FSBM
FACULTE DES SCIENCES BEN M'SICK
UNIVERSITÉ HASSAN II DE CASABLANCA

RAPPORT DE PROJET DE FIN D'ÉTUDES – MODÈLE DE DEEP LEARNING

FACULTÉ DES SCIENCES BEN M'SIK – UNIVERSITÉ HASSAN II DE CASABLANCA

S6 – Licence d'Excellence en Intelligence Artificielle

SYSTÈME DE RECOMMANDATION IMMOBILIÈRE BASÉ SUR LE DEEP LEARNING ET LA TRIPLET LOSS

Encadrés par : Mr. El Habib BENLAHMAR

Mr. Oussama KAICH

Réalisés par : Khadija RHAOUFAL

Ouiam MAKBOUL

Abderrahmane SABOR

Année scolaire : 2024 / 2025

Table des matières

Table des figures :

Chapitre 1 : Introduction

1. Contexte du projet

Dans un monde en constante évolution technologique, l'intelligence artificielle occupe une place de plus en plus centrale dans la transformation des secteurs traditionnels. Le domaine de l'immobilier n'échappe pas à cette révolution. Les plateformes en ligne de vente ou de location de biens immobiliers se multiplient, rendant le processus de recherche plus accessible, mais aussi plus complexe face à l'abondance de données. Face à cette surcharge d'informations, les utilisateurs ont besoin d'outils intelligents capables de leur proposer des résultats pertinents et personnalisés. C'est dans ce contexte que les systèmes de recommandation jouent un rôle essentiel, en facilitant la mise en relation entre l'utilisateur et les biens correspondant réellement à ses préférences.

Avec les progrès du deep learning, ces systèmes peuvent désormais dépasser les simples filtres de recherche (prix, surface, localisation, etc.) pour exploiter des relations plus complexes entre les différents attributs des biens immobiliers. L'approche par apprentissage de représentations (ou "embeddings") permet ainsi de cartographier chaque maison dans un espace vectoriel où la proximité reflète la similarité, rendant possible la recommandation de biens similaires, même si les critères exacts ne sont pas explicitement définis par l'utilisateur.

2. Problématique et motivations

Les systèmes classiques de recommandation dans le domaine immobilier s'appuient généralement sur des méthodes basées sur des règles ou sur des algorithmes de filtrage collaboratif, qui ont montré leurs limites en matière de personnalisation et de flexibilité. Ces systèmes ne capturent souvent que des relations superficielles entre les biens, ce qui peut conduire à des suggestions peu pertinentes. De plus, dans les cas où l'utilisateur n'interagit pas fréquemment avec la plateforme (problème de "cold start"), les algorithmes collaboratifs deviennent inefficaces. Pour pallier ces limitations, une approche basée sur des réseaux de neurones profonds, et plus précisément sur l'apprentissage de similarité à l'aide de la **Triplet Loss**, permet de mieux comprendre les relations sémantiques entre les logements.

L'idée est de projeter chaque bien immobilier dans un espace vectoriel où les distances reflètent la proximité sémantique : deux maisons aux caractéristiques similaires auront des vecteurs proches, tandis que des maisons différentes seront éloignées. Cette modélisation permet ainsi de proposer des recommandations plus fines, basées sur la notion de similarité globale et non uniquement sur des filtres explicites.

3. Objectifs et contributions

Ce projet a pour ambition de concevoir un système de recommandation de maisons s'appuyant sur des techniques avancées d'apprentissage profond. Les objectifs principaux sont les suivants :

- Prétraitement des données : nettoyer et analyser un dataset immobilier afin d'en extraire les caractéristiques les plus pertinentes pour la modélisation.

- Apprentissage de représentations : utiliser un réseau de neurones pour apprendre une représentation vectorielle de chaque bien immobilier.
- Entraînement avec Triplet Loss : mettre en œuvre une fonction de perte spécifique (Triplet Loss) afin d'optimiser la capacité du modèle à différencier des maisons similaires de celles qui ne le sont pas.
- Recommandation : développer deux stratégies de recommandation — l'une basée sur un identifiant de maison (recherche de biens similaires à une maison donnée), et l'autre basée sur des critères utilisateurs.
- Évaluation du système : évaluer la performance du système à l'aide de métriques rigoureuses, telles que la Top-k Accuracy et le Mean Reciprocal Rank (MRR), pour mesurer la pertinence des recommandations proposées.

Chapitre 2 : Présentation et Analyse des Données

1. Description du jeu de données:

Le dataset étudié est un fichier CSV nommé Housing.csv contenant 21 613 entrées, chacune représentant une maison vendue avec ses caractéristiques principales. Il comprend notamment les colonnes suivantes : id, price (prix de vente), bedrooms (nombre de chambres), bathrooms (nombre de salles de bain), sqft_living (surface habitable en pieds carrés), floors (nombre d'étages), waterfront (vue sur l'eau), view (vue générale), et condition (état général du logement).

Ces données offrent une diversité de caractéristiques, à la fois quantitatives (comme price, sqft_living, bedrooms) et catégorielles (view, waterfront, condition), qui influencent la valeur et la perception de similarité entre les logements. Une bonne compréhension de cette structure est essentielle pour entraîner un modèle capable de suggérer des maisons similaires selon un bien de référence.

2. Analyse exploratoire des données (EDA)

L'analyse exploratoire des données (EDA) constitue une étape fondamentale dans tout projet de science des données, car elle permet de mieux comprendre la structure, la distribution et les relations présentes dans le jeu de données. Avant de construire un modèle prédictif ou de passer à l'étape de traitement, il est essentiel d'examiner les caractéristiques des variables disponibles, de repérer d'éventuelles valeurs manquantes, des données aberrantes ou encore des incohérences. L'EDA permet aussi de visualiser les distributions de variables quantitatives comme le prix, la surface habitable ou le nombre de chambres, afin d'identifier des tendances générales, des déséquilibres ou des relations cachées. Ces visualisations facilitent non seulement la détection de problèmes de qualité des données, mais orientent également les choix de prétraitement et de sélection des variables pertinentes. En somme, l'EDA offre une vision globale du dataset, guide les étapes suivantes du projet, et contribue fortement à la qualité et à la performance des modèles d'apprentissage automatique qui en découleront.

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Price distribution
sns.histplot(df['price'], )
plt.title('Distribution of House Prices')
plt.show()

