Université Moulay Ismail

Ecole National Supérieure des Arts et Métiers

Rapport de Projet :

Modélisation des Préférences avec la Méthode AHP

Master : Sciences des Données pour une Industrie Intelligente

Module : Analyse multicritère

Réalisé par :

- **NAAIMI Ayman**
- **IBRAHIMI Amine**
- **HARASHI Mustapha**
- **LOUZALI Abdelhamid**

Encadré par :

Pr. Dadda Afaf

Année universitaire : 2024/2025

Abstract

Ce projet a pour objectif de développer un système de prise de décision combinant des modèles d'apprentissage automatique et la méthode AHP (Analyse Hiérarchique des Processus) afin d'évaluer et de prioriser des critères en fonction de leur importance. Après une préparation des données, plusieurs modèles de régression, tels que la régression linéaire, les forêts aléatoires et les réseaux de neurones, ont été entraînés et comparés selon des métriques de performance comme le coefficient de détermination (R^2). Le modèle avec le meilleur score a été sélectionné pour extraire les importances des critères.

Ces importances ont ensuite été intégrées dans une matrice de comparaison construite selon la méthode AHP, permettant de hiérarchiser les critères tout en vérifiant la cohérence des jugements via des indicateurs tels que le ratio de cohérence (CR). Cette approche hybride combine l'efficacité des algorithmes d'apprentissage automatique pour l'analyse de données et la robustesse de la méthode AHP pour la prise de décision multicritère, offrant une solution rigoureuse et pratique pour des problématiques complexes d'évaluation et de priorisation.

Table des matières

Contexte et Objectives

La prise de décision dans un environnement complexe repose souvent sur des critères multiples et des jugements subjectifs. La méthode AHP (Analytic Hierarchy Process) est un outil populaire permettant de structurer ces problèmes en hiérarchisant les critères et en évaluant les alternatives. Cependant, cette méthode souffre parfois d'incohérences dans les jugements des décideurs.

L'objectif de ce projet est de proposer une solution visant à réduire l'incohérence liée à la subjectivité inhérente à la méthode AHP. Cette amélioration est réalisée grâce à l'introduction d'une matrice idéale, obtenue à l'aide de modèles supervisés.

Bibliothèques et Outils Utilisés

Pour réaliser les différentes étapes du projet, plusieurs bibliothèques Python ont été utilisées pour la manipulation des données, la modélisation, et les calculs spécifiques liés à la méthode AHP. Les principales bibliothèques sont les suivantes :

- **pandas :** Manipulation et analyse des données structurées.
- **numpy :** Calculs numériques avancés, manipulation de matrices.
- **scikit-learn :** Modélisation des données, apprentissage automatique (Random Forest, Gradient Boosting, SVM, etc.), et évaluation des performances des modèles.
- **matplotlib :** Visualisation des données et des résultats.
- **Nearest Neighbors (de scikit-learn) :** Génération de données synthétiques à l'aide d'une méthode proche de SMOTE.
- **MinMaxScaler et StandardScaler (de scikit-learn) :** Normalisation et mise à l'échelle des données.
- **MLPRegressor :** Réseaux de neurones pour les tâches de régression.

Ces outils ont permis de garantir une approche robuste et reproductible pour le traitement des données, la génération de données synthétiques, et l'évaluation des priorités selon la méthode AHP.

Méthodologie

I. Préparation de l'ensemble des données

1. Description du Dataset Original

Le dataset original, utilisé comme base pour ce projet, a été récupéré depuis Kaggle. Il contient des informations quantitatives et qualitatives permettant d'évaluer les performances des fournisseurs selon plusieurs critères clés.

voilà le lien du dataset :
https://www.kaggle.com/code/krnk97/best-supplier-model-weights-correlation-scores/notebook

Caractéristiques Principales :

- **Nombre de fournisseurs :** 35
- **Nom des colonnes** :
  - Nom du fournisseur
  - Qualité
  - Conditions et méthode de paiement
  - Flexibilité
  - Prix
  - Délai de livraison
- **Type de données** :
  - Données numériques pour des critères comme le prix et la qualité.

