Projet de Machine Learning 2024 - 2025

Stephen Cohen - Alexis Kien PHAM TRUNG - Antoine Rech--Tronville

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import shap
```

```
c:\Users\Stephen\AppData\Local\Programs\Python\Python311\Lib\site-
packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please
update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Analyse des données

Présentation

Notre jeu de données combine plusieurs sources :

1. Données immobilières (mutations_d75.csv) :
   - Transactions immobilières sur Paris et l'île de France
   - Variables principales : date_mutation, type_bien, surface, prix, code_insee
2. Données macroéconomiques :
   - Indice de confiance des ménages (confiance_menage.csv)
   - Taux directeur de la BCE (taux.csv)
   - Cours du pétrole Brent (petrole.csv)

Cette combinaison nous permet d'analyser l'impact des facteurs économiques sur les prix immobiliers.

Notre objectif est d'essayer de prédire au mieux le prix d'un appartement selon le contexte économique et les caractéristiques de l'appartement.

```
df = pd.read_pickle('processed_data/clean_data.pkl')

print("Aperçu des données :")
display(df.head())
print("\nStatistiques descriptives :")
display(df.describe())
```

```
Aperçu des données :

  year_month l_codinsee       type_bien  surface       prix_m2
departement  \
4    2015-04      75120  UN APPARTEMENT     61.0   4836.065574
75
5    2015-12      75111  UN APPARTEMENT     50.0   5840.000000
75
6    2024-01      75118  UN APPARTEMENT     22.0  11272.727273
75
7    2024-01      75118  UN APPARTEMENT     22.0  11272.727273
75
8    2024-01      75118  UN APPARTEMENT     22.0  11272.727273
75

   valeurfonc       date  confiance_menages  taux_directeur
cours_petrole
4    295000.0        NaT                NaN             NaN
NaN
5    292000.0        NaT                NaN             NaN
NaN
6    248000.0 2024-01-01          94.446667           3.894
82.49
7    248000.0 2024-01-02          94.573333           3.906
82.49
8    248000.0 2024-01-03          94.700000           3.904
82.49


Statistiques descriptives :

            surface       prix_m2    valeurfonc  \
count  1.966462e+07  1.966462e+07  1.966462e+07
mean   6.187738e+02  4.547486e+03  4.318239e+05
min    1.000000e+00  8.400000e-01  1.000000e+00
25%    4.500000e+01  1.250000e+03  1.800000e+05
50%    7.200000e+01  3.642857e+03  2.800000e+05
75%    2.580000e+02  6.891304e+03  4.400000e+05
max    6.098051e+06  1.769964e+04  3.411290e+08
std    1.401090e+04  3.817738e+03  1.435034e+06

                                date  confiance_menages
taux_directeur  \
count                       18854925       1.885492e+07
```

```
1.864205e+07
mean   2021-11-23 12:07:56.155021312       9.361563e+01    5.103527e-
01
min              2014-01-01 00:00:00       8.300000e+01   -5.930000e-
01
25%              2020-11-03 00:00:00       8.692116e+01   -5.670000e-
01
50%              2021-11-15 00:00:00       9.394000e+01   -5.510000e-
01
75%              2022-12-12 00:00:00       9.830417e+01
1.404000e+00
max              2024-06-28 00:00:00       1.107867e+02
3.913000e+00
std                              NaN       6.811996e+00
1.699023e+00

       cours_petrole
count   1.885492e+07
mean    7.806843e+01
min     4.196000e+01
25%     6.664667e+01
50%     8.249000e+01
75%     8.889473e+01
max     1.009300e+02
std     1.554184e+01
```

