RAPPORT DU PROJET DE DATA SCIENCE

Modèle de prédiction des prix des immobiliers

Réalisé par : **HASSOUNE Naima**
MARGAL Halima
DIBE Imane
ESSAJIDE Fatima zahra
El ALAOUI Khadija
EL MRABTI Imane

Professeur encadrant : M. **Aimad QAZDAR**

Résumé (Abstract) :

Le présent projet de data science vise à développer un modèle de prédiction des prix immobiliers en utilisant des techniques avancées d'analyse de données. À travers une exploration approfondie des données immobilières, comprenant des caractéristiques telles que la localisation, la taille, les équipements, etc., nous avons construit un modèle prédictifs basés sur des algorithmes de machine learning.

Après une phase minutieuse de collecte et de nettoyage des données, nous avons appliqué des méthodes de prétraitement pour normaliser les variables et gérer les valeurs aberrantes. L'analyse exploratoire des données a permis de mettre en lumière des tendances significatives et des relations entre les différentes caractéristiques.

Le cœur de notre travail réside dans la modélisation, où nous avons utilisé plusieurs algorithmes de régression et d'apprentissage automatique pour prédire avec précision les prix immobiliers. Des métriques de performance détaillées et des validations croisées ont été utilisées pour évaluer la robustesse de nos modèles.

Les résultats obtenus démontrent une capacité significative à prédire les prix des biens immobiliers avec une précision notable. La discussion des résultats met en évidence les facteurs clés qui influent sur les prix, fournissant ainsi des informations exploitables pour les acteurs du marché immobilier.

En conclusion, ce projet offre une contribution importante à la prédiction des prix immobiliers en exploitant les capacités de la data science. Les implications pratiques de cette analyse peuvent être utilisées par les professionnels de l'immobilier, les investisseurs et les décideurs pour prendre des décisions éclairées dans le domaine de l'immobilier.

1. Introduction

Objectif du projet

Importance de la prédiction des prix immobiliers

2. Collecte et Compréhension des Données

2.1 Sources de Données

Extraction des données de la plateforme Kaggle

Jeu de données "Housing Prices Competition for Kaggle Learn Users"

2.2 Description des Variables

Caractéristiques du jeu de données sur les propriétés à New York

Prix, Surface, Chambres, Salles de bains, Étages, etc.

2.3 Méthodes de Collecte de Données

Respect des conditions d'utilisation de Kaggle

Téléchargement et préparation des jeux de données

2.4 Exploration Initiale des Données

Analyse des valeurs manquantes, lignes dupliquées, et statistiques descriptives

Identification des outliers et visualisation des données

3. Analyse des Données

3.1 Corrélation Matrix

Impact des variables sur le prix des propriétés

Relations entre la superficie, le nombre de chambres, salles de bains, etc.

3.2 Sélection des Features

Méthode de filtrage basée sur le coefficient de corrélation

Méthode Wrapper avec RandomForestRegressor

4. Développement du Modèle

4.1 Division du Data

Séparation des features (X) et de la variable cible (y)

Division en jeux d'entraînement (80%) et de test (20%)

4.2 Sélection et Entraînement du Modèle

Utilisation de la régression linéaire comme premier modèle

Evaluation de la performance du modèle

5. Évaluation du Modèle

Calcul des métriques R^2, erreur moyenne carrée, erreur absolue moyenne

Comparaison des modèles : Linear Regression, RandomForest Regressor, XGBRegressor

Introduction

1. Contexte du Projet

En tant que l'un des piliers fondamentaux de l'économie mondiale, le marché immobilier joue un rôle essentiel dans la vie de chacun, que ce soit en tant qu'acheteur, vendeur, locataire, investisseur ou simplement en tant que membre de la communauté. Comme le domaine de ce projet de data science se situe dans le secteur immobilier, plus spécifiquement dans l'évaluation des biens immobiliers. Il s'agit de l'ensemble des activités liées à l'estimation de la valeur financière d'une propriété(maison).la prédiction des prix des biens de ce marché et aussi importante comme il permet d'anticiper les tendances du marché d'optimiser les investissements et de minimiser les risques financières .

2. Objectifs du Projet

L'objectif de cette analyse est d'identifier les facteurs clés influençant les prix des logements à New York et de développer un modèle de prédiction des prix(en utilisant la régression linéaire par python) basé sur des données et des variables clés. Cette analyse sera bénéfique pour évaluer les offres pour les acheteurs et les investisseurs et la fixation des prix sur le marché pour les vendeurs. Autrement dit, pour comprendre le marché immobilier de la ville et prendre des décisions éclairées concernant la location ,d'achat, de vente ou d'investissement, ainsi que pour évaluer la rentabilité potentielle des propriétés.

3. Questions de Recherche

Pour atteindre nos objectifs, nous nous sommes posé plusieurs questions de recherche fondamentales :

Quels sont les facteurs clés qui influent sur les prix immobiliers ?

Quelle est la performance comparative des différents algorithmes de prédiction ?

Comment les résultats obtenus peuvent-ils être interprétés pour prendre des décisions éclairées dans le domaine de l'immobilier ?

4. Méthodologie de Travail

Pour répondre à ces questions, nous avons suivi une méthodologie rigoureuse qui englobe la collecte de données, le prétraitement, l'analyse exploratoire, la modélisation, et enfin, l'évaluation des résultats. Chaque étape de notre approche est détaillée dans les sections suivantes du rapport.

