OpenClassrooms - parcours Data Analyst Business Intelligence - P8

Milestone 1 - Analyse des données

Dans cette partie nous allons analyser le jeu de données que nous avons à notre disposition. Notre objectif est de comprendre les relations entre les variables, et surtout, celles qui influent sur le prix des transactions immobilières. Grâce à ce travail, nous pourrons ne conserver que les variables qui seront utiles à l'apprentissage de notre algorithme.

1.Type des données

La première étape à réaliser est d'importer les données dans le notebook et d'opérer des transformations sur les variables si celles-ci s'imposent (exemple : nettoyage ou changement de type des variables, etc.).

```
#On importe les librairies que nous utiliserons pour traiter les données et les visualiser
import pandas as pd, numpy as np
# N'oubliez pas d'avoir installé les librairies pandas, numpy et matplotlib via la commande pip
```

```
#On importe les données dans un dataframe
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Importation du fichier historique_immobilier_paris_2017_2021_vdef2.xlsx
df_historique = pd.read_excel("/content/drive/MyDrive/Datasets/LPBLP/historique_immobilier_paris_2017_2021_vdef2.xlsx")
#Importation du fichier portefeuille_actifs.xlsx
df_portefeuille = pd.read_excel("/content/drive/MyDrive/Datasets/LPBLP/portefeuille_actifs.xlsx")
```

Attention, si l'opération ci-dessus ne marche pas, il se peut que le package "openpyxl" ne soit pas présent sur votre ordinateur. Si c'est le cas vous pouvez l'installer en éxécutant la commande : pip install openpyxl ou conda install openpyxl.

```
#On vérifie que le type des données attribué par pandas est cohérent
```

```
df_historique.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26196 entries, 0 to 26195
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   date_mutation     26196 non-null  datetime64[ns]
 1   valeur_fonciere   26196 non-null  float64
 2   adresse_numero    26196 non-null  int64
 3   adresse_nom_voie  26196 non-null  object
 4   code_postal       26196 non-null  int64
 5   nom_commune       26196 non-null  object
 6   code_type_local   26196 non-null  int64
 7   type_local        26196 non-null  object
 8   surface_reelle    26196 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(4), object(3)
memory usage: 1.8+ MB
```

```
df_historique.head()
```

	date_mutation	valeur_fonciere	adresse_numero	adresse_nom_voie	code_postal	r
0	2017-01-03	5.505597e+05	8	RUE DES COUTURES SAINT GERVAIS	75003	A
1	2017-01-12	1.576492e+06	32	AV MARCEAU	75008	A
2	2017-01-10	6.577574e+05	52	RUE DU FAUBOURG SAINT HONORE	75008	A
3	2017-01-10	2.500868e+05	64	RUE DU VERTBOIS	75003	A
				RUE DES		

```
df_portefeuille.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 275 entries, 0 to 274
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   adresse_numero            275 non-null    int64
 1   adresse_nom_voie          275 non-null    object
 2   code_postal               275 non-null    int64
 3   code_commune              275 non-null    int64
 4   nom_commune               275 non-null    object
 5   surface_carrez            275 non-null    float64
 6   code_type_local           275 non-null    int64
 7   type_local                275 non-null    object
 8   surface_reelle_bati       275 non-null    int64
 9   nombre_pieces_principales 275 non-null    int64
 10  longitude                 275 non-null    float64
 11  latitude                  275 non-null    float64
dtypes: float64(3), int64(6), object(3)
memory usage: 25.9+ KB
```

```
#PROFILING
####
# Installation de ydata-profiling

!pip install ydata-profiling
```

```
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.10/
Collecting multimethod<2,>=1.4 (from ydata-profiling)
  Downloading multimethod-1.12-py3-none-any.whl.metadata (9.6 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.10
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.10/dist-p
Collecting imagehash==4.3.1 (from ydata-profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: wordcloud>=1.9.1 in /usr/local/lib/python3.10/dist-
Collecting dacite>=1.8 (from ydata-profiling)
  Downloading dacite-1.8.1-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: numba<1,>=0.56.0 in /usr/local/lib/python3.10/dist-
Collecting PyWavelets (from imagehash==4.3.1->ydata-profiling)
  Downloading pywavelets-1.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/d
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python
```