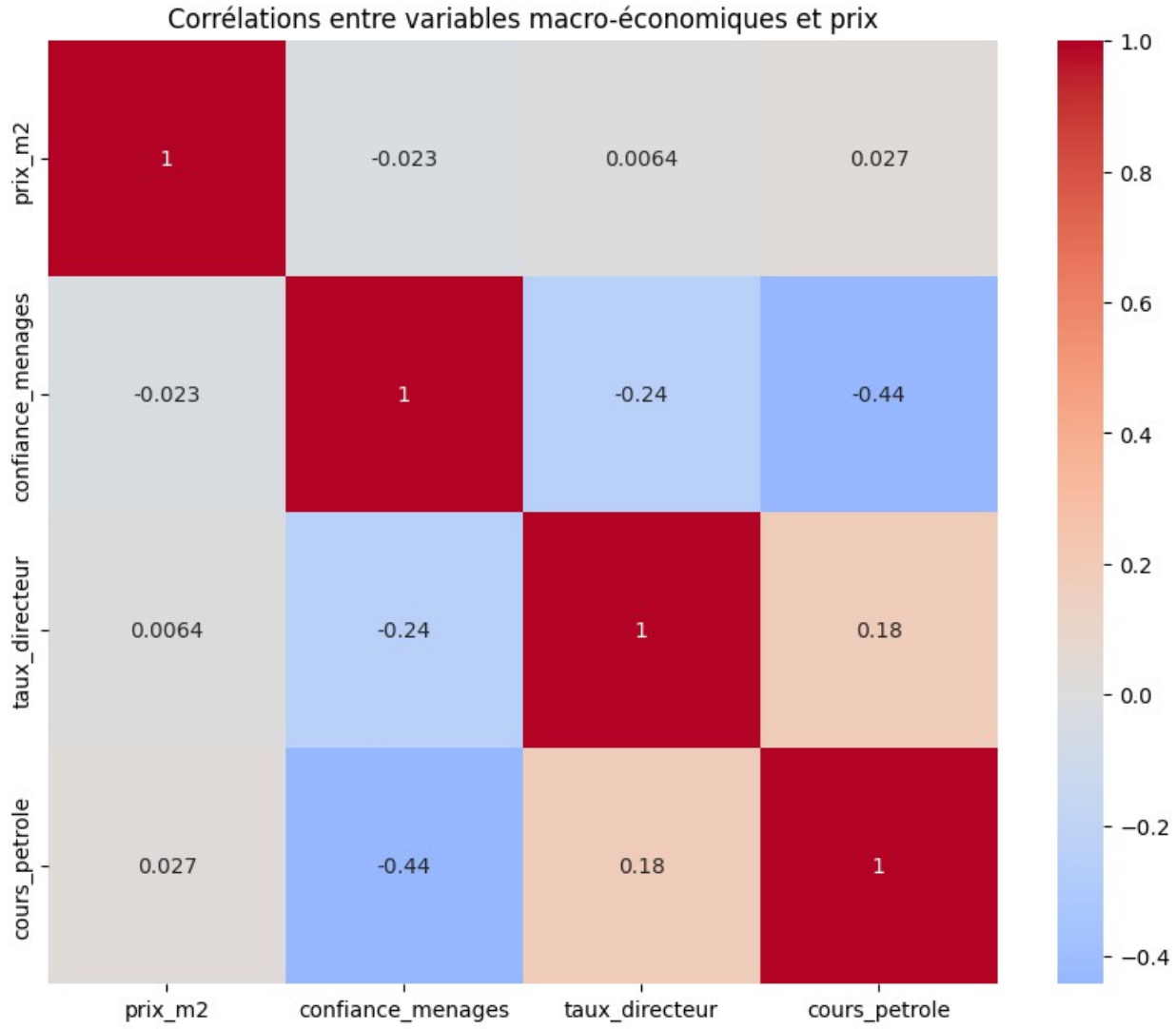
Le Volume de données est très important avec plus de 19,6 millions de transactions. Voici les caractéristiques :

- Prix moyen au m² : 4,547€ avec un écart-type de 3,818€, indiquant une forte dispersion
- Les surfaces varient considérablement (moyenne 619m², écart-type 14,011m²)
- Distribution des prix asymétrique : beaucoup de biens à prix modéré et quelques biens très chers tirant la moyenne vers le haut
- Période couverte : 2014-2024, donnant une bonne perspective historique

Les fortes de variations de prix et de distribution sont probablement dû aux différences entre départements et aux types de bien étudiés (pré ou appartement)

Corrélation des variables avec le prix au mètre carré :

```
correlations = df[['prix_m2', 'confiance_menages', 'taux_directeur',
'cours_petrole']].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlations, annot=True, cmap='coolwarm', center=0)
plt.title('Corrélations entre variables macro-économiques et prix')
plt.show()
```



Les trois variables macroéconomiques que nous avons décidé de rajouter à notre jeu de données semblent très peu corrélées avec le prix au mètre carré. Nous allons vérifier cela avec l'importance que les modèles prédictifs leur donnent.

Cependant, cela nous semble important car pour prédire l'avenir, il faut tenir compte des tendances macroéconomiques.