N'oubliez pas d'adapter cette introduction en fonction des détails spécifiques de votre projet, en mettant en évidence les aspects uniques et les défis particuliers auxquels vous avez été confronté.

2. Collecte et Compréhension des Données

2.1 Sources de Données

Les données utilisées dans le cadre de ce projet ont été extraites de la plateforme Kaggle, spécialisée dans le partage de jeux de données et la tenue de compétitions en data science. La source spécifique est le jeu de données "Housing Prices Competition for Kaggle Learn Users", qui propose une variété d'informations sur des propriétés résidentielles.

2.2 Description des Variables

Dans ce projet on va travailler sur une dateset qu'elle s'agit d'une base de données sur les caractéristiques des propriétés à NewYork prise de Kaggle. Ces Données contiennent :

Price: prix de l'immobilier

Area : surface

Bedrooms : chambres

Bathrooms:Salle de bains.

Stories: étages

Mainroad : route principale

Guestroom : chambre d'amis

Basement : sous-sol

Hotwaterheating : chauffage à l'eau chaude

Airconditioning: climatisation

Parking

Prefarea: zone préférée

Furnishingstatuts: statut d'ameublement

2.3 Méthodes de Collecte de Données

La collecte des données a été effectuée de manière éthique, en respectant les conditions d'utilisation fournies par Kaggle. Les jeux de données d'entraînement et de test ont été téléchargés et préparés pour une analyse approfondie.

2.4 Exploration Initiale des Données

Avant de plonger dans le processus de modélisation, une exploration initiale des données a été entreprise. Cela inclut une inspection des premières lignes du jeu de données, des statistiques descriptives, et une compréhension préliminaire des

distributions des variables. Cette étape préliminaire permet de détecter d'éventuelles anomalies et d'orienter les choix de prétraitement nécessaires.

On importe les données :

```
import pandas as pd
data= pd.read_csv('Housing.csv')
```

```
data
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished
...	...	...	...	...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	yes	no	yes	no	no	2	no	unfurnished
541	1767150	2400	3	1	1	no	no	no	no	no	0	no	semi-furnished
542	1750000	3620	2	1	1	yes	no	no	no	no	0	no	unfurnished
543	1750000	2910	3	1	1	no	no	no	no	no	0	no	furnished
544	1750000	3850	3	1	2	yes	no	no	no	no	0	no	unfurnished

545 rows × 13 columns

On aboutit a des informations des données :

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

Donc on a 7 variables catégoriels dans nos données qui sont : mainroad, guestroom, basement, hotwaterheating, airconditioning,

prefarea,furnrishingstatus et pour les autres variables sont de type entier .

- Identification des valeurs manquantes

```
import numpy as np
data.isnull()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
540	False	False	False	False	False	False	False	False	False	False	False	False	False
541	False	False	False	False	False	False	False	False	False	False	False	False	False
542	False	False	False	False	False	False	False	False	False	False	False	False	False
543	False	False	False	False	False	False	False	False	False	False	False	False	False
544	False	False	False	False	False	False	False	False	False	False	False	False	False

545 rows × 13 columns

Pour resultats plus précise :

```
data.isnull().sum()
```

```
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           0
prefarea          0
furnishingstatus  0
dtype: int64
```

- Identification des lignes dupliquées

```
data[data.duplicated()]
```

price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus

Il n’ y a pas de lignes dupliquées dans nos données .

Statistiques des données

```
data.describe()
```

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

Prix (Price) : La moyenne est d'environ 4,766,729 avec une dispersion de 1,870,440. Les prix varient de 1,750,000 à 13,300,000.

Surface (Area) : La moyenne de la surface est 5150, avec une dispersion de 2170. Les valeurs vont de 1650 à 16200.

Chambres (Bedrooms) : En moyenne, il y a près de 3 chambres par mobilière, avec une variation de 1 à 6.

Salles de bains (Bathrooms) : En moyenne, chaque mobilière a environ 1,3 salle de bain, avec une variation de 1 à 4.

Étages (Stories) : La moyenne est d'environ 1,8 étage par mobilière, avec une variation de 1 à 4.

Parking : En moyenne, il y a une disponibilité de parking pour environ 69% des mobilières.

Résumé :

En synthèse, on observe une diversité notable dans la région en ce qui concerne la taille, la disposition des pièces et les équipements des mobilières, ce qui se traduit par une variété significative des prix. La disponibilité de places de parking témoigne d'une prise en compte attentive des besoins pratiques des habitants de la région.

- Détecte des outliers

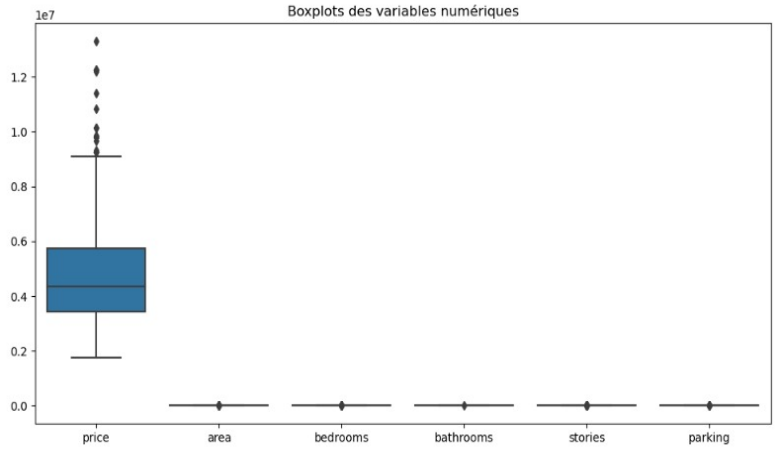
  On a utilisé la méthode IQR pour la détection des outliers, elle est basée sur la définition des limites supérieure et inférieure en utilisant l'amplitude interquartile (IQR)