# Other continuous features
num_cols = ['bedrooms', 'bathrooms', 'sqft_living', 'floors']
for col in num_cols:
    sns.histplot(df[col], )
    plt.title(f'Distribution of {col}')
    plt.show()
```

Figure 1: la distribution des principales variables numériques

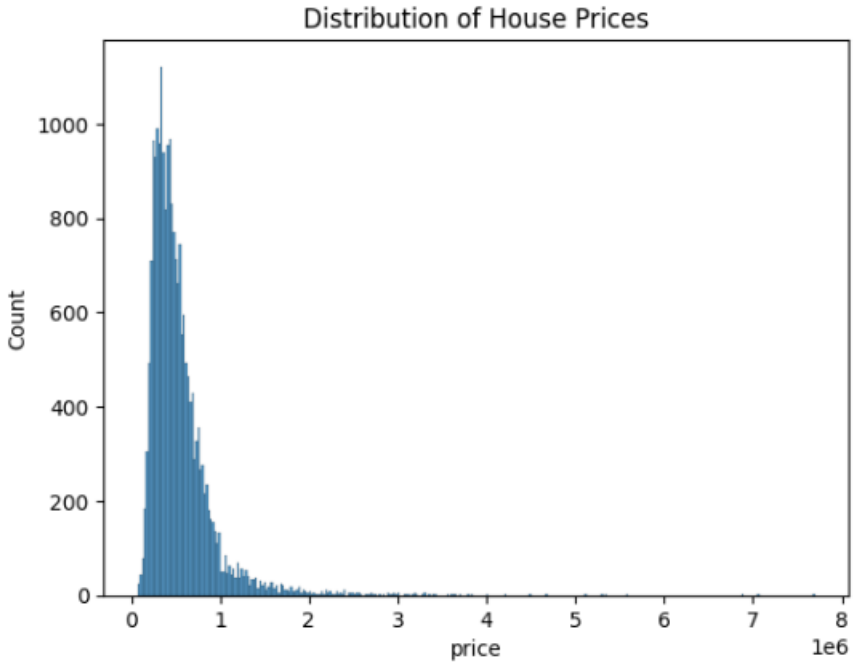

Figure 2: Distribution du prix des maisons

Dans l'histogramme représentant la distribution des prix des maisons, on observe une asymétrie positive, avec la majorité des maisons ayant un prix compris entre 100 000 et 600 000 dollars. Cela signifie que la plupart des logements sont proposés à un prix moyen ou abordable. On remarque également que certaines maisons très luxueuses ou atypiques atteignent des prix beaucoup plus élevés, dépassant parfois le million de dollars, ce qui crée une queue à droite du graphique.

Cela met en évidence la nécessité de traiter ces valeurs extrêmes (outliers) dans les étapes suivantes, afin d'éviter qu'elles n'influencent de manière excessive les modèles d'apprentissage automatique.

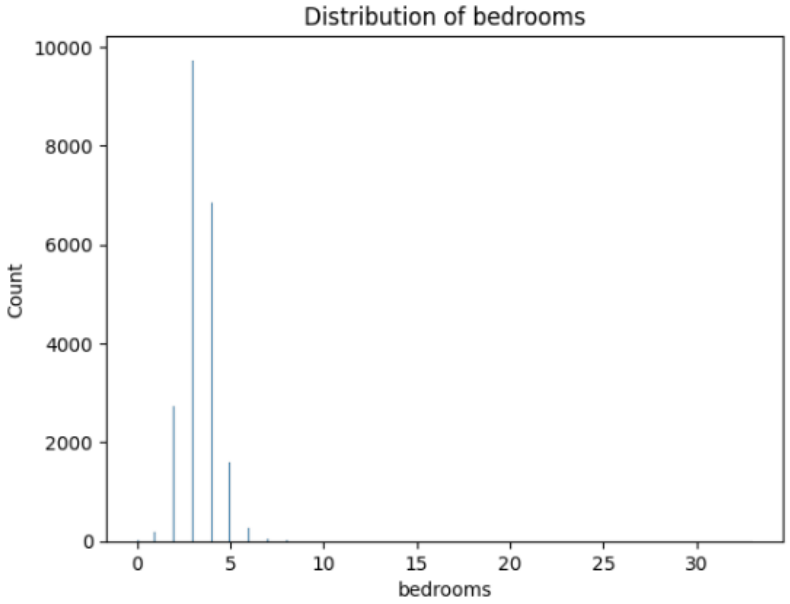

Figure 3: Répartition du nombre de chambres

Dans l'histogramme représentant la répartition du nombre de chambres, on observe que la majorité des maisons possèdent entre 2 et 4 chambres, avec un pic à 3 chambres. Cela indique que ces types de logements sont les plus courants sur le marché, car ils répondent généralement aux besoins des familles moyennes.

On remarque aussi quelques maisons avec un nombre très élevé de chambres (plus de 6), mais elles sont rares. Ces valeurs peuvent être considérées comme atypiques ou luxueuses, et peuvent influencer les statistiques si elles ne sont pas bien traitées.

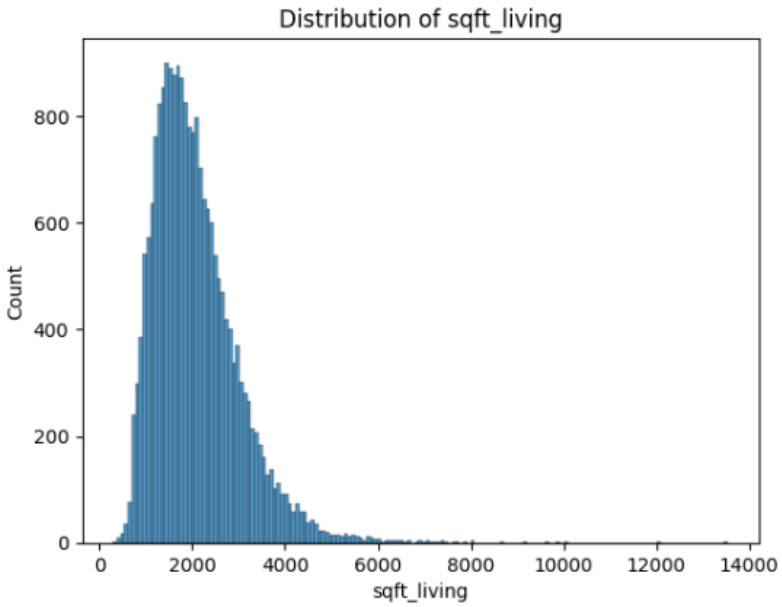

Figure 4: Répartition de la surface habitable (en pieds carrés)

Dans l'histogramme représentant la répartition de la surface habitable, on observe une concentration des maisons entre 1 000 et 2 500 pieds carrés. Cela reflète des logements de taille moyenne, adaptés à la plupart des familles.

On observe une queue étendue vers la droite, ce qui signifie qu'il existe quelques maisons très grandes (plus de 4 000 pieds carrés), mais elles sont peu fréquentes. Ces valeurs extrêmes représentent souvent des propriétés de luxe et peuvent influencer la moyenne, d'où l'importance de les traiter lors de l'analyse.

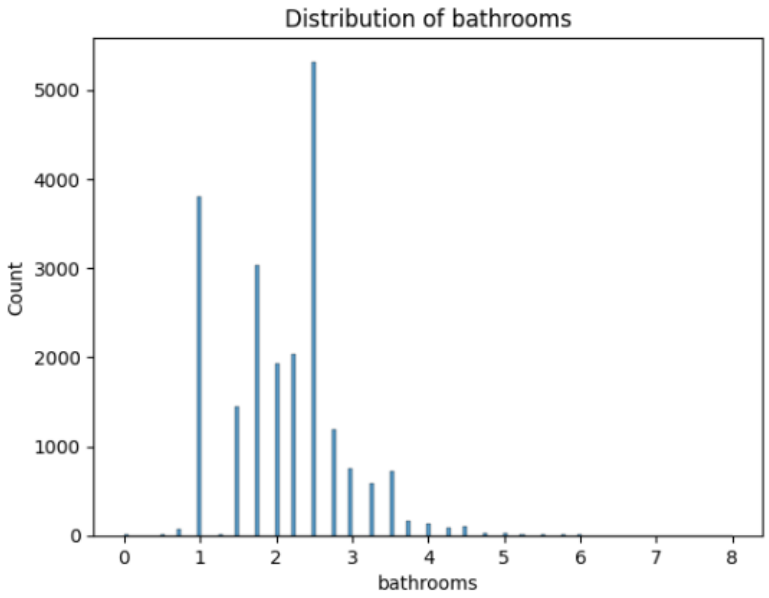

Figure 5:Répartition du nombre de salles de bain

Dans l'histogramme représentant la répartition du nombre de salles de bain, on observe que la majorité des maisons possèdent entre 1 et 2 salles de bain, avec un pic à 2 salles de bain. Cela reflète des configurations standards répondant aux besoins des familles moyennes.

On remarque également la présence de maisons avec 3 salles de bain ou plus, mais ces cas sont moins fréquents, ce qui peut indiquer des logements plus spacieux ou haut de gamme. Ces valeurs, bien que rares, peuvent influencer la moyenne et doivent être prises en compte lors du traitement des données.

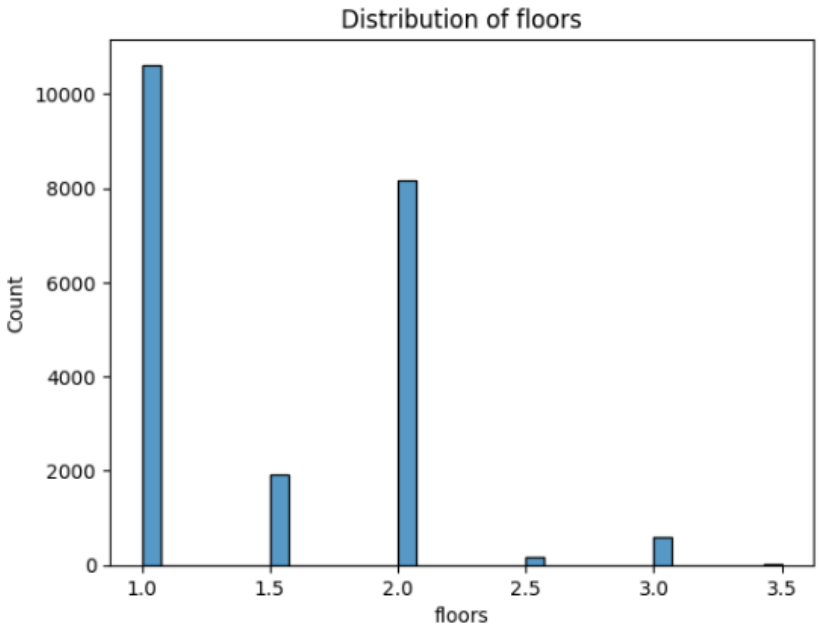

Figure 6: Répartition du nombre d'étages

Dans l'histogramme représentant la répartition du nombre d'étages, on observe que la majorité des maisons possèdent un seul étage, ce qui indique que les logements de plain-pied sont les plus courants sur le marché. On remarque également la présence de maisons avec deux étages, mais elles sont moins fréquentes. Les logements avec trois étages ou plus sont très rares, ce qui les rend atypiques.

Cela montre que les maisons à un étage répondent généralement aux besoins standards des acheteurs, tandis que les maisons à plusieurs étages peuvent être considérées comme plus spacieuses ou luxueuses. Ces valeurs rares doivent être prises en compte lors de l'analyse.