Aperçu des Données :

	supplier	quality	conditions and method of payment	flexibility	price	delivery time
0	1	4	3	4	5	2.0
1	2	2	2	3	7	3.0
2	3	2	3	4	8	4.0
3	4	4	3	5	7	1.0
4	5	3	3	2	7	2.0

Analyse Préliminaire :

- Les données révèlent une grande variabilité dans les performances des fournisseurs selon les critères.
- Certains critères comme la qualité et le prix sont corrélés de manière significative avec les scores globaux.
- Une normalisation est nécessaire pour uniformiser les échelles avant l'application de méthodes comme la régression ou la génération de données synthétiques.

Prétraitement des Données :

Avant d'utiliser ces données pour modéliser les préférences et appliquer la méthode AHP :

1. **Normalisation des données quantitatives** :
    - Les données ont été normalisées à l'aide de **`MinMaxScaler`**(échelle entre 0 et 1) pour garantir une interpolation réaliste et équilibrée lors de la génération de données synthétiques avec la méthode SMOTE-like.
2. **Gestion des données manquantes** :
    - Aucun traitement particulier n'était requis, car le dataset était complet.

```
# Préparation des données quantitatives pour la génération
# Sélection des colonnes quantitatives (toutes sauf "supplier")
quantitative_cols = df.columns[1:]

# Normalisation des données quantitatives avec MinMaxScaler
# Cela met les données à l'échelle [0, 1] pour faciliter la génération.
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df[quantitative_cols])

# Conversion en DataFrame pandas pour une manipulation plus facile
pd.DataFrame(scaled_data)
```

2. Génération de Données Synthétiques

Afin d'améliorer la robustesse de l'analyse, des données synthétiques ont été générées en utilisant une méthode similaire à SMOTE (Synthetic Minority Over-sampling Technique). Cette méthode génère des points synthétiques en interpolant entre des échantillons réels voisins.

1. **Méthodologie SMOTE-like :**
    - La méthode utilise des échantillons existants et leurs voisins proches (trouvés via KNN) pour générer des points synthétiques entre ces échantillons.
    - **Nombre d'échantillons** : Le jeu de données a été augmenté pour atteindre une taille cible de 1000 lignes.
2. **Réintégration des Données Synthétiques :**
    - Les données synthétiques ont été retransformées à l'échelle originale grâce à l'inverse du `MinMaxScaler`.
    - Les échantillons synthétiques ont été fusionnés avec les données originales.

Code Utilisé :

```
# Définition d'une méthode similaire à SMOTE pour générer des données synthétiques
def generate_synthetic_data(data, target_size):
    # Initialisation des paramètres
    n_samples, n_features = data.shape
    n_synthetic = target_size - n_samples
    synthetic_data = []

    # Utilisation de KNN pour trouver les voisins les plus proches
    knn = NearestNeighbors(n_neighbors=5).fit(data)
    for _ in range(n_synthetic):
        # Sélection aléatoire d'un point existant
        idx = np.random.randint(0, n_samples)
        sample = data[idx]

        # Trouver les indices des voisins du point sélectionné
        neighbors = knn.kneighbors([sample], return_distance=False)[0]
        neighbor_idx = np.random.choice(neighbors)
        neighbor_sample = data[neighbor_idx]

        # Génération d'un point synthétique entre le point et un voisin
        alpha = np.random.rand()  # Générer un facteur aléatoire entre 0 et 1
        synthetic_point = sample + alpha * (neighbor_sample - sample)
        synthetic_data.append(synthetic_point)
```

```
    return np.array(synthetic_data)

# Générer les nouvelles données synthétiques jusqu'à atteindre la taille cible
target_size = 1000  # Taille cible totale du DataFrame
synthetic_scaled_data = generate_synthetic_data(scaled_data, target_size)

# Revenir à l'échelle d'origine des données
synthetic_data = scaler.inverse_transform(synthetic_scaled_data)

# Création d'un DataFrame pour les données synthétiques
synthetic_df = pd.DataFrame(synthetic_data, columns=quantitative_cols)

# Ajouter une colonne "supplier" pour identifier les données synthétiques
synthetic_df.insert(0, 'supplier', range(len(df) + 1, target_size + 1))

# Fusionner les données d'origine et les données synthétiques
# Cela combine le DataFrame existant avec les nouvelles données générées.
df = pd.concat([df, synthetic_df], ignore_index=True)

# Afficher les dimensions finales du DataFrame fusionné
print(f"Nbr de lignes = {df.shape[0]}")  # Nombre total de lignes
print(f"Nbr de colonnes = {df.shape[1]}")  # Nombre total de colonnes
```

```
Nbr de lignes = 1000
Nbr de colonnes = 6
```