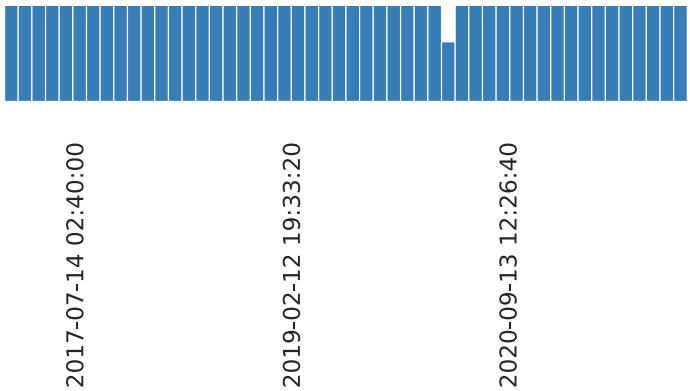
```
Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 13.5 MB/s eta 0:00:00
Downloading dacite-1.8.1-py3-none-any.whl (14 kB)
Downloading multimethod-1.12-py3-none-any.whl (10 kB)
Downloading phik-0.12.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 686.1/686.1 kB 15.0 MB/s eta 0:00:00
Downloading visions-0.7.6-py3-none-any.whl (104 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.8/104.8 kB 7.3 MB/s eta 0:00:00
Downloading pywavelets-1.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_6
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 47.0 MB/s eta 0:00:00
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) ... done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=27081 s
  Stored in directory: /root/.cache/pip/wheels/dd/91/29/a79cecb328d01739e64017b6fb
Successfully built htmlmin
Installing collected packages: htmlmin, PyWavelets, multimethod, dacite, imagehash
Successfully installed PyWavelets-1.7.0 dacite-1.8.1 htmlmin-0.1.12 imagehash-4.3.
```

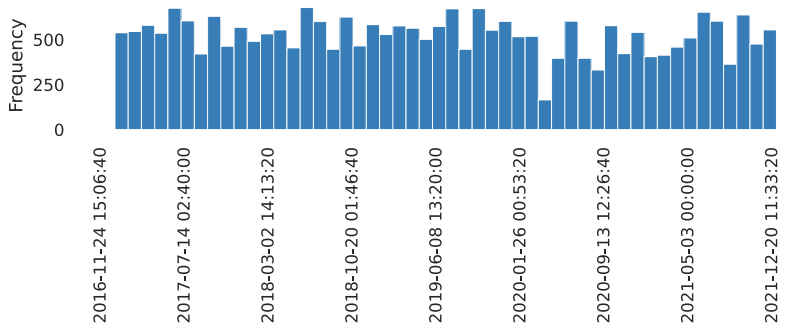
```
# Import de Profile Report
from ydata_profiling import ProfileReport


# Production du rapport de Profiling du df_portefeuille
# profile = ProfileReport(df_portefeuille)
# profile.to_notebook_iframe() # commenter / désactiver pour accélérer le rechargement


# Production du rapport de Profiling du df_historique
profile = ProfileReport(df_historique)
profile.to_notebook_iframe() # commenter / désactiver pour accélérer le rechargement d
```

Summarize dataset: 100% 34/34 [00:16<00:00, 1.70it/s, Completed]

Generate report structure: 100% 1/1 [00:18<00:00, 18.17s/it]

Render HTML: 100% 1/1 [00:02<00:00, 2.57s/it]





Histogram with fixed size bins (bins=50)

valeur_fonciere

Real number (ℝ)

HIGH CORRELATION (This variable has a high overall correlation with 1 fields:

D'après le profiling ci-dessus, il ressort que le DataFrame comporte 16 doublons. Nous allons donc le nettoyer pour les écarter.

```
# Suppression des doublons dans le DataFrame df_historique
# En supprimant les doublons, seules les lignes uniques sont conservées
df_historique = df_historique.drop_duplicates()
```

2. Regardons les différents types de biens immobiliers que nous avons dans nos données :

```
print('la liste des différents types de biens immobiliers que nous avons dans les donr
```

```
la liste des différents types de biens immobiliers que nous avons dans les données
```

```
# Obtenir les types de biens immobiliers uniques dans chaque DataFrame
types_historique = df_historique['type_local'].unique()
types_portefeuille = df_portefeuille['type_local'].unique()


# Conversion des types de biens en chaînes de caractères pour affichage
types_historique_str = ", ".join(types_historique)
types_portefeuille_str = ", ".join(types_portefeuille)

# Affichage des résultats avec print
print("La liste des différents types de biens immobiliers que nous avons dans les donr
print(f"Types dans df_historique : {types_historique_str}")
print(f"Types dans df_portefeuille : {types_portefeuille_str}")
```

```
La liste des différents types de biens immobiliers que nous avons dans les données
Types dans df_historique : Appartement, Local industriel. commercial ou assimilé
Types dans df_portefeuille : Appartement, Local industriel. commercial ou assimilé
```

