non_binary_variables = {key: data[key] for key in data.keys() if key not in binary_variables}
    non_binary_df = pd.DataFrame(non_binary_variables, index=[0])

    # Faire la prédiction en fonction du modèle spécifié
    if model_type == 1:
        # Standardiser les variables non binaires
        dfstand = scaler1.transform(non_binary_df)
        dfstand = pd.DataFrame(dfstand, columns=non_binary_df.columns)

        # Ajouter les variables binaires au DataFrame standardisé
        binary_df = pd.DataFrame(data, index=[0])[binary_variables]
        Xs_test_final = pd.concat([dfstand, binary_df], axis=1)

        # Réorganiser les colonnes pour suivre l'ordre original
        column_order = ['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8']
        Xs = Xs_test_final[column_order]

        prediction1 = model1.predict(Xs)
        return jsonify({'prediction1': prediction1[0]})
```

En plus du fichier python, il y’a d'autres fichiers et dossiers dans la structure du projet:



le fichier **script.js** communique avec le serveur Flask configuré dans **app.py**. **Fetch API** dans script.js, il est possible d'envoyer des données vers l'API Flask et de recevoir des réponses. Cela permet à l'application web d'échanger des informations de manière asynchrone avec le serveur.

```
function predictModel1() {
    // Récupération des valeurs des champs d'entrée
    const data1 = {
        model: 1,
        X1: parseFloat(document.getElementById('X1').value),
        X2: parseFloat(document.getElementById('X2').value),
        X3: parseFloat(document.getElementById('X3').value),
        X4: parseFloat(document.getElementById('X4').value),
        X5: parseFloat(document.getElementById('X5').value),
        X6: parseFloat(document.getElementById('X6').value),
        X7: parseFloat(document.getElementById('X7').value),
        X8: parseFloat(document.getElementById('X8').value)
    };

    // Envoi de la requête POST pour /predict avec les données
    fetch('/predict', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data1)
    })
    .then(response => response.json())
    .then(data1 => {
        // Affichage du résultat de la prédiction
        document.getElementById('result1').innerText = "Prediction  : " + data1.prediction1;
    })
    .catch(error => console.error('Erreur:', error));
}
```

Les données de **Data1** du **modèle 1** sont récupérées dans le fichier HTML predict.html, avec des références pour chaque variable.

```html
<div  class="content" >
    <h2> <span style="color: #007BFF;">Prediction</span></h2>
    <form id="predictForm">
        <!-- Champs pour les valeurs X1 à X6 -->
        <div class="input-section">
            <div class="section-title">Input Variables</div>
            <label for="X1"> Relative Compactness:</label>
            <input type="number" id="X1" required>

            <label for="X2">Surface Area:</label>
            <input type="number" id="X2" required>

            <label for="X3">Wall Area:</label>
            <input type="number" id="X3" required>

            <label for="X4">Roof Area:</label>
            <input type="number" id="X4" required>

            <label for="X5">Overall Height:</label>
            <input type="number" id="X5" required>

            <label for="X6">Orientation:</label>
            <input type="number" id="X6" required>
            <label for="X6">Glazing Area:</label>
            <input type="number" id="X7" required>
            <label for="X6">Glazing Area Distribution:</label>
            <input type="number" id="X8" required>
        </div>
```

Un fichier Dockerfile :Il contient une série d'instructions qui définissent comment construire l'image Docker de l'application, en installant les dépendances et en configurant l'environnement pour que Flask fonctionne correctement.

```
FROM python:3.11

# Créer le répertoire de travail
WORKDIR /app

# Copier le fichier requirements.txt et installer les dépendances
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copier les fichiers nécessaires dans l'image
COPY app.py /app
COPY model1.pkl /app
COPY model2.pkl /app
COPY scaler1.pkl /app
COPY scaler2.pkl /app


# Copier les fichiers statiques dans le dossier static
RUN mkdir -p /app/static
COPY script.js /app/static/
COPY styles.css /app/static/
COPY arfab.jpeg /app/static/
# Copier les fichiers HTML dans le dossier templates
RUN mkdir -p /app/templates
COPY hoom.html /app/templates
COPY apropos.html /app/templates
COPY predir.html /app/templates
COPY tester.html /app/templates
```

Lien Github :

4-Présentation de l'Interface Utilisateur

-Lien vers API :

Cliquez ici

Conclusion

Ce projet a consisté à développer une application web avec une API dédiée à la prédiction des charges énergétiques des bâtiments, spécifiquement la charge de chauffage (Y1) et la charge de refroidissement (Y2). Le processus a impliqué plusieurs étapes clés:

Analyse des données

Développement de l'API

Interface utilisateur

Implémentation et déploiement