```
corr = df.corr(numeric_only=True)

plt.figure(figsize=(10, 6))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```

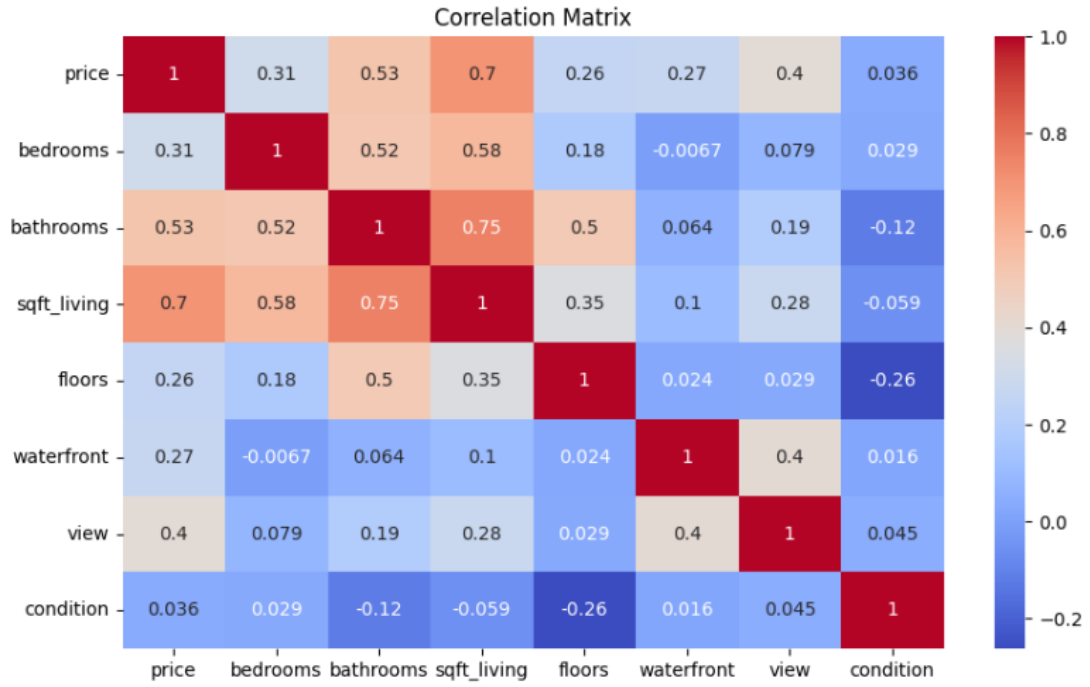

Figure 7: Matrice de corrélation entre les variables numériques

Dans la matrice de corrélation, on retrouve les coefficients de corrélation qui montrent la nature de la relation entre les différentes variables numériques du dataset.

sqft_living : le coefficient de corrélation (0,70) montre une forte corrélation positive avec le prix, ce qui indique que les maisons plus grandes sont généralement plus chères.

bathrooms : le coefficient de corrélation (0,53) présente également une corrélation positive avec le prix, suggérant que les logements avec plus de salles de bain tendent à avoir une valeur plus élevée.

bedrooms : le coefficient de corrélation (0,31) est plus faible mais reste positif, ce qui signifie que le nombre de chambres influence le prix, mais moins fortement que la surface ou les salles de bain.

floors : le coefficient de corrélation (0,26) indique une légère corrélation positive, ce qui signifie que les maisons à plusieurs étages peuvent être un peu plus coûteuses.

waterfront : le coefficient de corrélation (0,27) montre une corrélation positive, ce qui indique que les maisons avec vue sur l'eau sont généralement plus chères.

view : le coefficient de corrélation (0,40) montre une bonne corrélation avec le prix, car une belle vue peut faire augmenter la valeur d'un logement.