On observe qu'il y a ? types de biens, et que leurs codes types sont ?.

```
# Extraire les types de biens et leurs codes depuis le DataFrame historique
types_historique_info = df_historique[['type_local', 'code_type_local']].drop_duplicat

# Extraire les types de biens et leurs codes depuis le DataFrame portefeuille
types_portefeuille_info = df_portefeuille[['type_local', 'code_type_local']].drop_dupl

# Afficher les résultats
print("Dans les données historiques :")
print(f"Nombre de types de biens : {len(types_historique_info)}")
print(types_historique_info.to_string(index=False))

print("\nDans le portefeuille d'actifs :")
print(f"Nombre de types de biens : {len(types_portefeuille_info)}")
print(types_portefeuille_info.to_string(index=False))
```

```
Dans les données historiques :
Nombre de types de biens : 2
                            type_local  code_type_local
                           Appartement               2
Local industriel. commercial ou assimilé               4

Dans le portefeuille d'actifs :
Nombre de types de biens : 2
                            type_local  code_type_local
                           Appartement               2
Local industriel. commercial ou assimilé               4
```

3. Nombre de transactions

```
# Calculer le nombre de transactions dans le DataFrame historique
nombre_transactions_historique = len(df_historique)
# Afficher le résultat
print(f"Le nombre de transactions dans les données historiques est : {nombre_transacti
```

```
Le nombre de transactions dans les données historiques est : 26180
```

```
# Filtrer les transactions pour les appartements dans le DataFrame historique
transactions_appartements_historique = df_historique[df_historique['type_local'] == 'A
# Calculer le nombre de transactions pour les appartements
nombre_transactions_appartements_historique = len(transactions_appartements_historique
# Afficher le résultat
print(f"Le nombre de transactions pour les appartements dans les données historiques e
```

```
Le nombre de transactions pour les appartements dans les données historiques est :
```

```
# Filtrer les transactions pour les locaux commerciaux dans le DataFrame historique
transactions_locaux_commerciaux_historique = df_historique[df_historique['type_local']
# Calculer le nombre de transactions pour les locaux commerciaux
nombre_transactions_locaux_commerciaux_historique = len(transactions_locaux_commerciau
# Afficher le résultat
print(f"Le nombre de transactions pour les locaux commerciaux dans les données histori
```

```
Le nombre de transactions pour les locaux commerciaux dans les données historiques
```

4. interval de l'historique des données

```
# Trouver la première et la dernière date de transaction dans le DataFrame historique
premiere_date_historique = df_historique['date_mutation'].min()
derniere_date_historique = df_historique['date_mutation'].max()

# Afficher le résultat
print(f"La première date de transaction dans les données est le : {premiere_date_histo
print(f"La dernière date de transaction dans les données est le : {derniere_date_histo
```

```
La première date de transaction dans les données est le : 2017-01-02 00:00:00
La dernière date de transaction dans les données est le : 2021-12-31 00:00:00
```

Commençons à analyser les données

Maintenant nous allons analyser les données historiques pour les 2 différents types de biens immobiliers en essayant d'identifier les relations entre les variables. Mais avant cela, il nous faudra pouvoir comparer les biens entre eux, et pour cela nous allons créer une colonne du prix au metre carré.

```
# Calcul de la colonne


# Vérifier et convertir la colonne 'surface_reelle' en numérique si nécessaire
df_historique.loc[:, 'surface_reelle'] = pd.to_numeric(df_historique['surface_reelle']

# Calculer le prix au mètre carré
df_historique.loc[:, 'prix_metre_carre'] = df_historique['valeur_fonciere'] / df_histo

# Afficher les premières lignes du DataFrame pour vérifier
print(df_historique[['valeur_fonciere', 'surface_reelle', 'prix_metre_carre']].head())
```

```
   valeur_fonciere  surface_reelle  prix_metre_carre
0     5.505597e+05              50      11011.193562
1     1.576492e+06             163       9671.732877
2     6.577574e+05              66       9966.020548
3     2.500868e+05              22      11367.582877
4     1.762667e+05              15      11751.113836
```

Maintenant, analysons les appartements.

5. Evolution du prix au metre carré des appartements dans Paris

```
#On retire les colonnes qui sont à priori inutiles ET nous allons nous concentrer sur



# Filtrer les transactions pour les appartements
appartements_historique = df_historique[df_historique['type_local'] == 'Appartement']

# Sélectionner les colonnes utiles
colonnes_utiles = ['date_mutation', 'valeur_fonciere', 'surface_reelle', 'prix_metre_c
appartements_historique = appartements_historique[colonnes_utiles]
appartements_historique.head()
```