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

# Sélectionner les colonnes numériques (si vous avez des variables non numériques, vous pouvez les exclure)
numeric_columns = data.select_dtypes(include=['float64', 'int64']).columns

# Calculer le quartile Q1 et Q3 pour chaque colonne numérique
Q1 = data[numeric_columns].quantile(0.25)
Q3 = data[numeric_columns].quantile(0.75)

# Calculer l'amplitude interquartile (IQR)
IQR = Q3 - Q1

# Définir la borne inférieure (lower bound) et la borne supérieure (upper bound) pour détecter les outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identifier les outliers dans chaque colonne numérique
outliers = ((data[numeric_columns] < lower_bound) | (data[numeric_columns] > upper_bound))

# Compter le nombre d'outliers par colonne
outliers_count = outliers.sum()

# Afficher le nombre d'outliers par colonne
print("Nombre d'outliers par colonne:")
print(outliers_count)

# Afficher un graphique des boxplots pour chaque colonne numérique
plt.figure(figsize=(12, 6))
sns.boxplot(data=data[numeric_columns])
plt.title('Boxplots des variables numériques')
plt.show()
```

Affichage du résultat :

```
Nombre d'outliers par colonne:
price        15
area         12
bedrooms     12
bathrooms     1
stories      41
parking      12
dtype: int64
```

Boxplots :



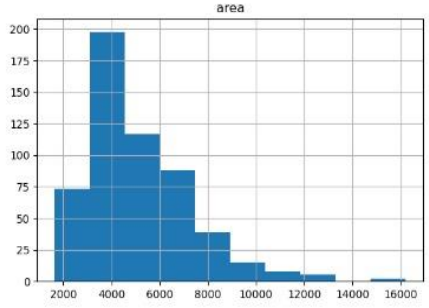
La présence d'outliers dans les données, décelée à travers des valeurs exceptionnelles dans différentes colonnes telles que le prix, la surface, le nombre de chambres, de salles de bains, le nombre d'étages et le parking, suggère des observations qui se démarquent significativement du reste de l'ensemble de données. Ces valeurs pourraient résulter de caractéristiques uniques, de localisations spécifiques ou même d'erreurs de collecte de données , ils indiquent les particularités potentielles dans la région

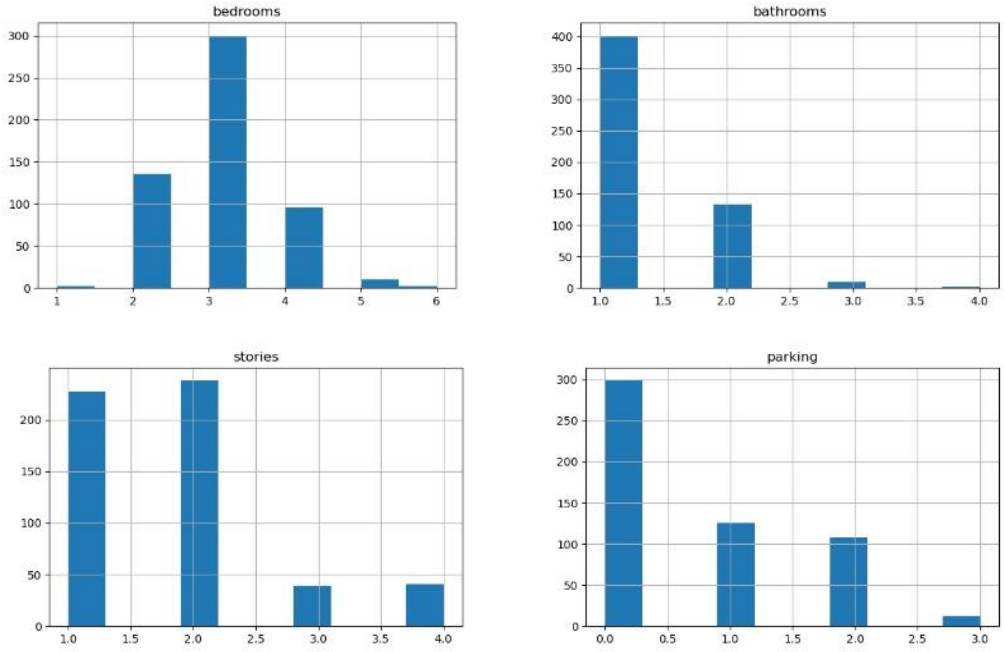
Visualisation des données