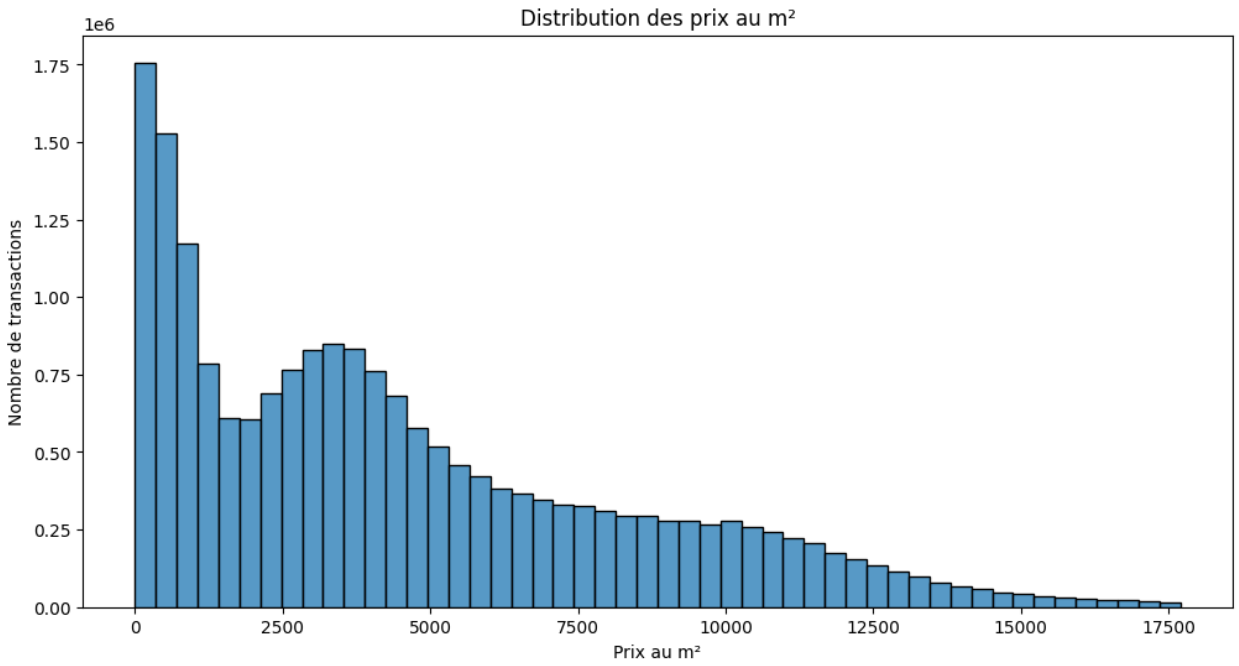
Analyse graphique

Analysons la distribution des prix au m² et leur évolution temporelle.

Distribution des prix au m2

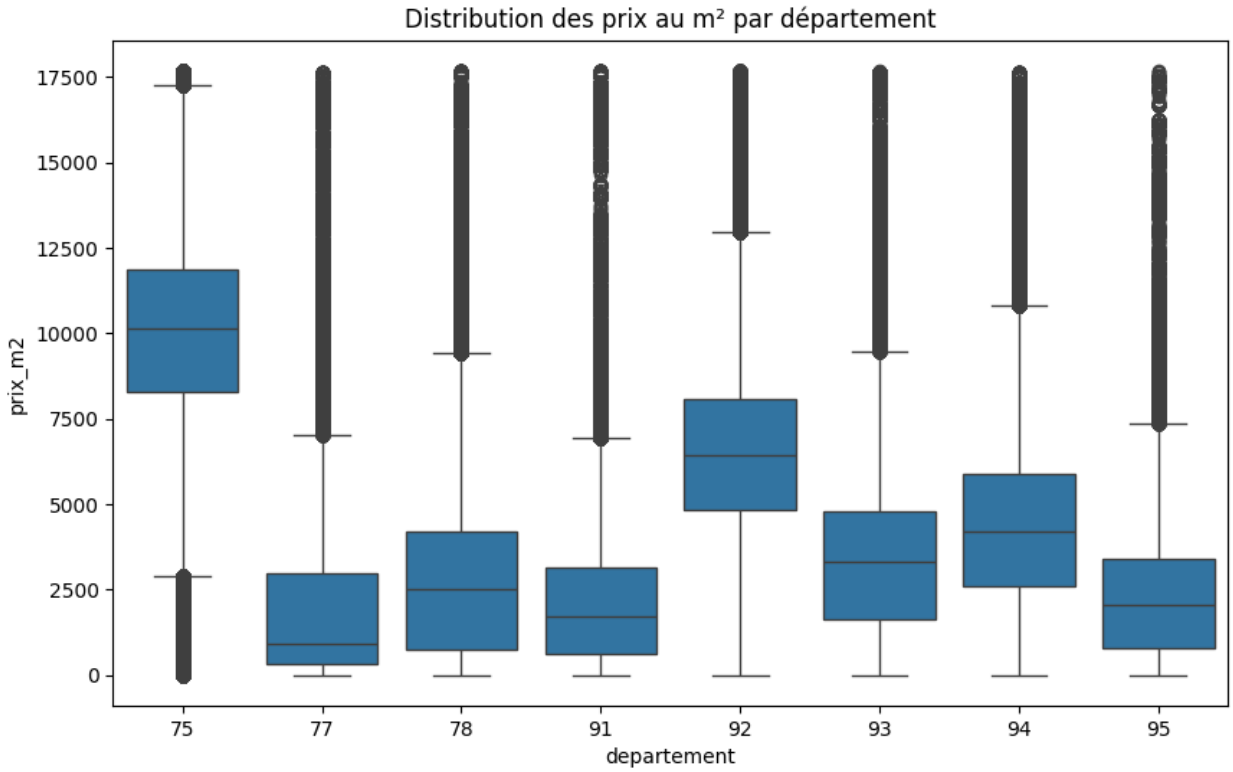
```
plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='prix_m2', bins=50)
plt.title('Distribution des prix au m²')
plt.xlabel('Prix au m²')
```

```
plt.ylabel('Nombre de transactions')
plt.show()
```