Condition : le coefficient de corrélation (0,036) présente une corrélation très faible, suggérant que l'état général du logement a un impact limité sur le prix par rapport aux autres variables.

```
# Price vs Bedrooms

sns.boxplot(x='bedrooms', y='price', data=df)
plt.title('Price vs Bedrooms')
plt.show()

# Price vs Bathrooms
plt.figure(figsize=(12,8))
sns.boxplot(x='bathrooms', y='price', data=df)
plt.title('Price vs Bathrooms')
plt.show()

# Price vs Waterfront
sns.boxplot(x='waterfront', y='price', data=df)
plt.title('Price vs Waterfront View')
plt.show()
```

Figure 8 : Analyser l'influence de différentes caractéristiques

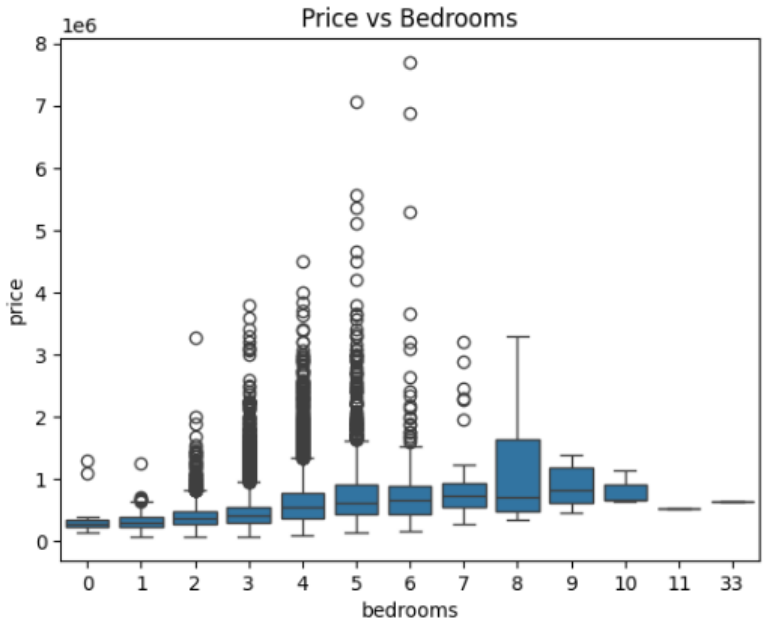

Figure 9: Prix en fonction du nombre de chambres

Dans le Boxplot représentant l'évolution du prix en fonction du nombre de chambres, on observe une tendance générale à la hausse : plus le nombre de chambres augmente, plus le prix médian des maisons tend à augmenter.

Cependant, cette progression n’est pas strictement linéaire. À partir de 5 chambres et plus, on remarque une plus grande dispersion des prix, ce qui indique que d’autres facteurs (comme la surface ou l’emplacement) influencent fortement la valeur des maisons à grand nombre de chambres.

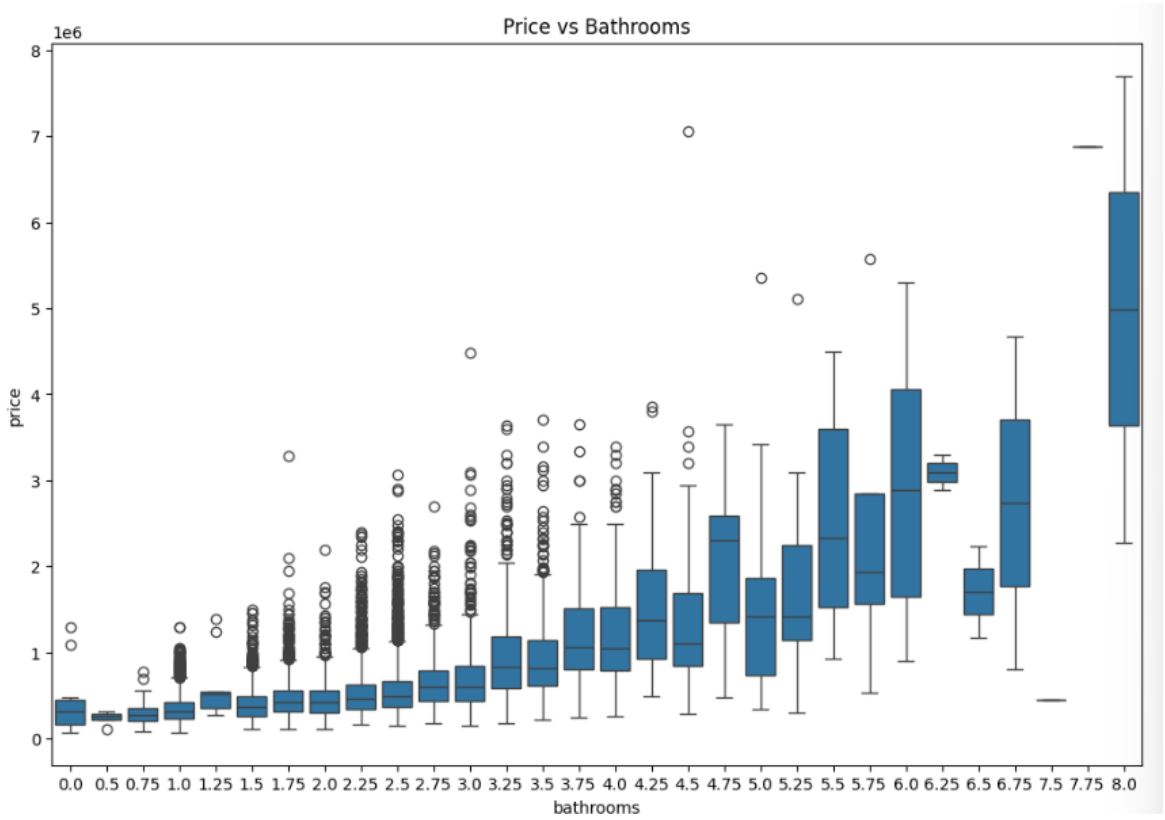

Figure 10 : Prix en fonction du nombre de salles de bains

Dans le Boxplot représentant la variation du prix en fonction du nombre de salles de bains, on observe que le prix des maisons augmente globalement avec le nombre de salles de bains.

Les maisons avec 1 à 2 salles de bains sont les plus courantes et présentent des prix relativement modérés. À partir de 3 salles de bains, les prix deviennent plus élevés, mais aussi plus dispersés, ce qui indique une plus grande variabilité selon d'autres critères (surface, localisation, etc.).

Cette tendance montre que le nombre de salles de bains est un facteur pertinent dans la valorisation des biens, mais qu’il doit être analysé en combinaison avec d’autres caractéristiques pour une estimation plus précise.

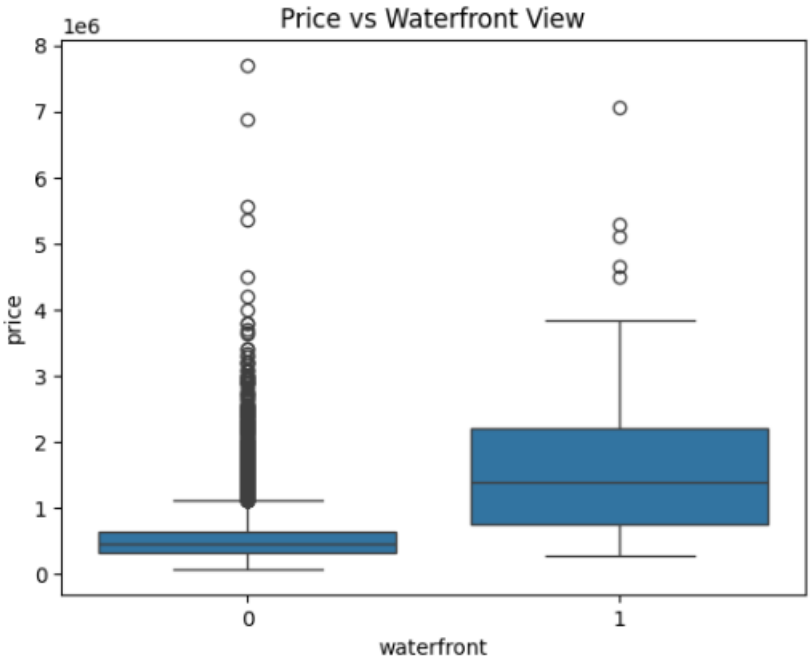

Figure 11: Prix en fonction de la présence d'une vue sur l'eau

Dans le Boxplot représentant la variation du prix selon la présence d'une vue sur l'eau, on observe une différence de prix nette entre les maisons avec vue et celles sans.