Dataset après la génération :

	supplier	quality	conditions and method of payment	flexibility	price	delivery time
0	1	4	3	4	5	2.0
1	2	2	2	3	7	3.0
2	3	2	3	4	8	4.0
3	4	4	3	5	7	1.0
4	5	3	3	2	7	2.0
...	...	...	...	...	...	...
995	996	3	3	3	4	3.0
996	997	3	3	4	4	4.0
997	998	3	2	2	7	1.0
998	999	5	4	4	5	2.0
999	1000	2	4	4	7	4.0

Les données synthétiques sont ensuite normalisées et fusionnées avec les données originales pour créer un ensemble de données enrichi.

3. Normalisation des données pour l'entraînement

1. **Normalisation** : Les données ont été normalisées avec **MinMaxScaler** pour uniformiser les caractéristiques sur une échelle de 0 à 1, améliorant ainsi la performance des modèles. La colonne "supplier" a été exclue, et la moyenne des valeurs normalisées par ligne a été calculée comme cible pour l'entraînement.
2. **Variable Cible** :
    - La moyenne des valeurs normalisées des critères a été calculée et utilisée comme étiquette (score) pour l'entraînement des modèles.

Code Utilisé :

```
# Normaliser les données pour une meilleure performance des modèles, qui garantit que toutes les caractéristiques sont sur la même
# On sépare les caractéristiques (features) de la cible "supplier"
X = df.drop('supplier', axis=1)

# Appliquer MinMaxScaler aux données (normalisation entre 0 et 1)
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
# X_scaled = (X - min(X)) / (max(X) - min(X))

# Calculer la moyenne de chaque ligne comme cible "y" pour l'entraînement
y = X_scaled.mean(axis=1)

# Affichage des données normalisées sous forme de DataFrame pour vérification
pd.DataFrame(X_scaled)
```

4. Construction de la variable cible

Un modèle **Random Forest Regressor** a été utilisé pour prédire le score des fournisseurs. Ce modèle, basé sur un ensemble d'arbres de décision, offre plusieurs avantages, notamment la réduction du surapprentissage et la capacité à capturer des relations non linéaires dans les données.

Après l'entraînement sur les données normalisées, le modèle a été utilisé pour générer des prédictions des scores, qui ont ensuite été intégrés dans le DataFrame final. Cette approche robuste a permis une évaluation précise et fiable des performances des fournisseurs.

Les données ont été divisées en ensembles d'entraînement (80%) et de test (20%).

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Train a Random Forest Regressor
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Predict scores for the dataset
df['score'] = model.predict(X_scaled)
```

5. Obtention du Dataset finale

Après la réalisation des étapes précédentes, on a obtenu un ensemble de données prêt pour l'application de l'**AHP** :

supplier	quality	conditions and method of payment	flexibility	price	delivery time	score
1	4	3	4	5	2	0,526666667
2	2	2	3	7	3	0,396666667
3	2	3	4	8	4	0,593333333
4	4	3	5	7	1	0,633333333
5	3	3	2	7	2	0,406666667
6	4	1	2	8	0,5	0,345733333
7	3	1	4	5	2	0,36
8	3	1	3	6	4	0,413333333
9	5	4	5	6	3	0,79
10	4	4	3	7	1	0,55

Cet ensemble de données a été obtenu après la génération de données synthétiques plusieurs fois afin d'atteindre la meilleure dataset ayant le mse (Erreur Quadratique Moyenne).

MSE = 6.949794444444544e-06.

Ce MSE qui est très proche de 0, reflète la précision du modèle pour prédire les scores des fournisseurs.

voila le dataset après la normalisation et la sélection des features (X) et de la variable cible (y):

	0	1	2	3	4
0	0.666667	0.50	0.666667	0.4	0.4
1	0.000000	0.25	0.333333	0.8	0.6
2	0.000000	0.50	0.666667	1.0	0.8
3	0.666667	0.50	1.000000	0.8	0.2
4	0.333333	0.50	0.000000	0.8	0.4
...	...	...	...	...	...
995	0.333333	0.50	0.333333	0.2	0.6
996	0.333333	0.50	0.666667	0.2	0.8
997	0.333333	0.25	0.000000	0.8	0.2
998	1.000000	0.75	0.666667	0.4	0.4
999	0.000000	0.75	0.666667	0.8	0.8

II. Application sur l'AHP

1. Entraînement et Évaluation les modèles

Plusieurs algorithmes de régression ont été testés pour prédire les scores des critères, notamment:

- Random Forest Regressor
- Gradient Boosting Regressor
- Régression Linéaire
- Support Vector Regressor (SVR)
- Decision Tree Regressor
- Réseau de Neurones (MLPRegressor)