Prix moyen par département

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='departement', y='prix_m2')
plt.title('Distribution des prix au m² par département')
plt.show()
```

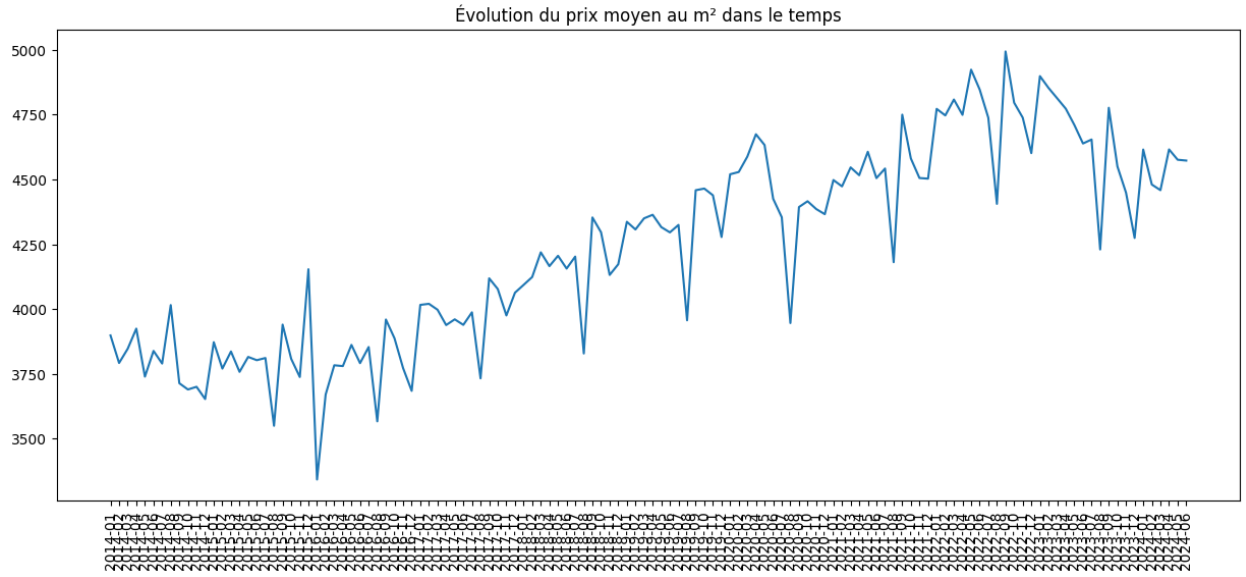


1. Paris (75) se démarque nettement :
- Médiane autour de 10,000€/m²
- Forte dispersion
- Nombreux outliers vers le haut
1. Autres départements d'Île-de-France :
- Prix médians entre 2,500€ et 5,000€/m²
- Dispersion plus modérée
- Gradient de prix clair selon la distance à Paris

Cela laisse paraître que le département est une variable clé

Evolution temporelle

```
prix_temps = df.groupby('year_month')['prix_m2'].mean().reset_index()
plt.figure(figsize=(15, 6))
plt.plot(prix_temps['year_month'].astype(str), prix_temps['prix_m2'])
plt.title('Évolution du prix moyen au m² dans le temps')
plt.xticks(rotation=90)
plt.show()
```



Nous observons une **tendance haussière constante** de 2014 à 2024 avec une progression de 3,500€/m² à 4,500€/m² (+29%)

De plus il y a une volatilité importante avec :

- *Pics ponctuels marqués*
- *Creux notables mais absorption rapide*
- *Accélération visible post-2020* (possible effet Covid)

On semble distinguer une saisonnalité visible mais modérée.

Segmentation des données

```
def segment_data(df):
    # Sélection des variables pour la segmentation
    segmentation_features = ['surface', 'prix_m2']

    # Standardisation
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(df[segmentation_features])

    # Détermination du nombre optimal de clusters avec elbow method
    inertias = []
    K = range(1, 11)
    for k in K:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(X_scaled)
        inertias.append(kmeans.inertia_)

    # Application du k-means avec le nombre optimal de clusters
    optimal_k = 4  # À ajuster selon l'elbow curve
    kmeans = KMeans(n_clusters=optimal_k, random_state=42)
    df['segment'] = kmeans.fit_predict(X_scaled)
```

```
    return df

# Modèle par segment
def train_segment_models(df, model_class):
    models = {}
    scores = {}

    for segment in df['segment'].unique():
        segment_data = df[df['segment'] == segment]
        X = segment_data.drop(['prix_m2', 'segment'], axis=1)
        y = segment_data['prix_m2']