Les maisons bénéficiant d'une vue sur l'eau affichent en moyenne des prix nettement plus élevés, ce qui souligne l'impact significatif de cette caractéristique sur la valeur du bien immobilier.

Cette observation met en évidence l'importance des éléments liés à l'environnement et au cadre de vie dans l'évaluation d'un bien, au-delà des simples caractéristiques structurelles comme la surface ou le nombre de chambres.

```
sns.boxplot(y='price', data=df)
plt.title('Boxplot of House Prices')
plt.show()
```

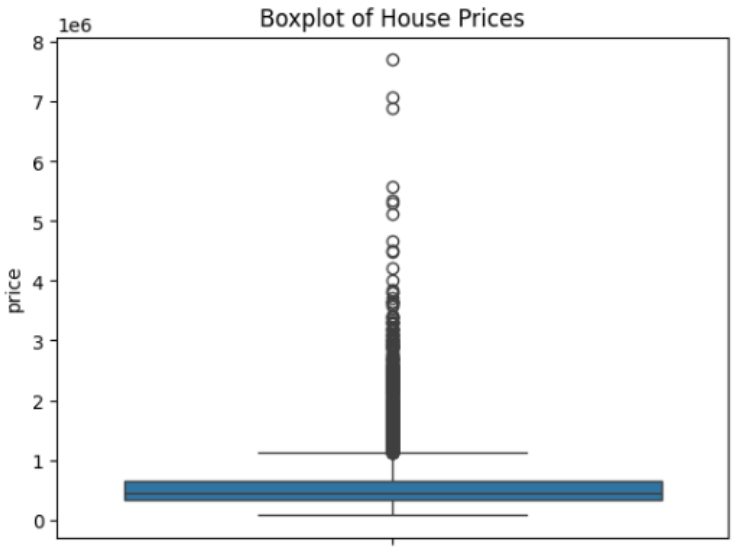

Figure 12: Répartition des prix des maisons avec identification des valeurs extrêmes

Dans le Boxplot représentant la répartition des prix des maisons, on observe que la boîte centrale (box) contient la majorité des données, avec une concentration des prix entre le premier et le troisième quartile, soit approximativement entre 200 000 et 600 000 dollars.

La ligne médiane au centre de la boîte indique la valeur médiane du prix, qui représente le prix typique d'une maison dans le dataset.

On distingue également de nombreux points au-dessus de la boîte, qui représentent des valeurs extrêmes (outliers). Ces prix très élevés (souvent supérieurs à 1 million de dollars) correspondent à des propriétés exceptionnelles ou de luxe.

```
# For smaller datasets
sns.pairplot(df[['price', 'bedrooms', 'bathrooms', 'sqft_living']])
plt.show()
```

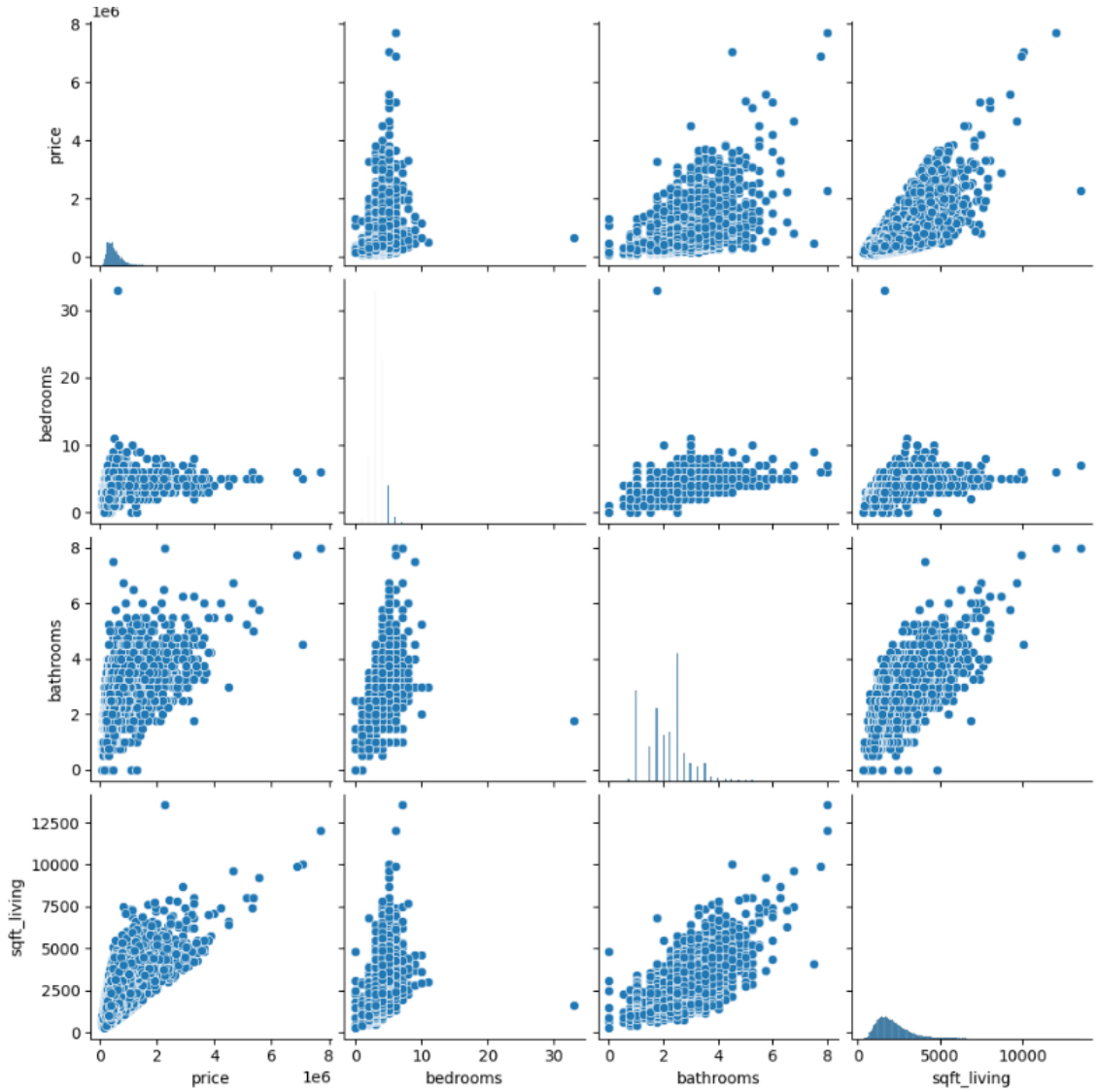

Figure 13: Analyse des relations entre les variables principales du dataset

Le graphe de nuages de points croisés permet de visualiser les relations linéaires entre les principales variables numériques du dataset, notamment en lien avec la variable prix.