```python
# Division des données en ensembles d'entraînement et de test (80% / 20%)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Fonction pour entraîner et évaluer différents modèles
def train_and_evaluate_model(model, model_name):
    # Entraîner le modèle sur les données d'entraînement
    model.fit(X_train, y_train)

    # Faire des prédictions sur l'ensemble de test
    y_pred = model.predict(X_test)

    # Calculer l'erreur quadratique moyenne (MSE)
    mse = mean_squared_error(y_test, y_pred)

    # Calculer le coefficient de détermination (R²)
    r2 = r2_score(y_test, y_pred)

    # Retourner le modèle entraîné, ainsi que les métriques de performance (MSE et R²)
    return model, mse, r2

# Entraîner plusieurs modèles avec différents algorithmes de régression
models = {
    "Random Forest": RandomForestRegressor(random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42),
    "Linear Regression": LinearRegression(),
    "Support Vector Regressor": SVR(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Neural Network": MLPRegressor(random_state=42, max_iter=1000)
}
# Initialisation d'un dictionnaire 'results' pour stocker les résultats des modèles
results = {}

# Boucle pour entraîner chaque modèle et évaluer ses performances
for model_name, model in models.items():
    # Entraîner et évaluer le modèle
    trained_model, mse, r2 = train_and_evaluate_model(model, model_name)

    # Stocker les résultats du modèle dans le dictionnaire 'results'
    results[model_name] = {
        "model": trained_model,
        "mse": mse,
        "r2": r2
    }
```

Une fonction générique a été mise en place pour entraîner et évaluer les performances de chaque modèle en utilisant deux métriques principales :

- Erreur Quadratique Moyenne (MSE) : mesure la précision des prédictions. Une valeur faible est préférable.
- Coefficient de Détermination (R^2) : indique la qualité d'ajustement du modèle aux données. Une valeur proche de 1 est idéale.

Les résultats des performances ont été comparés dans un tableau. Le modèle ayant obtenu le meilleur score R^2 a été sélectionné pour l'analyse des critères. Ce processus permet d'identifier le modèle le plus performant pour effectuer des prédictions fiables.

index	mse	r2
Random Forest	3.0189257083332538e-06	0.9998349786606537
Gradient Boosting	6.081596706212509e-05	0.9966756610437534
Linear Regression	6.955215961173358e-06	0.9996198120907094
Support Vector Regressor	0.0024482564653513 1	0.8661727382492174
Decision Tree	1.8151889725849698e-32	1.0
Neural Network	0.000422983031222281	0.9768787863376883

2. Sélection le meilleur modèle

Le choix du modèle à utiliser pour la détermination de la matrice de comparaison est basé sur le coefficient de détermination R^2 .

```
# Sélectionner le meilleur modèle basé sur le coefficient de détermination (R²)
# Le modèle avec le score R² le plus élevé est considéré comme le meilleur.
best_model_name = max(results, key=lambda x: results[x]['r2'])

# Récupérer le modèle entraîné correspondant au meilleur score R²
best_model = results[best_model_name]['model']

#Afficher le meilleur modèle pour Dataset
best_model
```

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

Dans notre cas, le meilleur modèle est **Decision Tree Regressor** .

3. Analyse de l'Importance des Critères

Pour le modèle sélectionné, l'importance des caractéristiques a été analysée afin de déterminer les critères ayant le plus d'influence sur le score des fournisseurs. Ces informations ont été utilisées pour générer une matrice cohérente basée sur l'analyse AHP (Analytic Hierarchy Process).