```
from matplotlib import pyplot as plt
data.hist(figsize=(15,15))
```

```
array([[<Axes: title={'center': 'price'}>,
        <Axes: title={'center': 'area'}>],
       [<Axes: title={'center': 'bedrooms'}>,
        <Axes: title={'center': 'bathrooms'}>],
       [<Axes: title={'center': 'stories'}>,
        <Axes: title={'center': 'parking'}>]], dtype=object)
```

- *Les variables numériques*
  En utilisant les histogrammes qui illustre la distribution des données

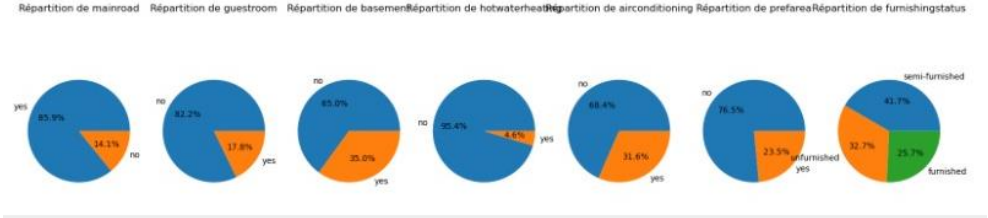




D'après la visualisation, on remarque que la majorité ont des prix situés entre 3 millions et 6 millions, tandis que la plupart présentent une superficie comprise entre 2000 et 3000. En ce qui concerne le nombre de chambres, la tendance dominante est d'avoir soit 3, soit 2 chambres dans la plupart des maisons, et pour les salles de bains, la préférence va soit vers 1, soit vers 2. En outre, la majorité des maisons ont soit 2 étages, soit un seul, avec la plupart bénéficiant d'une place de parking.

- *Les variables catégorielles*

```
# Créez un subplot pour afficher plusieurs pie charts
fig, axes = plt.subplots(1, 7, figsize=(20, 5))

# Liste des colonnes pour lesquelles vous voulez créer des pie charts
columns = ['mainroad','guestroom','basement','hotwaterheating','airconditioning','prefarea','furnishingstatus']

for i, column in enumerate(columns):
    variable_pie_chart = data[column].value_counts()
    axes[i].pie(variable_pie_chart, labels=variable_pie_chart.index, autopct='%1.1f%%')
    axes[i].set_title(f"Répartition de {column}")
    axes[i].axis('equal')

plt.show()
```

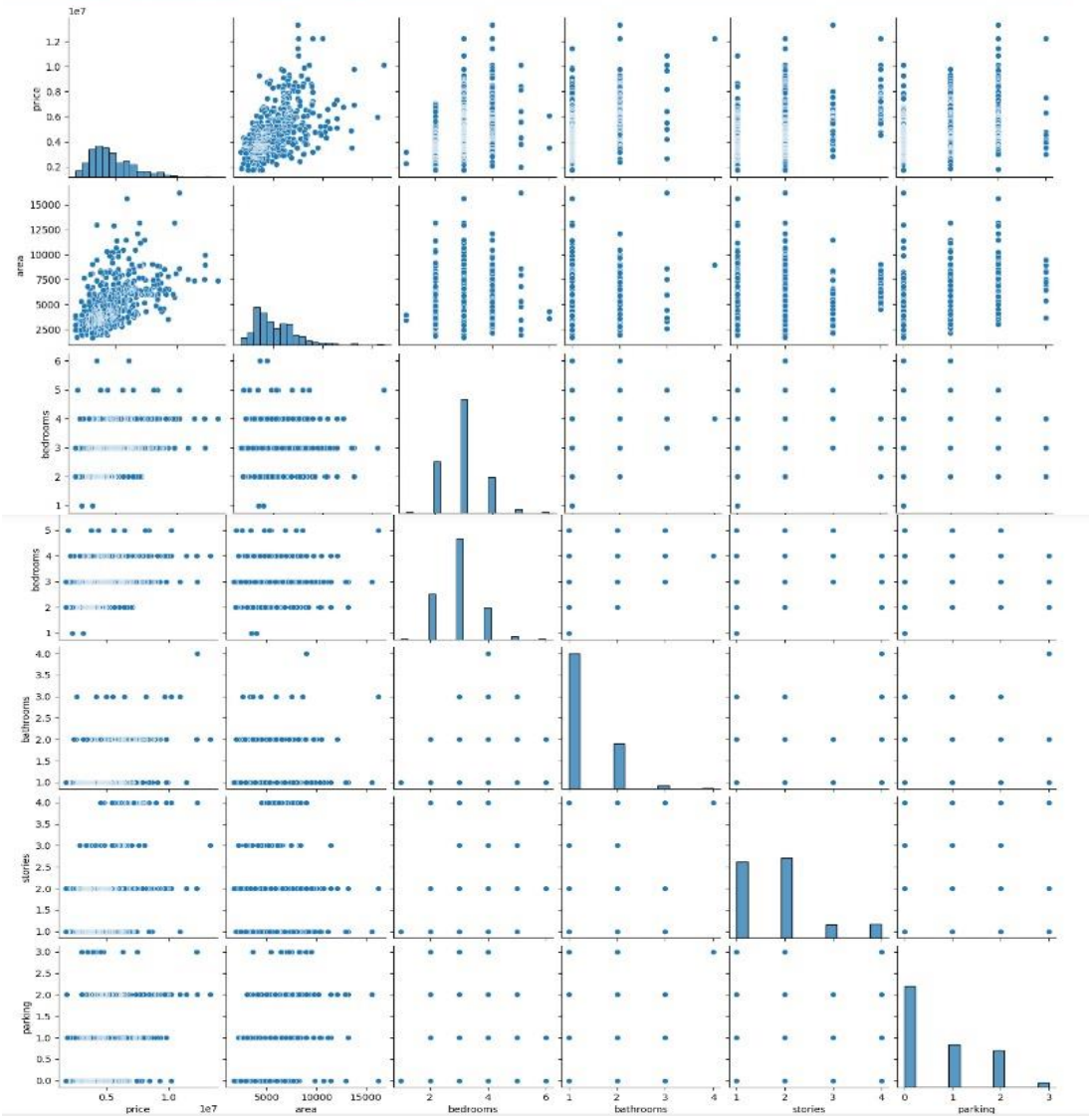


D'après les graphiques, il est observé que la majorité des logements sont situés à proximité d'une route principale. Cependant, la disponibilité d'une chambre d'amis (guestroom) est limitée, n'étant présente que dans 17.8% des maisons. Le sous-sol est présent dans 65% des cas et absent dans 35%. En ce qui concerne le chauffage à l'eau chaude, il est peu répandu, étant disponible dans seulement

4.6% des maisons. la climatisation n'est pas disponible dans 76.5% des cas. En ce qui concerne le statut d'ameublement, la plupart des maisons sont partiellement meublées.