Cette visualisation permet de confirmer et renforcer les résultats obtenus dans la matrice de corrélation :

sqft_living (surface habitable) : une relation linéaire claire et positive avec le prix est visible. On remarque que, globalement, plus la surface est grande, plus le prix augmente, ce qui renforce le coefficient de corrélation élevé (0,70).

bathrooms (salles de bains) : le graphe montre également une relation linéaire positive avec le prix. Les maisons avec plus de salles de bains tendent à être plus chères, bien que la dispersion augmente à partir de 3 salles de bains.

bedrooms (chambres) : la relation avec le prix est moins marquée, ce qui est cohérent avec un coefficient de corrélation plus faible (0,31). Certaines maisons avec beaucoup de chambres ont des prix variés, ce qui suggère que ce critère seul n'est pas un bon prédicteur.

3. Nettoyage et préparation initiale des données

Avant de procéder à toute modélisation ou analyse approfondie, il est essentiel de s'assurer que les données sont propres, cohérentes et prêtes à être exploitées. Cette étape vise à identifier et corriger les incohérences, supprimer les valeurs manquantes ou aberrantes, et harmoniser le format des variables. Un nettoyage rigoureux permet de garantir la fiabilité des résultats, d'améliorer la performance des modèles prédictifs, et de réduire les biais liés à la qualité des données. Dans cette section, nous détaillons les principales opérations de nettoyage appliquées au dataset, notamment la gestion des valeurs extrêmes et la vérification de la complétude des données.

```
import pandas as pd
df=pd.read_csv('Housing.csv')
df=df[[
'price',
'bedrooms',
'bathrooms',
'sqft_living',
'floors',
'waterfront',
'view',
'condition',
]]
```

Figure 14: Sélection des variables clés pour une analyse ciblée du marché immobilier.

Le dataset initial est chargé, puis seules les colonnes jugées utiles pour l'analyse sont conservées. Cela permet de se concentrer uniquement sur les variables pertinentes pour la modélisation.

```
df = df.dropna()
```

Figure 15 : Suppression des valeurs manquantes

Cette commande permet d'éliminer les lignes contenant des valeurs manquantes. Pour avoir une bonne qualité des données.

```
df=df.drop_duplicates()
```

Figure 16: Suppression des doublons

Les enregistrements dupliqués sont supprimés pour éviter les biais statistiques liés à la redondance des données.

```python
# Calculate Q1 and Q3
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1

# Define bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
df = df[(df['price'] >= lower_bound) & (df['price'] <= upper_bound)]

# Check shape after removing outliers
print("New shape after removing price outliers:", df.shape)
```

```
New shape after removing price outliers: (20036, 8)
```

Figure 17: Détection et suppression des valeurs aberrantes dans les prix des maisons

Ce code élimine les valeurs aberrantes du prix en appliquant la méthode de l'IQR, garantissant ainsi une distribution plus représentative et améliorant la qualité des analyses statistiques.

```python
df.to_csv('cleaned_housing.csv')
```

Figure 18: Export des données nettoyées pour les prochaines étapes du projet.

Le dataset final, propre et filtré, est sauvegardé pour un usage ultérieur. Cela garantit la traçabilité et la reproductibilité des analyses.

Chapitre 3 : Méthodologie Approchée

1. Prétraitement des données

a. Normalisation des variables :

La normalisation des données est une étape essentielle en machine learning, en particulier lorsque les variables ont des échelles très différentes. Par exemple, dans notre dataset immobilier, les valeurs de la superficie habitable peuvent atteindre plusieurs milliers, tandis que le nombre d'étages ou la condition d'une maison varient sur une échelle beaucoup plus petite. Sans normalisation, les algorithmes basés sur des calculs de distance (comme la régression, les SVM ou les réseaux de neurones) pourraient accorder trop d'importance aux variables ayant de grandes valeurs. La normalisation permet de centrer les données autour de 0 avec un écart-type de 1, assurant ainsi que chaque variable contribue de manière équitable à la modélisation. Elle améliore également la convergence des algorithmes d'optimisation et la stabilité numérique du modèle.

```python
import pandas as pd
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import cosine_similarity

# --- Chargement et preprocessing ---
df = pd.read_csv('/kaggle/working/cleaned_housing.csv')
features = df.drop(columns=['id', 'Unnamed: 0'])
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

Figure 19: Application de la normalisation

Pour appliquer la normalisation, nous avons commencé par charger le jeu de données nettoyé (cleaned_housing.csv). Ensuite, les colonnes non nécessaires à la modélisation telles que id et Unnamed: 0 ont été supprimées. Le reste des variables numériques ont été extraites dans un dataframe features. À partir de là, nous avons utilisé l'outil StandardScaler de la bibliothèque scikit-learn pour transformer ces variables. La méthode fit_transform() a été appliquée afin de calculer les moyennes et écarts-types des variables, puis de les standardiser. Le résultat est un tableau de données prêtes à être utilisées pour l'entraînement d'un modèle de prédiction, où chaque variable a la même importance initiale.

b. Conversion et structuration pour TensorFlow

```python
# --- TensorFlow dataset ---
features_tensor = tf.convert_to_tensor(features_scaled, dtype=tf.float32)
```

Figure 20: es données normalisées en un tenseur TensorFlow.

Cette ligne convertit les données normalisées en un tenseur TensorFlow. Les tenseurs sont le format natif que TensorFlow utilise pour traiter les données.L'option dtype=tf.float32 garantit que les valeurs soient bien en format flottant, ce qui est standard pour l'entraînement des modèles. Cela rend les données prêtes pour l'étape suivante du deep learning.

c. Construction des triplets d'entraînement

Dans les systèmes de recommandation basés sur l'apprentissage par similarité, la génération de triplets est une étape essentielle. Un triplet est composé de trois éléments : un ancrage (anchor), un positif (similaire à l'ancrage) et un négatif (différent). Ces triplets servent à entraîner un modèle à apprendre une représentation des données dans laquelle les éléments similaires sont plus proches que les éléments différents.

```
def generate_triplets(features, num_triplets=5000):
    triplets = []
    n = len(features)
    for _ in range(num_triplets):
        anchor = np.random.randint(0, n)
        positive = (anchor + 1) % n
        negative = np.random.randint(0, n)
        while negative == anchor or negative == positive:
            negative = np.random.randint(0, n)
        triplets.append((anchor, positive, negative))
    return np.array(triplets)

triplet_indices = generate_triplets(features_scaled, num_triplets=5000)
anchor = tf.gather(features_tensor, triplet_indices[:, 0])
positive = tf.gather(features_tensor, triplet_indices[:, 1])
negative = tf.gather(features_tensor, triplet_indices[:, 2])

triplet_dataset = tf.data.Dataset.from_tensor_slices((anchor, positive, negative)).batch(3000)
```

Figure 21: le code de génération des triplets

Dans ce code, 5 000 triplets sont générés à partir des données normalisées, les indices des points sont ensuite utilisés pour extraire les vecteurs correspondants avec tf.gather(), puis transformés en un dataset TensorFlow grâce à from_tensor_slices() pour être utilisés en batchs lors de l'apprentissage. Ce processus permet au modèle d'apprendre à rapprocher les points similaires et à éloigner les points différents dans l'espace des caractéristiques.

2. Conception du modèle de Deep Learning

a. Architecture du réseau de neurones

L'objectif de cette étape est de transformer les caractéristiques tabulaires des biens immobiliers en vecteurs numériques de faible dimension (embeddings).Le modèle d'embedding est un réseau de neurones conçu pour transformer les caractéristiques des maisons en vecteurs denses de dimension réduite (ici 32). Il permet de capturer les similarités latentes entre biens immobiliers en apprenant des représentations compactes et significatives. Composé de couches entièrement connectées avec activation ReLU, ce modèle sert de base pour le calcul des distances entre maisons, facilitant ainsi la recommandation de biens similaires.