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        model = model_class()
        model.fit(X_train, y_train)

        scores[segment] = model.score(X_test, y_test)
        models[segment] = model

    return models, scores
```

Choix et Analyse des modèles

Nous avons sélectionné trois modèles différents pour la prédiction des prix :

1. **Régression linéaire :**
   - *Avantages :* Simplicité, interprétabilité, rapidité
   - *Inconvénients :* Hypothèses fortes de linéarité, sensible aux outliers
2. **Random Forest :**
   - *Avantages :* Gestion des non-linéarités, robustesse aux outliers, peu de paramètres à régler
   - *Inconvénients :* Moins interprétable, plus lent que la régression linéaire
3. **XGBoost :**
   - *Avantages :* Performances généralement supérieures, gestion fine des non-linéarités
   - *Inconvénients :* Plus complexe à paramétrer, risque de surapprentissage

Préparation des données

```
numeric_features = ['surface', 'confiance_menages', 'taux_directeur',
'cours_petrole']
categorical_features = ['type_bien', 'departement', 'l_codinsee']
```

Encodage des variables catégorielles

```
encoders = {}
X = df[numeric_features].copy()

# POur les valeurs manquantes

numeric_imputer = SimpleImputer(strategy='mean')
X[numeric_features] =
numeric_imputer.fit_transform(X[numeric_features])

for feature in categorical_features:
    le = LabelEncoder()
    # Gestion des NaN dans les variables catégorielles
    X[feature] = df[feature].fillna('MISSING')  # Remplace les NaN par
'MISSING'
    X[feature] = le.fit_transform(X[feature].astype(str))
    encoders[feature] = le

y = df['prix_m2'].copy()

print("Nombre de valeurs manquantes par colonne :")
print(X.isna().sum())
```

```
Nombre de valeurs manquantes par colonne :
surface              0
confiance_menages    0
taux_directeur       0
cours_petrole        0
type_bien            0
departement          0
l_codinsee           0
dtype: int64
```

Division train set et data set

```
# Suppression des lignes où la variable prix_m2 est manquante
mask = ~y.isna()
X = X[mask]
y = y[mask]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print("Dimensions des ensembles d'entraînement et de test :")
print(f"X_train : {X_train.shape}")
print(f"X_test : {X_test.shape}")
```

```
Dimensions des ensembles d'entraînement et de test :
X_train : (15731692, 7)
X_test : (3932923, 7)
```

Entrainement et évaluation des modèles

Initialisation et entraînement

```
models = {
    'Régression linéaire': LinearRegression(),
    'Random Forest': RandomForestRegressor(
        n_estimators=50,  # Réduit de 100 à 50
        max_depth=5,      # Réduit de 10 à 8
        min_samples_leaf=4,  # Ajout pour accélérer
        n_jobs=-1,          # Utilisation de tous les cœurs
        random_state=42
    ),
    'XGBoost': xgb.XGBRegressor(
        max_depth=4,        # Réduit de 6 à 4
        n_estimators=50,    # Réduit de 100 à 50
        learning_rate=0.1,
        tree_method='hist', # Méthode plus rapide
        n_jobs=-1,         # Utilisation de tous les cœurs
        random_state=42
    )
}

# Entraînement et évaluation
results = {}
for name, model in models.items():
    print(f"Entraînement du modèle {name}...")
    model.fit(X_train, y_train)

    # Prédictions
    y_pred = model.predict(X_test)

    # Métriques
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    # Cross-validation
    cv_scores = cross_val_score(model, X, y, cv=5, scoring='r2')

    results[name] = {
        'RMSE': rmse,
        'R2': r2,
        'CV_R2_mean': cv_scores.mean(),
        'CV_R2_std': cv_scores.std()
    }
```

```
Entraînement du modèle Régression linéaire...
Entraînement du modèle Random Forest...
Entraînement du modèle XGBoost...
```