```
import seaborn as sns
sns.pairplot(data)
plt.show()
```

```
C:\Users\pc\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



- Encoder variables catégoriels en utilisant label encoder

On a encoder les variables catégoriels avec label encoder qui garde la même dimension des données .Pour tous les variables ila transformé no en 0 et yes en 1,et pour la dernière variable(furnishingstatus) il a donné a ‘furnished’ la valeur 0 et ‘semifurnished’ la valeur 1 et a ‘unfurnished’ la valeur 2 .

```
data1 = data.copy()
from sklearn import preprocessing


label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Warehouse_block' and 'Mode_of_Shipment'
data1['mainroad']= label_encoder.fit_transform(data['mainroad'])
data1['mainroad'].unique()
data1['guestroom']= label_encoder.fit_transform(data['guestroom'])
data1['guestroom'].unique()
data1['basement']= label_encoder.fit_transform(data['basement'])
data1['basement'].unique()
data1['hotwaterheating']= label_encoder.fit_transform(data['hotwaterheating'])
data1['hotwaterheating'].unique()
data1['airconditioning']= label_encoder.fit_transform(data['airconditioning'])
data1['airconditioning'].unique()
data1['prefarea']= label_encoder.fit_transform(data['prefarea'])
data1['prefarea'].unique()
data1['furnishingstatus']= label_encoder.fit_transform(data['furnishingstatus'])
data1['furnishingstatus'].unique()
data1
```

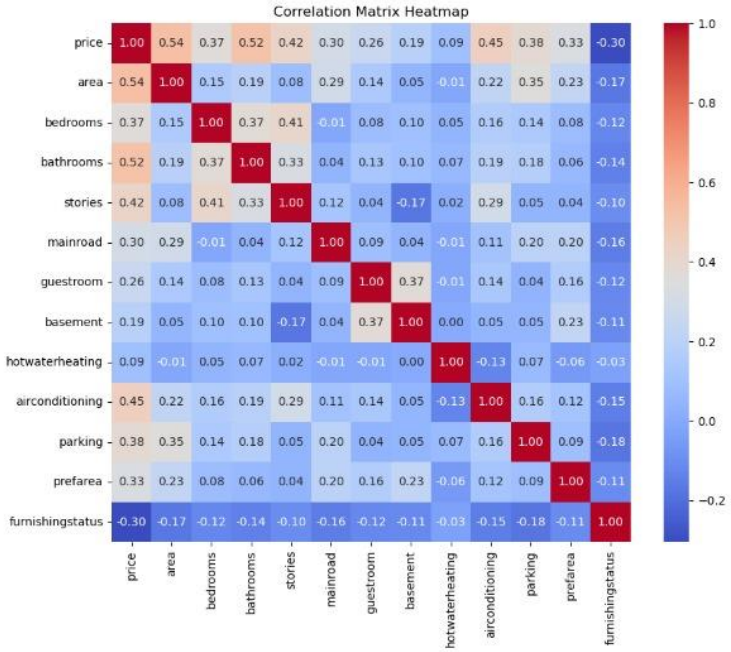
On obtient les données suivants :

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	0
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	0
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	1
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	0
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	1	0	1	0	0	2	0	2
541	1767150	2400	3	1	1	0	0	0	0	0	0	0	1
542	1750000	3620	2	1	1	1	0	0	0	0	0	0	2
543	1750000	2910	3	1	1	0	0	0	0	0	0	0	0
544	1750000	3850	3	1	2	1	0	0	0	0	0	0	2

545 rows × 13 columns

- **Matrice de corrélation**

```
# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(data1.corr(), annot=True,cmap='coolwarm', fmt='.2f', square=True)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



on se basant sur la matrice de corrélation on remarque que le prix des propriétés est influencé par plusieurs facteurs. Une corrélation positive est observée entre le prix et la superficie, indiquant que des surfaces plus grandes sont associées à des coûts plus élevés. De plus, le nombre de salles de bains et la présence de climatisation ont également un impact sur le prix. Par ailleurs, le nombre d'étages est corrélé au nombre de chambres, suggérant que des maisons avec plusieurs chambres tendent à avoir plus d'un étage. Enfin, la superficie influence la disponibilité de places de parking, montrant que la taille de la propriété détermine la présence de parkings.

- La sélection des features

On a utilisé deux méthodes pour déterminer les features pertinents dans notre données (data) pour améliorer les predictions :

1 – méthode de filtrage : on a utilisé l’une des propriétés statistiques , Il s’agit du coefficient de corrélation des variables avec le prix (variable cible). on a utilisé la valeur absolue du coefficient de corrélation qui signifie la force de la relation entre deux variables plus qu'il se proche de 1 on a une forte relation et lorsqu'on se proche de 0 cela signifie que la relation est faible : On a obtenu les résultats suivants

```
# Calculer la corrélation des autres fonctionnalités avec 'price'
correlation_with_price = data1.drop('price', axis=1).corrwith(data['price'])