```
# Extraire l'importance des caractéristiques en fonction du modèle sélectionné
# Récupérez les noms des autres colonnes restantes en tant que caractéristiques
features = X.columns

feature_importances = (
    best_model.feature_importances_ if hasattr(best_model, 'feature_importances_') else np.abs(best_model.coef_)
    if hasattr(best_model, 'coef_') else np.ones(len(features))  # Cas où les importances ne sont pas disponibles
)

# Créer un DataFrame avec les caractéristiques et leurs importances respectives
importance_df = pd.DataFrame({'Criterion': features, 'Importance': feature_importances})

# Afficher le DataFrame des priorités
importance_df
```

index	Criterion	Importance
0	quality	0.3918820735816186
1	conditions and method of payment	0.22840407854227404
2	flexibility	0.16147218320429757
3	price	0.07274905684786068
4	delivery time	0.1454926078239492

Visualisation des importances : Un graphique en barres a été généré pour visualiser l'importance relative des différents critères, classés par ordre décroissant.

4. Matrice de Comparaison des Critères

La méthode Analytic Hierarchy Process (AHP) a été appliquée pour construire une matrice de comparaison cohérente à partir des importances des critères :

1. **Construction de la matrice cohérente :** Une matrice carrée a été générée, où chaque élément est le rapport des importances entre deux critères, arrondi à l'échelle standard de l'AHP.
2. **Calcul des priorités relatives :** Les priorités relatives ont été calculées en normalisant la matrice et en prenant la moyenne des lignes. Cela permet d'obtenir une pondération pour chaque critère.

```python
# Approxime une valeur donnée à l'échelle AHP
def ahp_scale(value):
    scale = [1/9, 1/7, 1/5, 1/3, 1, 3, 5, 7, 9]  # Échelle AHP standard
    closest = min(scale, key=lambda x: abs(x - value))  # Trouver la valeur la plus proche
    return closest

# Génère une matrice cohérente selon la méthode AHP
def generate_ahp_matrix(priorities):
    n = len(priorities)  # Nombre de critères
    matrix = np.zeros((n, n))  # Initialiser une matrice vide de taille (n, n)

    for i in range(n):
        for j in range(n):
            if i == j:
                # La diagonale principale doit être égale à 1 (auto-comparaison)
                matrix[i, j] = 1
            else:
                # Calculer le ratio des priorités pour i et j
                ratio = priorities[i] / priorities[j]
                # Approximer le ratio à l'échelle AHP
                matrix[i, j] = ahp_scale(ratio)
                # Assurer la réciprocité pour la case symétrique
                matrix[j, i] = round(1 / matrix[i, j], 2)

    return matrix

# Extraire les priorités à partir de importance_df1
priorities = importance_df['Importance'].values

# Générer la matrice cohérente en utilisant les priorités relatives
consistent_matrix = generate_ahp_matrix(priorities)

# Créer un DataFrame pour afficher la matrice cohérente avec des noms de critères comme index et colonnes
consistent_matrix_df = pd.DataFrame(consistent_matrix, index=features, columns=features)

# Afficher la matrice cohérente
pd.DataFrame(consistent_matrix_df)
```

	quality	conditions and method of payment	flexibility	price	delivery time
quality	1.000000	3.000000	3.000000	5.0	3.0
conditions and method of payment	0.333333	1.000000	1.000000	3.0	3.0
flexibility	0.333333	1.000000	1.000000	3.0	1.0
price	0.200000	0.333333	0.333333	1.0	1.0
delivery time	0.333333	0.333333	1.000000	1.0	1.0

5. Analyse AHP et cohérence

Une matrice cohérente a été générée à partir des priorités relatives dérivées des importances des caractéristiques. L'indice de cohérence (IC) et le ratio de cohérence (CR) ont été calculés pour

évaluer la fiabilité des jugements. Les résultats ont montré que la matrice était cohérente, avec un CR inférieur au seuil de 0,1.

Le **Lambda Max (λmax)** a été calculé pour mesurer la consistance des jugements.

L'**Indice de Cohérence (IC)** et le **Ratio de Cohérence (CR)** ont été évalués :

```python
# Fonction pour calculer les priorités relatives à partir de la matrice de comparaison AHP
def calculate_priorities(matrix):
    column_sums = matrix.sum(axis=0)
    normalized_matrix = matrix / column_sums

    # Calculer la moyenne de chaque ligne pour obtenir les priorités relatives
    priorities = normalized_matrix.mean(axis=1)
    return priorities

# Fonction pour calculer λ_max, qui mesure la consistance de la matrice
def calculate_lambda_max(matrix, priorities):
    # Calculer la somme pondérée de la matrice en utilisant les priorités
    weighted_sum = matrix.dot(priorities)
    # Calculer λ_max comme la moyenne de la division de la somme pondérée par les priorités
    lambda_max = (weighted_sum / priorities).mean()
    return lambda_max