```
results_df = pd.DataFrame(results).T
display(results_df)
```

	RMSE	R2	CV_R2_mean	CV_R2_std
Régression linéaire	3485.448410	0.166743	-2.665765	2.572227
Random Forest	2043.330721	0.713622	-0.121692	0.961569
XGBoost	1930.865039	0.744279	-0.083344	1.058347

Analyse des différents modèles :

- XGBoost montre la meilleure performance :
  - R^2 = 0.744 (74.4% de variance expliquée)
  - RMSE = 1,930€/m²
  - Meilleure stabilité en cross-validation
- Random Forest proche :
  - R^2 = 0.714
  - RMSE = 2,043€/m²
  - Bonne robustesse
- Régression linéaire décevante :
  - R^2 = 0.167
  - RMSE = 3,485€/m²
  - Cross-validation très instable

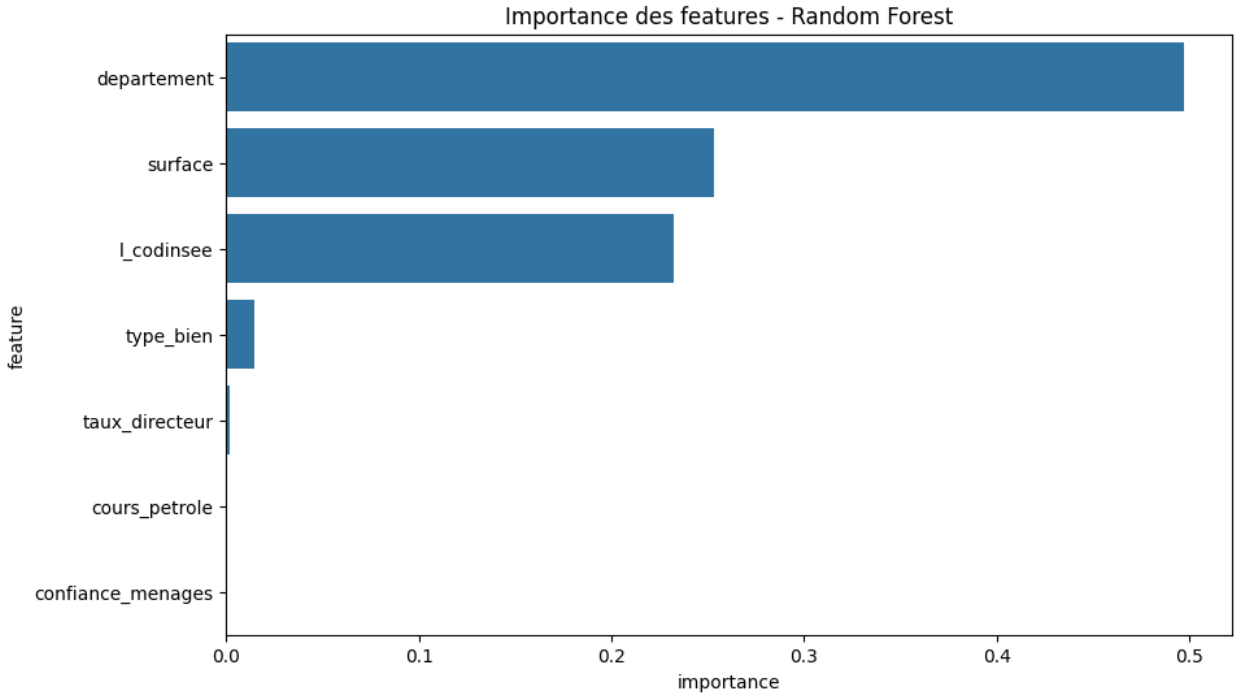
Importance des variables explicatives

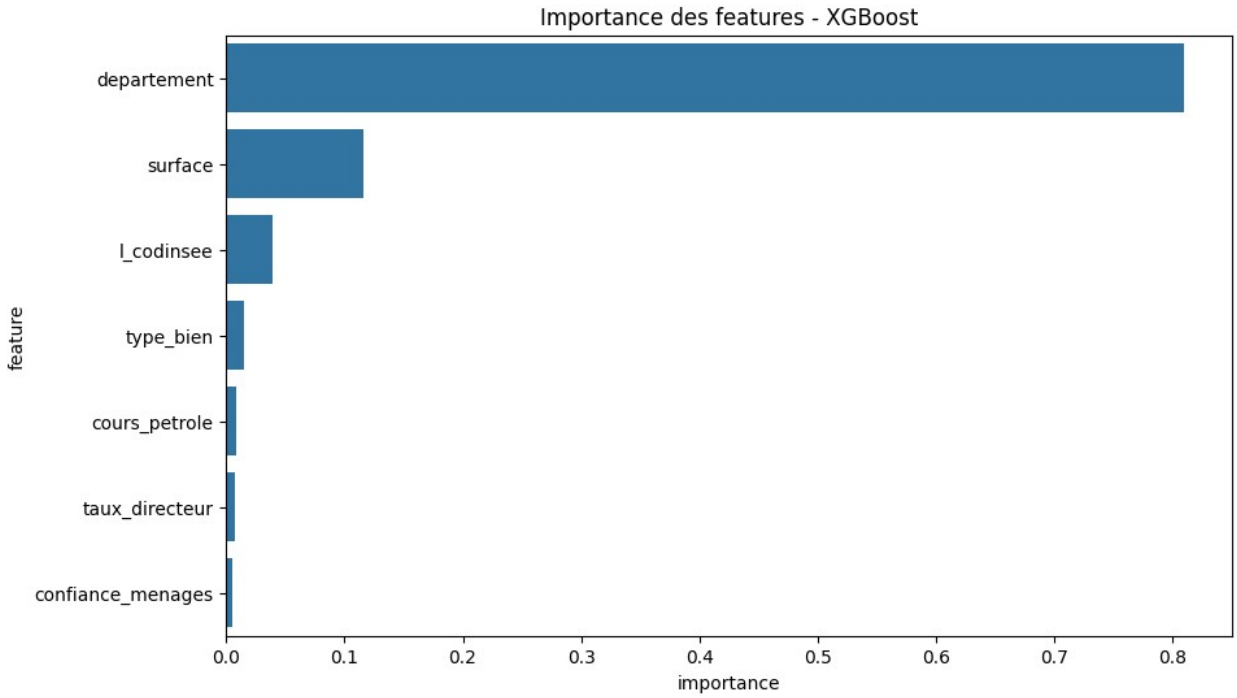
```
rf_importance = pd.DataFrame({
    'feature': numeric_features + categorical_features,
    'importance': models['Random Forest'].feature_importances_
}).sort_values('importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(data=rf_importance, x='importance', y='feature')
plt.title('Importance des features - Random Forest')
plt.show()