	date_mutation	valeur_fonciere	surface_reelle	prix_metre_carre
0	2017-01-03	5.505597e+05	50	11011.193562
1	2017-01-12	1.576492e+06	163	9671.732877
2	2017-01-10	6.577574e+05	66	9966.020548
3	2017-01-10	2.500868e+05	22	11367.582877
4	2017-01-13	1.762667e+05	15	11751.113836

```
#Préparons un dataframe en regroupant les prix moyens de ventes des appartements par a



# S'assurer que 'date_mutation' est de type datetime
appartements_historique['date_mutation'] = pd.to_datetime(appartements_historique['dat

# Ajouter une colonne 'année' pour l'analyse par année
appartements_historique['annee'] = appartements_historique['date_mutation'].dt.year

# Calculer le prix moyen au mètre carré et la surface moyenne par année
stats_annee = appartements_historique.groupby('annee').agg({
    'prix_metre_carre': 'mean',
    'surface_reelle': 'mean'
}).reset_index()

# Renommer les colonnes pour plus de clarté
stats_annee.rename(columns={
    'prix_metre_carre': 'prix_moyen_metre_carre',
    'surface_reelle': 'surface_moyenne'
}, inplace=True)

# Arrondir les valeurs à deux décimales
stats_annee['prix_moyen_metre_carre'] = stats_annee['prix_moyen_metre_carre'].round(2)
stats_annee['surface_moyenne'] = stats_annee['surface_moyenne'].round(2)

# Afficher le DataFrame résultant
print(stats_annee)
```

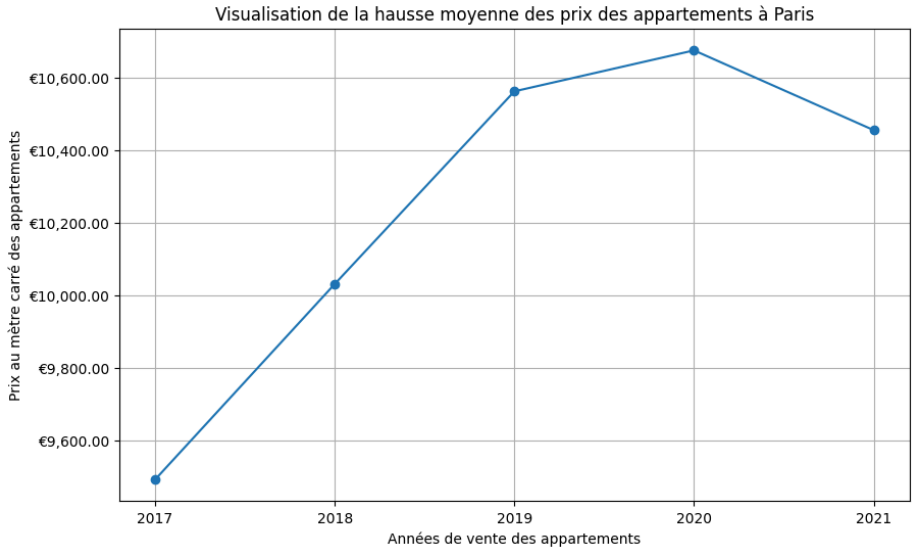
```
   annee  prix_moyen_metre_carre  surface_moyenne
0   2017                 9492.44            44.63
1   2018                10031.40            44.27
2   2019                10562.46            43.37
3   2020                10675.34            42.91
4   2021                10455.60            43.48
```

On constate sur la moyenne des prix de vente des appartements à Paris que ?

On constate que le prix moyen au m2 à Paris a tendance à augmenter avec les année.

Nous allons créer un graphique pour mieux le visualiser.

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# S'assurer que les données sont prêtes
# (stats_annee contient 'annee', 'prix_moyen_metre_carre', 'surface_moyenne')

plt.figure(figsize=(10, 6))  # Taille du graphique

# Tracer le graphique en utilisant les données de stats_annee
plt.plot(stats_annee['annee'], stats_annee['prix_moyen_metre_carre'], marker='o', line

# Configuration des étiquettes des axes
plt.xlabel('Années de vente des appartements')
plt.ylabel('Prix au mètre carré des appartements')

# Formater l'axe des ordonnées pour afficher l'euro
formatter = ticker.StrMethodFormatter('€{x:,.2f}')
plt.gca().yaxis.set_major_formatter(formatter)

# Configurer les ticks de l'axe des x pour montrer chaque année
plt.xticks(stats_annee['annee'])

# Ajouter une grille et un titre
plt.grid(True)
plt.title('Visualisation de la hausse moyenne des prix des appartements à Paris')

# Afficher le graphique
plt.show()
```



6. Différences de prix au metre carré entre les arrondissements

Maintenant nous allons chercher à afficher l'évolution des prix par arrondissement. Vérifions la liste des arrondissements que nous avons dans nos données. Normalement à Paris nous avons 20 codes postaux différents, 1 par arrondissement allant de 75001 à 75020.