```
embedding_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(features.shape[1],)),  # ici 9
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32)
])
```

Figure 22: Modèle d'embedding

Architecture du modèle :

- Input layer : dimension d'entrée égale au nombre de features (ex : 9).
- Dense layers : deux couches cachées avec activation ReLU.
- Output layer : vecteur final de 32 dimensions.

b. Fonction de perte utilisée : Triplet Loss

Une fonction de perte (ou loss function) est une fonction mathématique utilisée en apprentissage automatique pour mesurer l'erreur entre la prédiction du modèle et la valeur réelle attendue. Elle permet de dire au modèle à quel point il s'est trompé. Plus la perte est élevée, plus le modèle s'est éloigné de la bonne réponse. Le but de l'entraînement est de minimiser cette perte.

```
def triplet_loss(a, p, n, margin=1.0):
    pos_dist = tf.reduce_sum(tf.square(a - p), axis=1)
    neg_dist = tf.reduce_sum(tf.square(a - n), axis=1)
    loss = tf.maximum(pos_dist - neg_dist + margin, 0.0)
    return tf.reduce_mean(loss)
```

Figure 23 : la Fonction de Perte : Triplet Loss

Explication :

- pos_dist calcule la distance entre l'ancre et le positif.
- neg_dist calcule la distance entre l'ancre et le négatif.
- loss = max(pos_dist - neg_dist + margin, 0) : on veut que pos_dist soit **au moins margin plus petit** que neg_dist. Si ce n'est pas le cas, la perte sera positive et le modèle sera pénalisé.

- tf.reduce_mean prend la moyenne de toutes les pertes pour l'optimisation.

c. Stratégie d'entraînement et paramètres choisis

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
print("\n--- Entraînement avec Triplet Loss ---")
epochs = 200
for epoch in range(epochs):
    epoch_loss = 0
    for step, (a, p, n) in enumerate(triplet_dataset):
        with tf.GradientTape() as tape:
            a_embed = embedding_model(a)
            p_embed = embedding_model(p)
            n_embed = embedding_model(n)
            loss = triplet_loss(a_embed, p_embed, n_embed)
        grads = tape.gradient(loss, embedding_model.trainable_variables)
        optimizer.apply_gradients(zip(grads, embedding_model.trainable_variables))
        epoch_loss += loss.numpy()
    print(f"Epoch {epoch + 1}/{epochs} - Loss: {epoch_loss / (step + 1):.4f}")

# --- Embeddings finaux ---
final_embeddings = embedding_model(features_tensor).numpy()
```

Figure 24: la phase d'entraînement du modèle d'embedding

Ce code correspond à la phase d'entraînement du modèle d'embedding à l'aide de la Triplet Loss. On commence par définir un optimiseur, ici Adam, avec un taux d'apprentissage fixé à 0.01. L'entraînement se déroule sur 200 époques, et pour chaque époque, on boucle sur les triplets (ancre, positive, négative) du dataset. À chaque itération, le modèle génère les embeddings de ces trois exemples et calcule la perte avec la fonction triplet_loss. À l'aide de tf.GradientTape, on enregistre les opérations pour calculer automatiquement les gradients, puis on met à jour les poids du modèle en appliquant ces gradients. La perte moyenne par époque est affichée pour suivre la progression. À la fin de l'entraînement, on génère les embeddings finaux pour toutes les données d'entrée en appliquant le modèle entraîné sur l'ensemble du jeu de données.

Chapitre 4 : Évaluation des Performances du Système

1. Fonction de recommandation par ID :

Cette fonction a pour objectif de recommander les maisons les plus similaires à une maison de référence, identifiée par son house_id. Elle repose sur les embeddings générés par le modèle d'apprentissage, qui représentent chaque bien immobilier dans un espace vectoriel de caractéristiques latentes. En calculant la similarité cosinus entre l'embedding de la maison cible et ceux de toutes les autres maisons du dataset, la fonction identifie les propriétés les plus proches selon cette représentation vectorielle. Elle exclut volontairement la maison elle-même des résultats afin d'éviter tout biais. Cette fonction est principalement utilisée pour fournir à l'utilisateur final une liste personnalisée de biens similaires, jouant ainsi le rôle d'un moteur de suggestion dans un système de recommandation immobilière.

```python
def recommend_similar(house_id, top_k=5):
    idx = df[df['id'] == house_id].index[0]
    query_embedding = final_embeddings[idx:idx+1]
    sims = cosine_similarity(query_embedding, final_embeddings)[0]
    sims[idx] = -np.inf  # Exclure la maison elle-même
    top_k_indices = np.argsort(sims)[-top_k:][::-1]
    return df.iloc[top_k_indices]

# --- Exemple ---
house_id = df['id'].iloc[0]
print(f"\nMaisons similaires à la maison avec ID {house_id} :")
print(recommend_similar(house_id, top_k=5))
```

Figure 25: Fonction de recommandation par ID

2. Evaluation du système de recommandation :

L'évaluation du système de recommandation est cruciale car elle permet de mesurer l'efficacité réelle du modèle à fournir des suggestions pertinentes et utiles aux utilisateurs. Sans cette étape, il serait impossible de savoir si les recommandations sont précises ou simplement aléatoires. L'évaluation aide à quantifier la performance à l'aide de métriques objectives, comme la précision (Top-k Accuracy) et le rang moyen inverse (MRR), ce qui guide l'amélioration continue du système. Elle garantit aussi que le modèle répond bien aux attentes des utilisateurs, augmente leur satisfaction, et favorise ainsi l'adoption et le succès de la solution développée. En somme, c'est un élément clé pour valider la qualité et la fiabilité du moteur de recommandation avant sa mise en production.