# Afficher la corrélation avec 'price'
print("Corrélation avec 'price':")
print(np.abs(correlation_with_price))
```

Résultat :

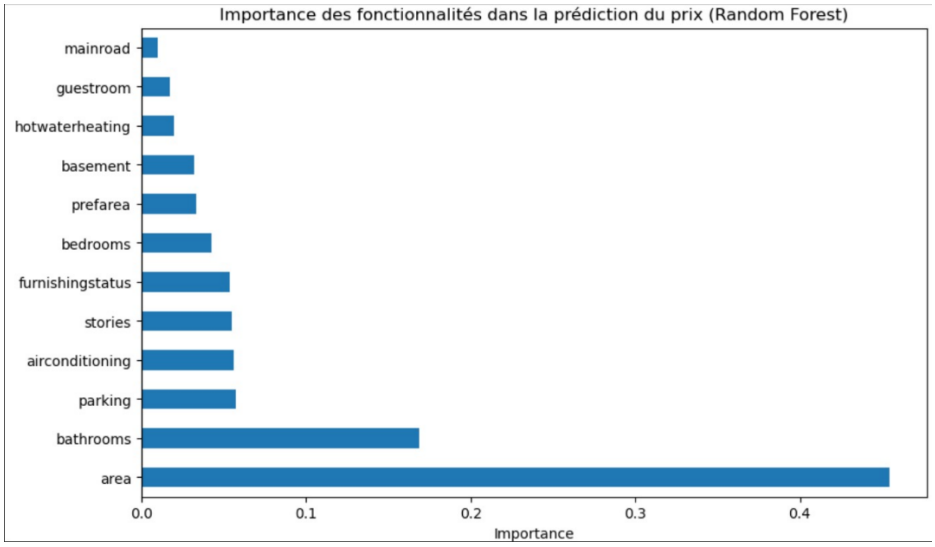
```
Corrélation avec 'price':
area                0.535997
bedrooms            0.366494
bathrooms           0.517545
stories             0.420712
mainroad            0.296898
guestroom           0.255517
basement            0.187057
hotwaterheating     0.093073
airconditioning     0.452954
parking             0.384394
prefarea            0.329777
furnishingstatus    0.304721
dtype: float64
```

2 – Méthode Wrapper : On a utilisé le modèle RandomForestRegressor qui implique l'utilisation de l'importance des fonctionnalités intégrée. Après l'entraînement du modèle, les fonctionnalités sont évaluées en fonction de leur contribution à la réduction de l'erreur. Les fonctionnalités les plus importantes peuvent être extraites à l'aide de l'attribut "feature_importances_" du modèle.

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Séparer les données en variables explicatives (X) et variable cible (y)
X = data1.drop('price', axis=1)
y = data1['price']

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialiser le modèle de forêt aléatoire
model = RandomForestRegressor()

# Ajuster le modèle aux données d'entraînement
model.fit(X_train, y_train)

# Obtenir l'importance des fonctionnalités
feature_importances = pd.Series(model.feature_importances_, index=X.columns).sort_values(ascending=False)

# Afficher l'importance des fonctionnalités
print("Importance des fonctionnalités:")
print(feature_importances)

# Afficher un graphique des importances
plt.figure(figsize=(10, 6))
feature_importances.plot(kind='barh')
plt.title('Importance des fonctionnalités dans la prédiction du prix (Random Forest)')
plt.xlabel('Importance')
plt.show()
```

```
Importance des fonctionnalités:
area                0.454391
bathrooms           0.168658
parking             0.057040
airconditioning     0.055945
stories             0.055090
furnishingstatus    0.053529
bedrooms            0.042552
prefarea            0.033593
basement            0.032132
hotwaterheating     0.019825
guestroom           0.017587
mainroad            0.009659
dtype: float64
```

On a obtenu le résultat suivant :



Et d’après les résultats obtenus , On a sélectionné features suivants :bathrooms, stories, airconditioning, parking, bedrooms, furnishingstatus,

Prefarea ,mainroad, guestroom (ils ont un coefficient de corelation absolue proche de 1 et sont important avec randomforrestregressor).

Le développement du modèle

1- La dévision du data

-Séparé les données en features qu'on a sélectionné auparavant (X) et la variable cible (y) (prix).

-Divisé data en une partie d'entrainement 80% des données originales et une partie du test 20%

```
Entrée [22]: X = data1[['area','bathrooms','stories','airconditioning','parking','bedrooms','furnishingstatus','prefarea','mainroad','guestro
             y = data1['price']

Entrée [23]: from sklearn.model_selection import train_test_split
             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2- sélection et entrainement du modèle

- **Regression linèaire**

On a sélectionné le modèle de régression linéaire et on l'a entrainé sur notre data