```
#Liste des codes postaux dans nos données.


# Filtrer les transactions pour les appartements
appartements_historique = df_historique[df_historique['type_local'] == 'Appartement']

# Extraire les arrondissements uniques
arrondissements_uniques = appartements_historique['code_postal'].unique()

# Trier les arrondissements pour une meilleure lisibilité
arrondissements_uniques = sorted(arrondissements_uniques)

# Afficher les arrondissements uniques
print(f"Les arrondissements présents dans les données sont : {arrondissements_uniques}
```

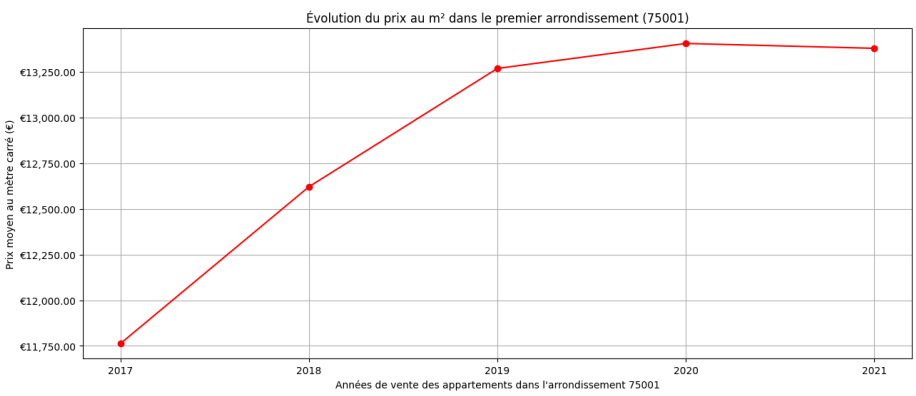
```
Les arrondissements présents dans les données sont : [75001, 75002, 75003, 75004,
```

```
#Créons un dataframe pour une deuxième visualisation avec l'evolution des prix par arr


# Filtrer les transactions pour les appartements
appartements_historique = df_historique[df_historique['type_local'] == 'Appartement'].

# S'assurer que 'date_mutation' est de type datetime
appartements_historique['date_mutation'] = pd.to_datetime(appartements_historique['dat

# Ajouter une colonne 'année' pour l'analyse par année
appartements_historique['annee'] = appartements_historique['date_mutation'].dt.year

# Vérifier et filtrer uniquement les arrondissements de Paris (75001 à 75020)
appartements_historique_paris = appartements_historique[
    appartements_historique['code_postal'].between(75001, 75020)
]

# Calculer le prix moyen au mètre carré par année et par arrondissement
prix_par_arrondissement = appartements_historique_paris.groupby(['annee', 'code_postal
    'prix_metre_carre': 'mean'
}).reset_index()

# Arrondir les valeurs à deux décimales
prix_par_arrondissement['prix_metre_carre'] = prix_par_arrondissement['prix_metre_carr
```

```
# Afficher le DataFrame résultant
print(prix_par_arrondissement.head())
```

```
   annee  code_postal  prix_metre_carre
0   2017        75001          11762.71
1   2017        75002          10920.13
2   2017        75003          11679.82
3   2017        75004          12407.23
4   2017        75005          11306.20
```

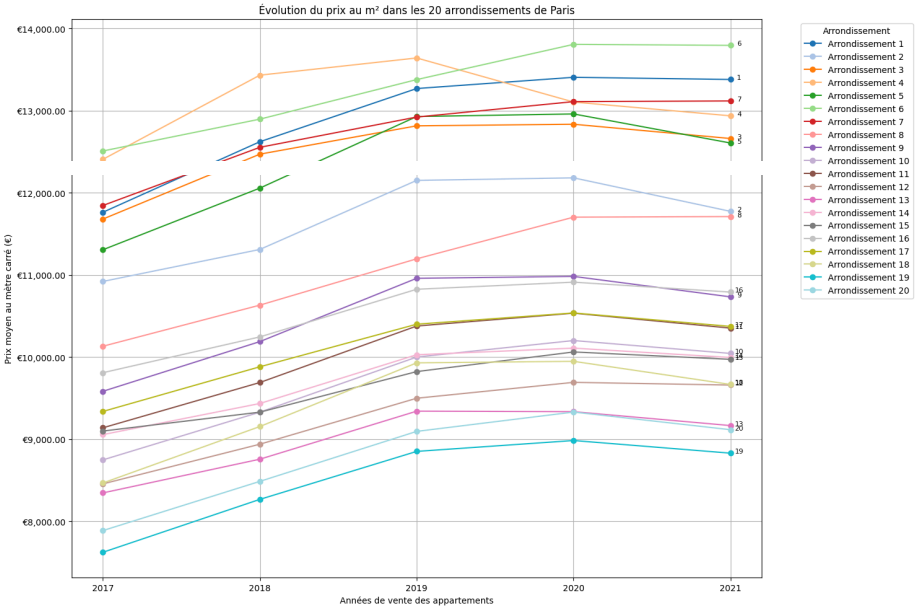
```
#Création d'un graphique pour visualiser la hausse de la moyenne des prix au metre car