xg_importance = pd.DataFrame({
    'feature': numeric_features + categorical_features,
    'importance': models['XGBoost'].feature_importances_
}).sort_values('importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(data=xg_importance, x='importance', y='feature')
plt.title('Importance des features - XGBoost')
plt.show()
```





Analyse des variables explicatives :

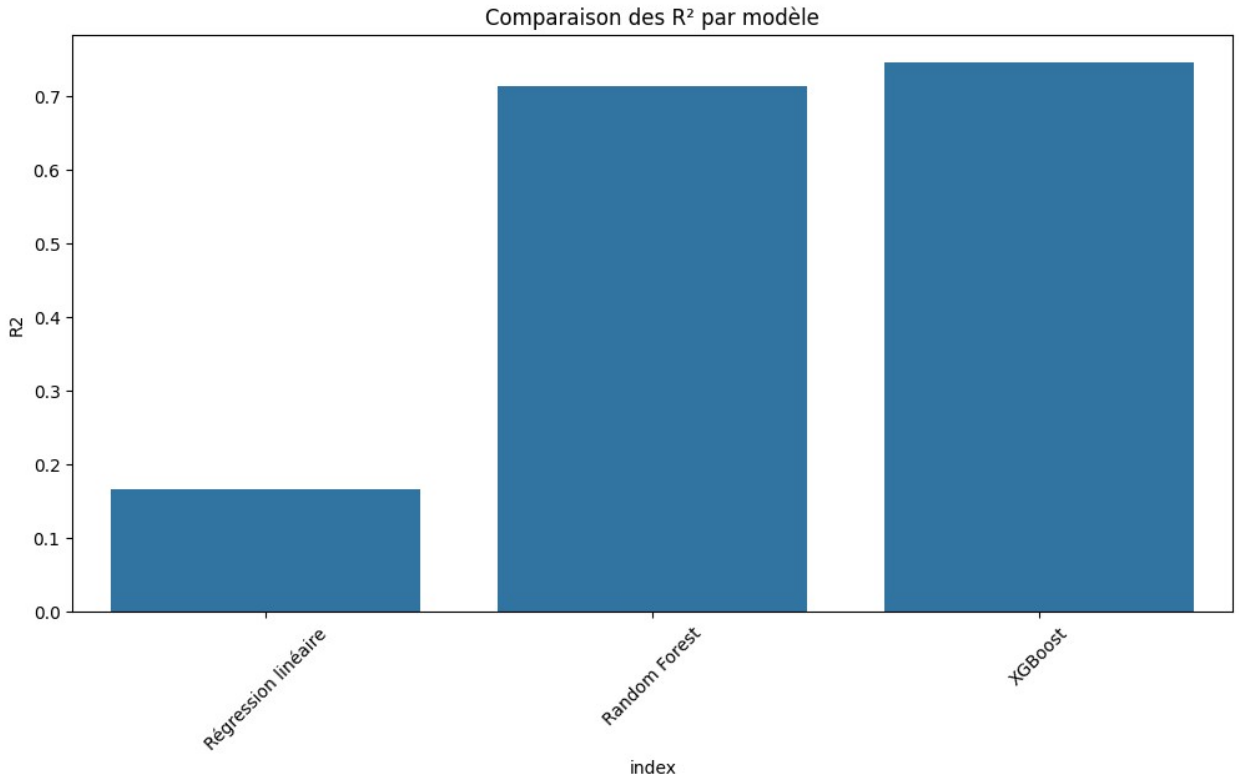
- Dominance écrasante de la localisation :
  - XGBoost : département ~80% d'importance
  - Random Forest : département ~50%
- Surface en second facteur :

  - XGBoost : ~10%
  - Random Forest : ~25%
- Variables macroéconomiques marginales :
  - Toutes < 5% d'importance
  - Confirme l'analyse des corrélations
- Code INSEE significatif pour Random Forest (~20%) mais pas pour XGBoost

Visualisation des résultats

Comparaison des résultats