[37]:
```python
def evaluate_recommendations(df, final_embeddings, top_k=5, sample_ratio=0.2):
    reciprocal_ranks = []
    topk_hits = 0
    total = len(df)

    # Prendre un échantillon de 20% des données
    sample_size = int(sample_ratio * total)
    sampled_indices = np.random.choice(total, size=sample_size, replace=False)

    print(f"\n--- Évaluation sur un échantillon de {sample_size} maisons ---")
    for count, i in enumerate(sampled_indices):
        try:
            query_embedding = final_embeddings[i]
            house_id = df['id'].iloc[i]

            # Similarités avec toutes les maisons
            sims = cosine_similarity(query_embedding.reshape(1, -1), final_embeddings)[0]
            sims[i] = -np.inf  # on ignore la maison elle-même

            # Vrai plus proche voisin
            true_closest_idx = np.argmax(sims)
            true_closest_id = df['id'].iloc[true_closest_idx]

            # Recommandations du modèle
            recs = recommend_similar(house_id, top_k=top_k)
            rec_ids = recs['id'].values

            # Top-K hit
            if true_closest_id in rec_ids:
                topk_hits += 1

            # MRR
            if true_closest_id in rec_ids:
                rank = np.where(rec_ids == true_closest_id)[0][0] + 1
                reciprocal_ranks.append(1.0 / rank)
            else:
                reciprocal_ranks.append(0.0)

            if count % 100 == 0:
                print(f"{count}/{sample_size} maisons traitées...")

        except Exception as e:
            print(f"Erreur à l'index {i} : {e}")
            continue

    topk_accuracy = topk_hits / sample_size
    mrr = np.mean(reciprocal_ranks)

    print("\n--- Résultats de l'Évaluation ---")
    print(f"Top-{top_k} Accuracy : {topk_accuracy:.4f}")
    print(f"MRR : {mrr:.4f}")
```

[38]:
```python
evaluate_recommendations(df, final_embeddings, top_k=5)
```

Cette fonction a pour rôle d'évaluer la performance globale du système de recommandation. En prenant un échantillon aléatoire de maisons dans la base de données, elle mesure la qualité des recommandations produites par le modèle en comparant les biens recommandés avec la vraie maison la plus proche selon les embeddings. Pour cela, elle calcule des métriques clés comme la Top-k Accuracy, qui indique la proportion de cas où la maison la plus similaire figure bien parmi les recommandations, et le Mean Reciprocal Rank (MRR), qui renseigne sur la position moyenne de cette maison dans la liste recommandée. Grâce à cette évaluation, on obtient des scores quantitatifs permettant d'apprécier la précision et la pertinence du moteur de recommandation, ce qui est essentiel pour valider et améliorer le modèle.

3. Résultats de l'évaluation

L'évaluation a été réalisée sur un échantillon de 4 094 maisons, avec un suivi progressif de l'avancement toutes les 100 maisons traitées, assurant ainsi une bonne traçabilité du processus. Les résultats montrent une **Top-5 Accuracy de 99,39 %**, ce qui signifie que dans plus de 99 % des cas, la maison la plus similaire réelle figure bien parmi les 5 premières recommandations du système. Cela témoigne d'une très haute précision dans la capacité du modèle à proposer des recommandations pertinentes et proches des attentes utilisateurs.

De plus, le **Mean Reciprocal Rank (MRR) de 0,9905** indique que la maison réellement la plus proche est très souvent classée en première position parmi les recommandations, ce qui reflète une excellente qualité de classement des suggestions. En résumé, ces métriques révèlent que le système de recommandation est performant, fiable et apte à offrir une expérience utilisateur satisfaisante en proposant des biens immobiliers similaires avec une grande précision. Ces résultats positifs valident l'efficacité du modèle et son potentiel d'intégration dans une application réelle.

```
--- Résultats de l'Évaluation ---
Top-5 Accuracy : 0.9939
MRR : 0.9905
```

Figure 26: résultats de l'évaluation

4. La phase de test et observations

Cette partie sert à tester et à montrer comment utiliser le modèle pour obtenir des recommandations en pratique, à partir d'une entrée utilisateur. C'est une étape clé pour valider le comportement du système hors des données d'entraînement ou d'évaluation formelle.

```python
def recommend_by_features(input_features_dict, top_k=10):

    # Convert input features into same order as training data
    input_df = pd.DataFrame([input_features_dict])
    input_scaled = scaler.transform(input_df)
    input_tensor = tf.convert_to_tensor(input_scaled, dtype=tf.float32)

    # Get embedding
    input_embedding = embedding_model(input_tensor).numpy()

    # Compute cosine similarity
    sims = cosine_similarity(input_embedding, final_embeddings)[0]
    top_k_indices = np.argsort(sims)[-top_k:][::-1]

    return df.iloc[top_k_indices]
```

Figure 27: Implémentation du système de recommandation

Le calcul de la similarité cosinus entre cet embedding d'entrée et ceux des biens existants permet d'identifier rapidement les maisons les plus proches en termes de caractéristiques, répondant ainsi efficacement à une requête personnalisée. Les résultats obtenus, présentés dans l'exemple, démontrent la capacité du modèle à extraire des recommandations pertinentes, cohérentes et alignées avec les critères saisis.

```python
input_features = {
     'price':16500,
    'bedrooms': 1.5,
    'bathrooms': 1,
    'sqft_living': 1000,

    'floors': 1,
    'waterfront': 0,
    'view': 0,
    'condition': 3,

}



top_similar = recommend_by_features(input_features, top_k=10)
print(top_similar)
```

Figure 28: Code de teste du modèle

Cette étape de test confirme non seulement la justesse du modèle d'embedding et la pertinence des similarités calculées, mais aussi la qualité globale du système de recommandation. En outre, elle illustre que le projet a été mené à bien avec succès, aboutissant à un outil performant capable de répondre aux besoins des utilisateurs de manière fiable et précise. Ce succès est un indicateur fort de la robustesse du pipeline de traitement des données, du choix des caractéristiques, ainsi que de l'efficacité des algorithmes d'apprentissage et d'optimisation employés.

	Unnamed: 0	id	price	bedrooms	bathrooms	sqft_living	\
15186	15986	87000213	129000.0	2	1.0	1150	
2459	2589	5061300030	134000.0	2	1.5	980	
5528	5816	7568700480	153000.0	2	1.0	1140	
13861	14581	6929602721	95000.0	2	1.0	960	
3229	3397	2172000750	160000.0	2	1.0	1180	
12923	13601	8698600395	150000.0	2	1.0	1250	
15864	16714	1322049150	85000.0	2	1.0	910	
9605	10105	5466310060	139500.0	2	1.5	1230	
17524	18468	7999600180	83000.0	2	1.0	900	
2954	3108	1721801591	89950.0	1	1.0	570	

	floors	waterfront	view	condition
15186	1.0	0	0	3
2459	2.0	0	0	3
5528	1.0	0	0	3
13861	1.0	0	0	3
3229	1.0	0	0	3
12923	1.0	0	0	3
15864	1.0	0	0	3
9605	2.0	0	0	3
17524	1.0	0	0	3
2954	1.0	0	0	3

Figure 29: Résultat de projet réalisé

Chapitre 5 : Conclusion

En conclusion, ce projet de système de recommandation immobilière a permis de mettre en œuvre et d'intégrer plusieurs compétences clés allant de la préparation et la normalisation des données, à la construction et à l'entraînement d'un modèle d'embeddings performant, jusqu'à l'évaluation rigoureuse des résultats obtenus via des métriques pertinentes telles que la Top-k Accuracy et le MRR. Grâce à l'utilisation de techniques avancées de machine learning et de calculs de similarité, le système développé est capable de proposer des recommandations précises et adaptées aux critères spécifiques des utilisateurs, démontrant ainsi son efficacité et sa robustesse.

Cette expérience a constitué une véritable opportunité d'enrichir mes connaissances tant théoriques que pratiques dans les domaines de l'apprentissage automatique, du traitement de données complexes, et du développement de systèmes intelligents. Elle m'a également permis de renforcer ma capacité à gérer un projet complet, depuis la collecte des données jusqu'à la validation finale, tout en adoptant une approche méthodique et professionnelle. Ce projet représente ainsi une étape significative dans mon parcours, consolidant mes compétences en informatique et en data science, et m'ouvrant la voie vers des applications encore plus ambitieuses.