# Filtrer les données pour l'arrondissement 75001
arrondissement_75001 = prix_par_arrondissement[prix_par_arrondissement['code_postal']

# Création du graphique
plt.figure(figsize=(15, 6))  # Taille du graphique

# Tracer le graphique en utilisant les données de l'arrondissement 75001
plt.plot(arrondissement_75001['annee'], arrondissement_75001['prix_metre_carre'], colo

# Configuration des étiquettes des axes
plt.xlabel('Années de vente des appartements dans l\'arrondissement 75001')
plt.ylabel('Prix moyen au mètre carré (€)')

# Formater l'axe des ordonnées pour afficher l'euro
formatter = ticker.StrMethodFormatter('€{x:,.2f}')
plt.gca().yaxis.set_major_formatter(formatter)

# Configurer les ticks de l'axe des x pour montrer chaque année
plt.xticks(stats_annee['annee'])

# Ajouter une grille et un titre
plt.grid(True)
plt.title('Évolution du prix au m² dans le premier arrondissement (75001)')

# Afficher le graphique
plt.show()
```



Nous allons ajouter à cette visualisation tous les autres arrondissement avec une couleur différente.

```
import seaborn as sns

# Créer une palette de couleurs plus distinctes
palette = sns.color_palette("tab20", len(arrondissements_uniques))

# Création du graphique
plt.figure(figsize=(15, 10))  # Taille du graphique
```

```
# Tracer le graphique pour chaque arrondissement
for idx, code_postal in enumerate(arrondissements_uniques):
    data = prix_par_arrondissement[prix_par_arrondissement['code_postal'] == code_post
    plt.plot(data['annee'], data['prix_metre_carre'], marker='o', linestyle='-', color
             label=f"Arrondissement {code_postal - 75000}")

    # Ajouter une étiquette à la fin de chaque ligne
    plt.annotate(f"{code_postal - 75000}",
                 (data['annee'].values[-1], data['prix_metre_carre'].values[-1]),
                 textcoords="offset points", xytext=(10, 0), ha='center', fontsize=8)

# Configuration des étiquettes des axes
plt.xlabel('Années de vente des appartements')
plt.ylabel('Prix moyen au mètre carré (€)')

# Formater l'axe des ordonnées pour afficher l'euro
formatter = ticker.StrMethodFormatter('€{x:,.2f}')
plt.gca().yaxis.set_major_formatter(formatter)

# Configurer les ticks de l'axe des x pour montrer chaque année
plt.xticks(stats_annee['annee'])

# Ajouter une grille, un titre et une légende
plt.grid(True)
plt.title('Évolution du prix au m² dans les 20 arrondissements de Paris')
plt.legend(loc='upper left', bbox_to_anchor=(1.05, 1), title="Arrondissement")

# Ajuster l'affichage pour éviter que la légende soit coupée
plt.tight_layout()

# Afficher le graphique
plt.show()
```



On observe que globalement que ?

Dans l'ensemble des arrondissements les prix au m2 onf fortement augmentés jusqu'en 2019 avant d'amorcer un ralentissement jusqu'en 2020 puis une stagnation voir une baisse des prix en 2021.

7. Prix au metre carré par transaction dans un arrondissement

```
#Vérifions le nombre de transaction dans le 6ème car le prix semble élevé


# Filtrer les transactions pour le 6ème arrondissement (code postal 75006)
transactions_6eme = df_historique[
    (df_historique['type_local'] == 'Appartement') &
    (df_historique['code_postal'] == 75006)
]