```
plt.figure(figsize=(12, 6))
sns.barplot(data=results_df.reset_index(), x='index', y='R2')
plt.title('Comparaison des R² par modèle')
plt.xticks(rotation=45)
plt.show()
```

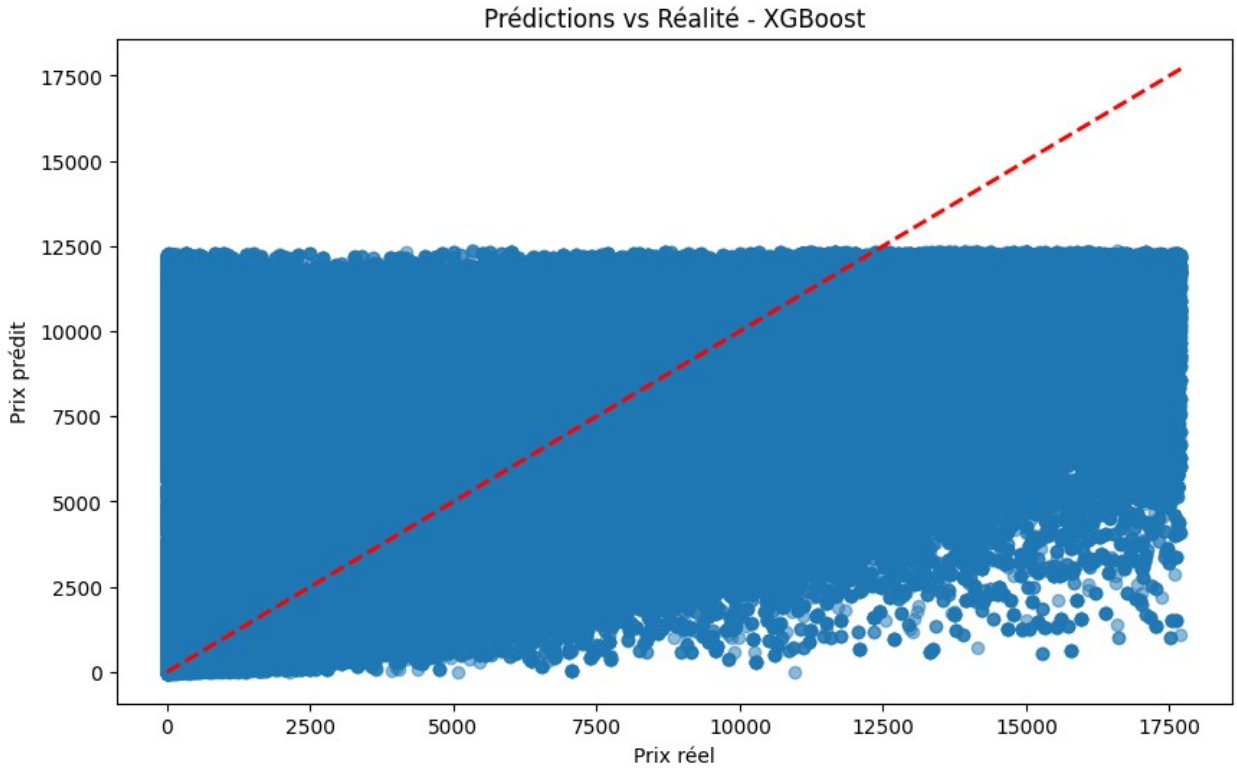


Analyse des résidus pour le meilleur modèle

```
best_model_name = results_df['R2'].idxmax()
best_model = models[best_model_name]
y_pred_best = best_model.predict(X_test)

plt.figure(figsize=(10, 6))
```

```
plt.scatter(y_test, y_pred_best, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'r--', lw=2)
plt.xlabel('Prix réel')
plt.ylabel('Prix prédit')
plt.title(f'Prédictions vs Réalité - {best_model_name}')
plt.show()
```



Analyse des résidus

Le modèle semble avoir des difficultés à prédire correctement les prix élevés (au-delà de 10000). Il tend à sous-estimer systématiquement ces valeurs. Pour les prix plus bas (inférieurs à 10000), les prédictions semblent plus précises, avec une distribution plus proche de la ligne idéale. Il y a une forte densité de points dans la partie inférieure du graphique, suggérant que la majorité des données concerne des prix plus bas.

Pistes d'amélioration

- Un rééchantillonnage des données pour mieux représenter les prix élevés
- Une transformation logarithmique des prix pour mieux gérer les grandes valeurs
- L'ajout de features plus pertinentes pour la prédiction des prix élevés
- L'utilisation d'un ensemble de modèles spécialisés par gamme de prix

Conclusion

L'analyse du marché immobilier francilien révèle une dynamique où la géographie prime sur l'économie. Notre modélisation XGBoost, avec un R^2 de 0.74, démontre que le département est le facteur déterminant des prix, loin devant les variables macroéconomiques (corrélations < 0.03). Cette faible influence des indicateurs économiques traditionnels (taux directeur, confiance des ménages, cours du pétrole) suggère un marché résilient aux cycles économiques.

La comparaison entre Paris (médiane ~10,000€/m²) et sa périphérie (2,500-5,000€/m²) illustre une forte segmentation géographique. L'évolution 2014-2024 montre une hausse continue des prix (+29%), accélérée post-2020, témoignant d'un marché dynamique malgré les crises.

Bien que performant, le modèle pourrait être amélioré par l'intégration de données locales (transports, services) et une segmentation plus fine.