# Fonction pour calculer l'indice de cohérence (IC) et le ratio de cohérence (CR)
def calculate_consistency(matrix, priorities):
    n = matrix.shape[0]  # Nombre de critères (taille de la matrice)
    lambda_max = calculate_lambda_max(matrix, priorities)  # Calcul de λ_max
    # Calcul de l'indice de cohérence (IC) en fonction de λ_max et de la taille de la matrice
    IC = (lambda_max - n) / (n - 1)

    # Définir les valeurs d'indices aléatoires (RI) pour différentes tailles de matrice
    IA = {1: 0.00, 2: 0.00, 3: 0.58, 4: 0.90, 5: 1.12, 6: 1.24, 7: 1.32, 8: 1.41, 9: 1.45, 10: 1.49, 11: 1.51}
    RI = IA.get(n)
    # Calcul du ratio de cohérence (CR)
    CR = IC / RI if RI != 0 else 0
    return IC, CR

# Appliquer les calculs sur la matrice de comparaison cohérente
priorities = calculate_priorities(consistent_matrix_df)
IC, CR = calculate_consistency(consistent_matrix_df, priorities)

# Afficher les résultats
print(f"Lambda max (λmax) : \t\t {calculate_lambda_max(consistent_matrix_df, priorities):.3f}")
print(f"Indice de cohérence (IC) : \t {IC:.3f}")
print(f"Ratio de cohérence (CR) : \t {CR:.3f}\n")

# Vérifier la cohérence de la matrice en fonction du ratio de cohérence
if CR < 0.1:
    print("➡️ La matrice est cohérente :)")
else:
    print("➡️ La matrice n'est pas cohérente. Veuillez vérifier les jugements :(")
```

```
Lambda max (λmax) :             5.187
Indice de cohérence (IC) :      0.047
Ratio de cohérence (CR) :       0.042