# Compter le nombre de transactions
nombre_transactions_6eme = transactions_6eme.shape[0]

# Afficher le résultat
print(f"Le nombre de transactions pour les appartements dans le 6ème arrondissement es
```

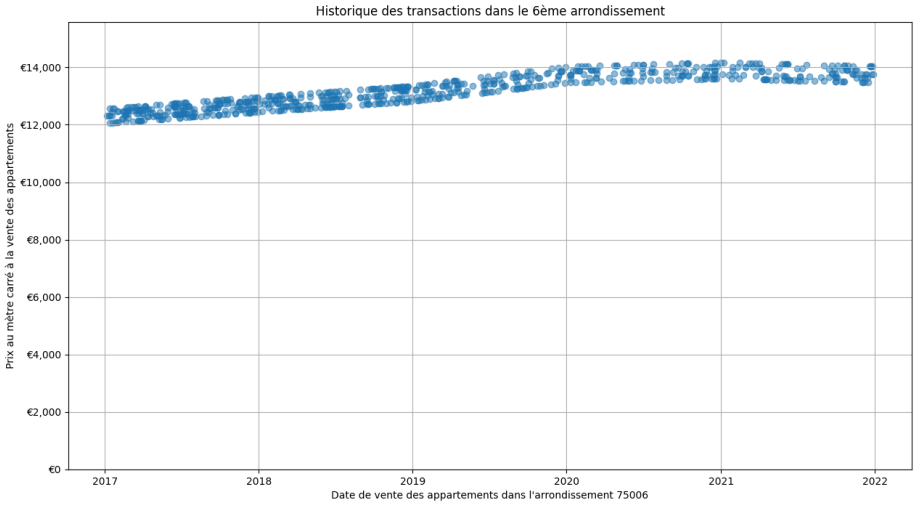
```
Le nombre de transactions pour les appartements dans le 6ème arrondissement est :
```

On constate que ?

On constate qu'il y a un nombre significatif de transactions.

Affichons l'historique des transactions pour visualiser la dispersion des données :

```
# Filtrer les transactions pour le 6ème arrondissement (code postal 75006)
transactions_6eme = df_historique[
    (df_historique['type_local'] == 'Appartement') &
    (df_historique['code_postal'] == 75006)
].copy()
# S'assurer que 'date_mutation' est de type datetime
transactions_6eme['date_mutation'] = pd.to_datetime(transactions_6eme['date_mutation']

# Créer le scatter plot
plt.figure(figsize=(15, 8))
plt.scatter(transactions_6eme['date_mutation'], transactions_6eme['prix_metre_carre'],

# Configuration des étiquettes des axes
plt.xlabel('Date de vente des appartements dans l\'arrondissement 75006')
plt.ylabel('Prix au mètre carré à la vente des appartements')

# Formater l'axe des ordonnées pour afficher l'euro
formatter = ticker.StrMethodFormatter('€{x:,.0f}')
plt.gca().yaxis.set_major_formatter(formatter)

# Ajuster la limite de l'axe des ordonnées si nécessaire
plt.ylim(0, transactions_6eme['prix_metre_carre'].max() * 1.1)

# Ajouter une grille et un titre
plt.grid(which='both')
plt.title('Historique des transactions dans le 6ème arrondissement')

# Afficher le graphique
plt.show()
```



On observe que ?

On constate que le prix au m2 des transactions dans le sixième évolue dans une fourchete de prix constante au fil du temps et qu'il y a une plus grande densité sur les années 2017 et 2018. Il y a donc vraisemblablement un volume de ventes supérieur sur ce 2 années.

Pour nous rassurer sur la relation entre les données nous allons utiliser un test statistique de corrélation.

8. Vérification de la relation entre le prix au m² et la date dans le 6ème arrondissement par le calcul du coefficient de corrélation de Pearson

Pour cette distribution nous allons calculer le coefficient de corrélation de Pearson. La relation entre le prix au metre carré et la date n'est pas forcément linéaire mais elle s'en rapproche suffisamment pour que cette analyse soit pertinente (voir l'évolution du prix au metre carré globale montrée plus haut, on a presque une droite sauf entre 2020 et 2021). Ce qui nous interesse c'est de prouver que le temps a une influence sur le prix.

```
#On calcule la corrélation de Spearman
from scipy import stats

# Créer une copie explicite du DataFrame filtré
transactions_6eme = df_historique[
    (df_historique['type_local'] == 'Appartement') &
    (df_historique['code_postal'] == 75006)
].copy()

# S'assurer que 'date_mutation' est de type datetime
transactions_6eme['date_mutation'] = pd.to_datetime(transactions_6eme['date_mutation']

# Convertir les dates en valeurs numériques (timestamps)
transactions_6eme['timestamp'] = transactions_6eme['date_mutation'].apply(lambda x: x.