```
from sklearn.linear_model import LinearRegression
# Model selection and training

model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

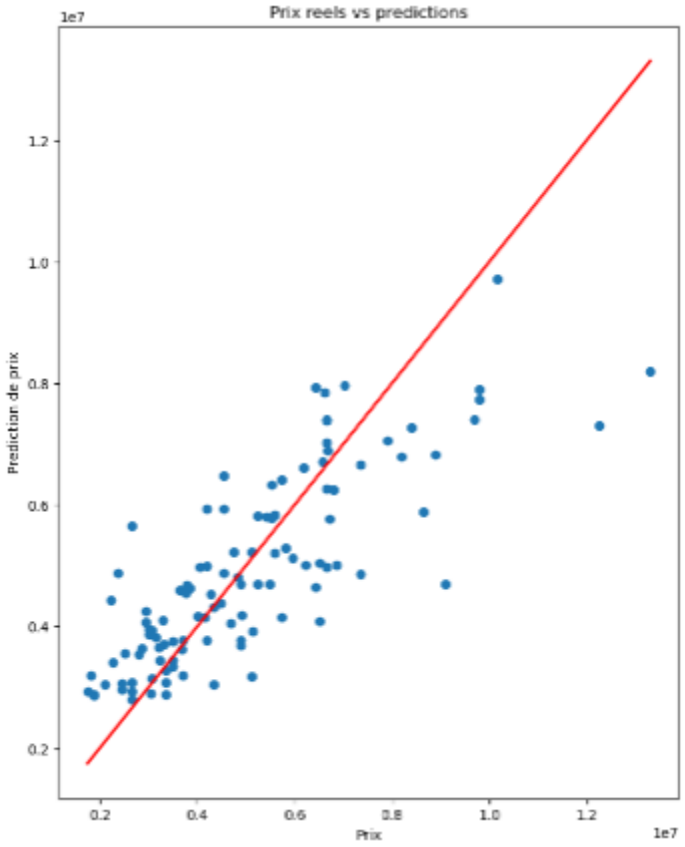
L'évaluation du modèle

On a évalué le modèle précédant en calculant ces métriques :R carrée(R^2),erreur moyenne carré (MSE) et erreur absolue moyenne (MAE) et on a trouvé les résultats suivants:

```
from sklearn.metrics import  r2_score, mean_squared_error, mean_absolute_error

model_train_preds = model.predict(X_train)
y_pred = model.predict(X_test)

# Evaluation scores

# Calcu du coefficient de détermination (R²)
r2 = r2_score(y_test, y_pred)

# Calcul de l'erreur quadratique moyenne (RMSE)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

# Calcul de l'erreur absolue moyenne (MAE)
mae = mean_absolute_error(y_test, y_pred)

print(f"Squared R (R²): {r2}")
print(f"Mean Squared Error (Test): {rmse}")
print(f"Mean absolute error (MAE): {mae}")
```

```
Squared R (R²): 0.6357622384656229
Mean Squared Error (Test): 1356858.6337759488
Mean absolute error (MAE): 987959.429239644
```

On a essayé d'illustrer la prédiction et les valeurs réelles dans le graphe suivant:

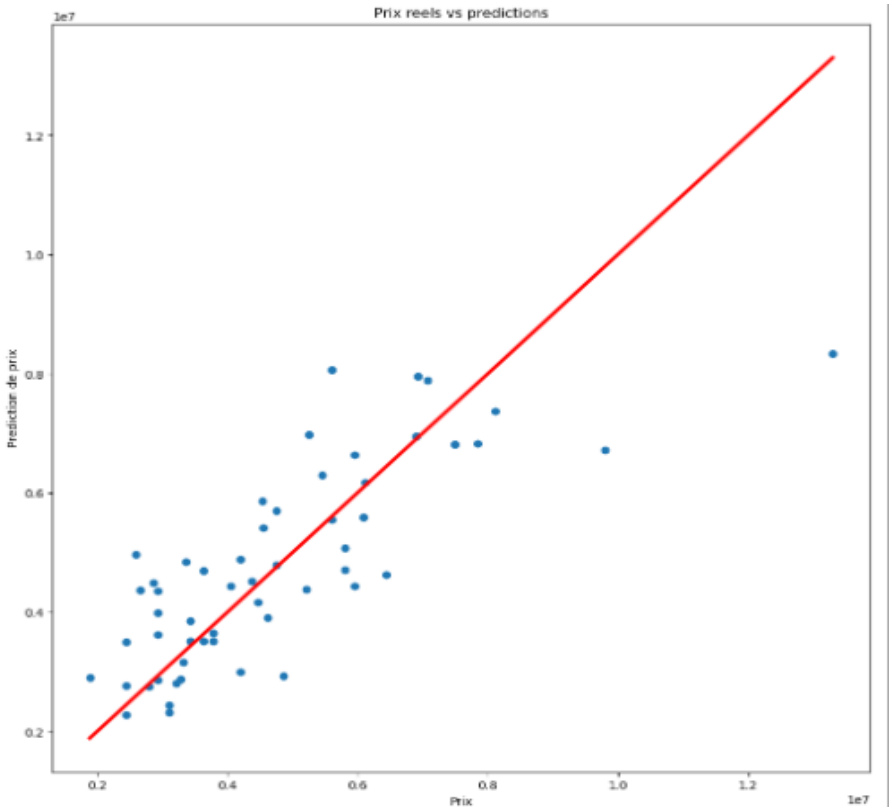


- **RandomForestRegressor :** De méme on a fait pour le modele de random forest regressor et on a obtenu le score suivant :

```
X = data1[['area','bathrooms','stories','airconditioning','parking','bedrooms','furnishingstatus','prefarea','mainroad','guestroo
y = data1['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

from sklearn import ensemble
rf = ensemble.RandomForestRegressor()
rf.fit(X_train, y_train)
y_rf = rf.predict(X_test)
print(rf.score(X_test,y_test))

plt.figure(figsize=(12,12))
plt.scatter(y_test, y_rf)
plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()], color='red', linewidth=3)
plt.xlabel("Prix")
plt.ylabel("Prediction de prix")
plt.title("Prix reels vs predictions")
```

```
0.6350888386777787
```