➡️ La matrice est cohérente :)
```

Dans notre cas CR= 0.042<0.1 Donc la matrice est cohérente.

III. Résultats et Implémentation Technique

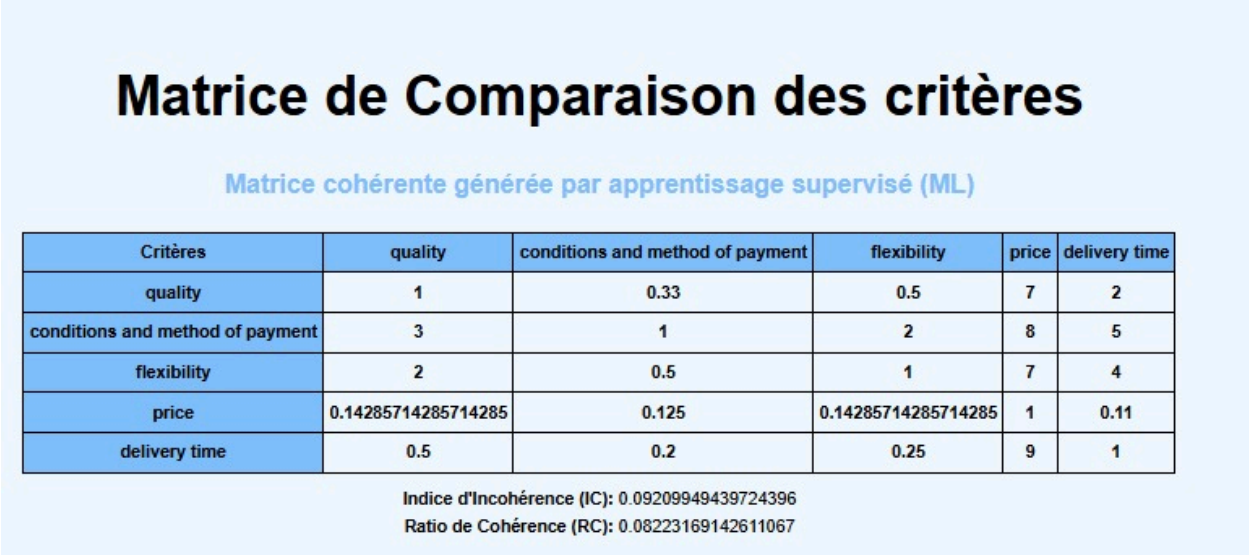
1. Description de l'interface

Une interface utilisateur a été développée avec **Django** pour permettre une interaction conviviale avec les matrices de comparaison. Cette interface offre trois fonctionnalités principales :

1. **Affichage d'une matrice générée automatiquement** : Une matrice cohérente est calculée et présentée, basée sur un modèle supervisé.
2. **Saisie des valeurs pour une matrice ajustable** : Les utilisateurs peuvent modifier les valeurs de la matrice selon leurs jugements.
3. **Validation de la cohérence** : Les indices d'incohérence (IC) et le ratio de cohérence (RC) sont calculés pour la matrice finale.

Aperçu de l'Interface :

- Une matrice cohérente générée par apprentissage supervisé.



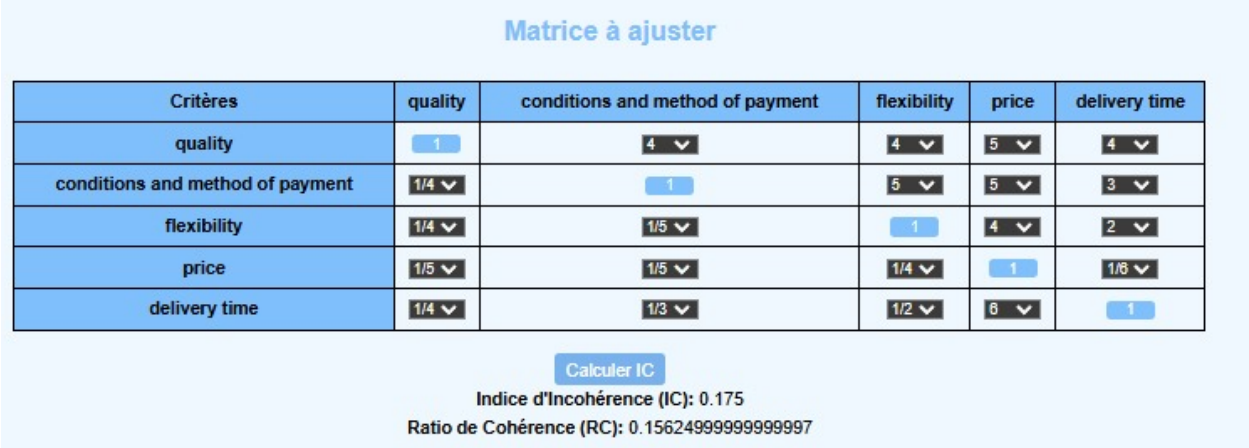
Matrice de Comparaison des critères

Matrice cohérente générée par apprentissage supervisé (ML)

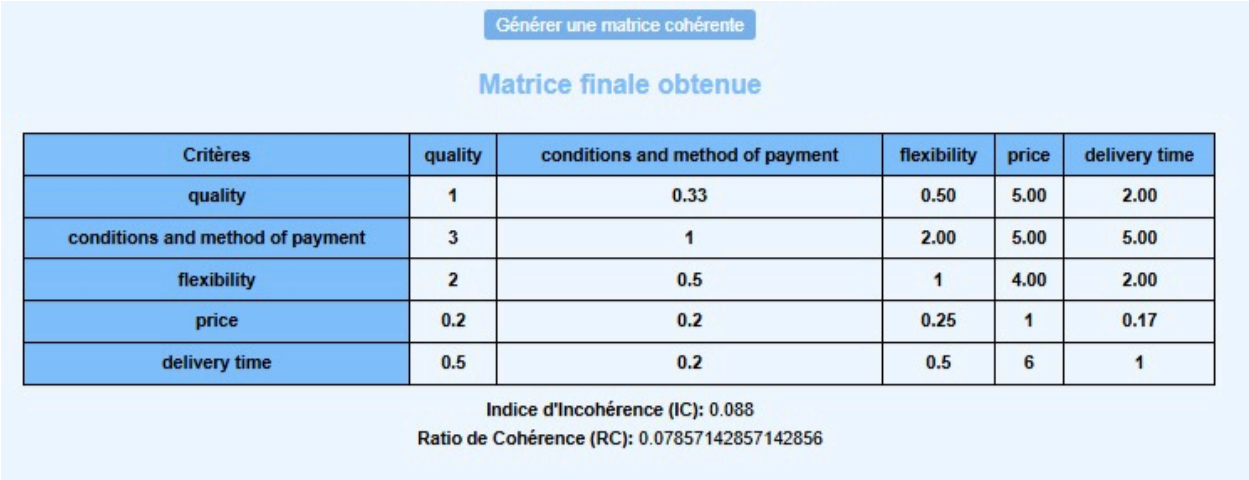
Critères	quality	conditions and method of payment	flexibility	price	delivery time
quality	1	0.33	0.5	7	2
conditions and method of payment	3	1	2	8	5
flexibility	2	0.5	1	7	4
price	0.14285714285714285	0.125	0.14285714285714285	1	0.11
delivery time	0.5	0.2	0.25	9	1

Indice d'Incohérence (IC): 0.09209949439724396
Ratio de Cohérence (RC): 0.08223169142611067

- Une matrice ajustable par l'utilisateur pour modifier les jugements.



- La matrice finale obtenue après ajustement et validation de la cohérence.



2. Algorithme de Fusion des Matrices et Génération de Matrice Cohérente

Pour garantir la cohérence de la matrice finale, un algorithme en **JavaScript** a été implémenté. Celui-ci combine les données d'une matrice ajustable par l'utilisateur et d'une matrice externe générée automatiquement.

Fonctionnement de l'Algorithme :

- **Préférence aux valeurs ajustables :** L'algorithme privilégie les données de la matrice saisie par l'utilisateur, garantissant une personnalisation des jugements.
- **Incorporation de valeurs externes :** Lorsque nécessaire, des données de la matrice générée automatiquement sont intégrées pour assurer la complétion et la cohérence.

- **Réciprocité des jugements :** Les matrices respectent la règle de réciprocité (a[i][j] = 1/a[j][i]).
- **Validation de la cohérence :** L'indice de cohérence (IC) est calculé, et le processus itératif se poursuit jusqu'à ce que l'IC atteigne un niveau acceptable (IC < 0.1).

Code Utilisé :

Voici le code de l'algorithme utilisé pour mixer les matrices et générer une matrice cohérente :