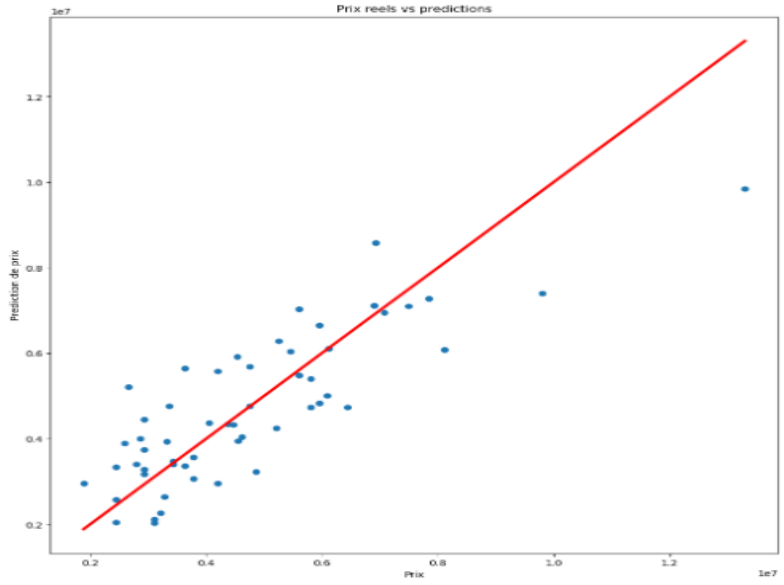
Et le graphe suivant :



- **XGBRegressor :** (même chose)

```
import xgboost as XGB
xgb  = XGB.XGBRegressor()
xgb.fit(X_train, y_train)
y_xgb = xgb.predict(X_test)
print(xgb.score(X_test,y_test))

plt.figure(figsize=(12,12))
plt.scatter(y_test, y_xgb)
plt.plot([y_test.min(),y_test.max()],[y_test.min(),y_test.max()], color='red', linewidth=3)
plt.xlabel("Prix")
plt.ylabel("Prediction de prix")
plt.title("Prix reels vs predictions")
```

```
0.6907546217033811
```



Pour comparer les modèles on rassemble les résultats obtenus dans le tableau suivant :

Modèle	LinearRegression	RandomForestRegressor	XGBRegressor
R^2	0.6357622384656229	0.6350888386777787	0.6907546217033811

Généralement, les trois modèles ont des performances relativement similaires, avec des scores R^2 assez proches les uns des autres. Cependant, XGBRegressor semble avoir la meilleure performance parmi ces trois modèles en termes de R^2, bien que la différence soit minime.

Cross validation :

On a continué à ajuster les hyperparamètres de nos modèles, en particulier pour des modèles tels que Random Forest et XGBoost. En utilisant la recherche par grille (GridSearchCV) pour rechercher les meilleures combinaisons d'hyperparamètres, et pour améliorer le score R^2

```
# Définir les hyperparamètres à optimiser pour chaque modèle
param_grid_rf = {
    'n_estimators': [50, 100, 200, 300, 500],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 8]
}

param_grid_xgb = {
    'n_estimators': [50, 100, 200, 300, 500],
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'max_depth': [3, 5, 7, 10],
    'subsample': [0.8, 0.9, 1.0]
}

param_grid_gbr = {
    'n_estimators': [50, 100, 200, 300, 500],
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'max_depth': [3, 5, 7, 10],
    'subsample': [0.8, 0.9, 1.0]
}

# Effectuer la recherche par grille pour chaque modèle
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='r2')
grid_search_xgb = GridSearchCV(xgb, param_grid_xgb, cv=5, scoring='r2')
grid_search_gbr = GridSearchCV(gbr, param_grid_gbr, cv=5, scoring='r2')
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import StackingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Sélectionner les caractéristiques
X = data1[['area', 'bathrooms', 'airconditioning', 'stories', 'parking', 'furnishingstatus', 'bedrooms', 'basement', 'prefarea']]
y = data1['price']

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

# Initialiser les modèles de base
rf = RandomForestRegressor(random_state=1)
xgb = XGBRegressor(random_state=1)
gbr = GradientBoostingRegressor(random_state=1)
```

```python
# Obtenir les meilleurs hyperparamètres
best_params_rf = grid_search_rf.best_params_
best_params_xgb = grid_search_xgb.best_params_
best_params_gbr = grid_search_gbr.best_params_

# Instancier de nouveaux modèles avec les meilleurs hyperparamètres
best_rf = RandomForestRegressor(random_state=1, **best_params_rf)
best_xgb = XGBRegressor(random_state=1, **best_params_xgb)
best_gbr = GradientBoostingRegressor(random_state=1, **best_params_gbr)

# Initialiser le modèle de stacking avec les meilleurs modèles de base
stacking_model = StackingRegressor(
    estimators=[('random_forest', best_rf), ('xgboost', best_xgb), ('gradient_boosting', best_gbr)],
    final_estimator=LinearRegression()
)

# Entraîner le modèle de stacking sur l'ensemble d'entraînement
stacking_model.fit(X_train, y_train)

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

# Évaluer la performance du modèle de stacking
r2_stacking = r2_score(y_test, y_stacking)
print("Coefficient de détermination R^2 avec le modèle de stacking :", r2_stacking)
```

```
Coefficient de détermination R^2 avec le modèle de stacking : 0.646192499881096
```

Après long temps d'éxecution on a obtenu la valeur suivante R^2=0.646192499881096