```
const generateConsistentMatrix = () => {
    if (!matrice) {
      alert("External Matrice not loaded yet.");
      return;
    }
    let consistentMatrix = [];
    let ic = 1.0;
    while (ic / 1.12 >= 0.1) {
      const newMatrix = matrix.map((row, i) => [...row]); // Clone the editable matrix
      let editableCount = 0;
      let externalCount = 0;

      // Try to build a consistent matrix using mainly editable values
      for (let i = 0; i < matrixSize; i++) {
        for (let j = 0; j < matrixSize; j++) {
          if (i !== j) {
            let source;

            // Give preference to editable values
            if (editableCount < matrixSize - 1) { // Allow most of the matrix to be filled with editable
              source = "editable";
            } else if (externalCount < matrixSize) {
              source = "external"; // Use external values only when necessary
            } else {
              // Once most of the matrix is filled, we can randomly choose between the two
              source = Math.random() < 0.5 ? "editable" : "external";
            }
```

```
            if (source === "editable") {
              newMatrix[i][j] = matrix[i][j];
              editableCount++;
            } else {
              newMatrix[i][j] = parseFloat(matrice[criteria[i]][criteria[j]]);
              externalCount++;
            }
            newMatrix[j][i] = (1 / newMatrix[i][j]).toFixed(2); // Ensure reciprocity
          }
        }
      }

      // Recalculate IC for the newly generated matrix
      ic = calculateIC(newMatrix);
      consistentMatrix = newMatrix; // Update consistentMatrix
    }

    // After exiting the loop, set the consistent matrix and IC
    setConsistentMatrix(consistentMatrix);
    setICConsistent(ic);

  };
```

Résultats :

L'algorithme permet de générer une matrice cohérente en respectant les jugements personnalisés tout en garantissant un indice de cohérence acceptable. Cela offre une solution flexible et automatisée pour combiner les préférences humaines et les résultats calculés.

Conclusion et Perspectives

Ce projet a démontré l'application pratique de la méthode AHP combinée à des techniques de Machine Learning pour la prise de décision multicritère.

- **Conclusion** : La méthode AHP a permis de hiérarchiser efficacement les critères, tandis que les modèles d'apprentissage ont fourni une évaluation précise des performances des fournisseurs.
- **Perspectives** :
    - Intégrer des critères supplémentaires pour enrichir l'analyse.
    - Automatiser le processus pour une utilisation à grande échelle.
    - Améliorer l'interface utilisateur.