# Calculer le coefficient de corrélation de Pearson
correlation, p_value = stats.pearsonr(transactions_6eme['timestamp'], transactions_6em

# Afficher les résultats
```

```
print(f"Le coefficient de corrélation de Pearson est : {correlation:.4f}")
print(f"La valeur p associée est : {p_value:.4e}")

# Interprétation
if p_value < 0.05:
    print("La corrélation est statistiquement significative (p < 0.05).")
else:
    print("La corrélation n'est pas statistiquement significative (p >= 0.05).")
```

```
Le coefficient de corrélation de Pearson est : 0.9038
La valeur p associée est : 7.1084e-263
La corrélation est statistiquement significative (p < 0.05).
```

Le coefficient de corrélation est de ? avec une pvalue de ? donc nous pouvons confirmer la corrélation.

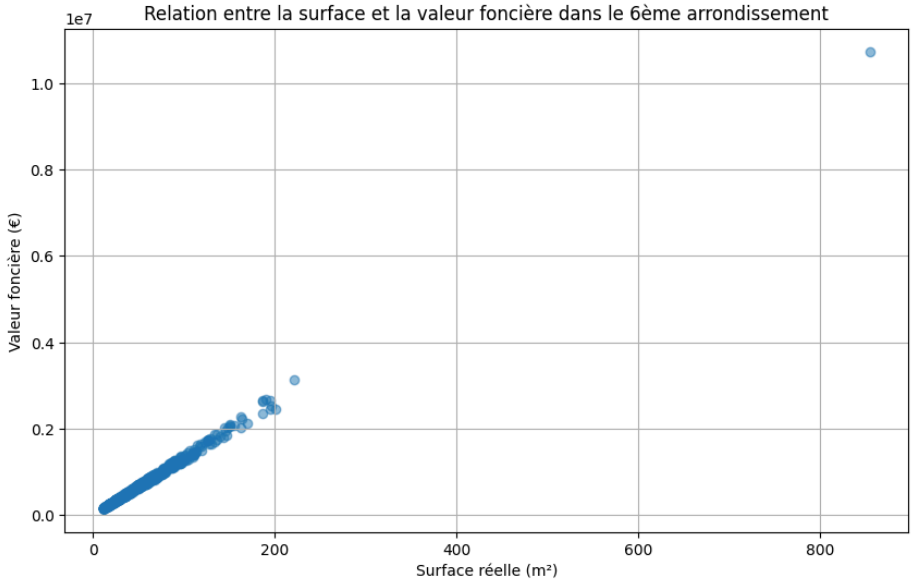
9. Vérification de la relation entre la valeur fonciere et la surface

```
# Créer une copie explicite du DataFrame filtré pour le 6ème arrondissement
transactions_6eme = df_historique[
    (df_historique['type_local'] == 'Appartement') &
    (df_historique['code_postal'] == 75006)
].copy()

# Vérifier les colonnes pertinentes
print("Colonnes disponibles : ", transactions_6eme.columns)

# S'assurer que les colonnes 'valeur_fonciere' et 'surface_reelle_bati' sont au bon fo
transactions_6eme['valeur_fonciere'] = pd.to_numeric(transactions_6eme['valeur_foncier
transactions_6eme['surface_reelle'] = pd.to_numeric(transactions_6eme['surface_reelle'

# Supprimer les lignes avec des valeurs NaN qui pourraient causer des erreurs dans le
transactions_6eme.dropna(subset=['valeur_fonciere', 'surface_reelle'], inplace=True)

# Calculer le coefficient de corrélation de Pearson
correlation, p_value = stats.pearsonr(transactions_6eme['valeur_fonciere'], transactio

# Afficher les résultats
print(f"Le coefficient de corrélation de Pearson entre la valeur foncière et la surfac
print(f"La valeur p associée est : {p_value:.4e}")

# Interprétation
if p_value < 0.05:
    print("La corrélation est statistiquement significative (p < 0.05).")
else:
    print("La corrélation n'est pas statistiquement significative (p >= 0.05).")
```

```
Colonnes disponibles :  Index(['date_mutation', 'valeur_fonciere', 'adresse_numero
       'adresse_nom_voie', 'code_postal', 'nom_commune', 'code_type_local',
       'type_local', 'surface_reelle', 'prix_metre_carre'],
      dtype='object')
Le coefficient de corrélation de Pearson entre la valeur foncière et la surface es
La valeur p associée est : 0.0000e+00
La corrélation est statistiquement significative (p < 0.05).
```

Le coefficient de corrélation est de ? avec une pvalue de ?. La relation est donc confirmée.

```
# Confirmation avec un graphique de dispersion (ajout Alex)
# Création du graphique
plt.figure(figsize=(10, 6))
plt.scatter(transactions_6eme['surface_reelle'], transactions_6eme['valeur_fonciere'],
plt.xlabel('Surface réelle (m²)')
plt.ylabel('Valeur foncière (€)')
plt.title('Relation entre la surface et la valeur foncière dans le 6ème arrondissement
plt.grid(True)
plt.show()
```



Regardons maintenant si le type de bien à une influence sur le prix au metre carré également.

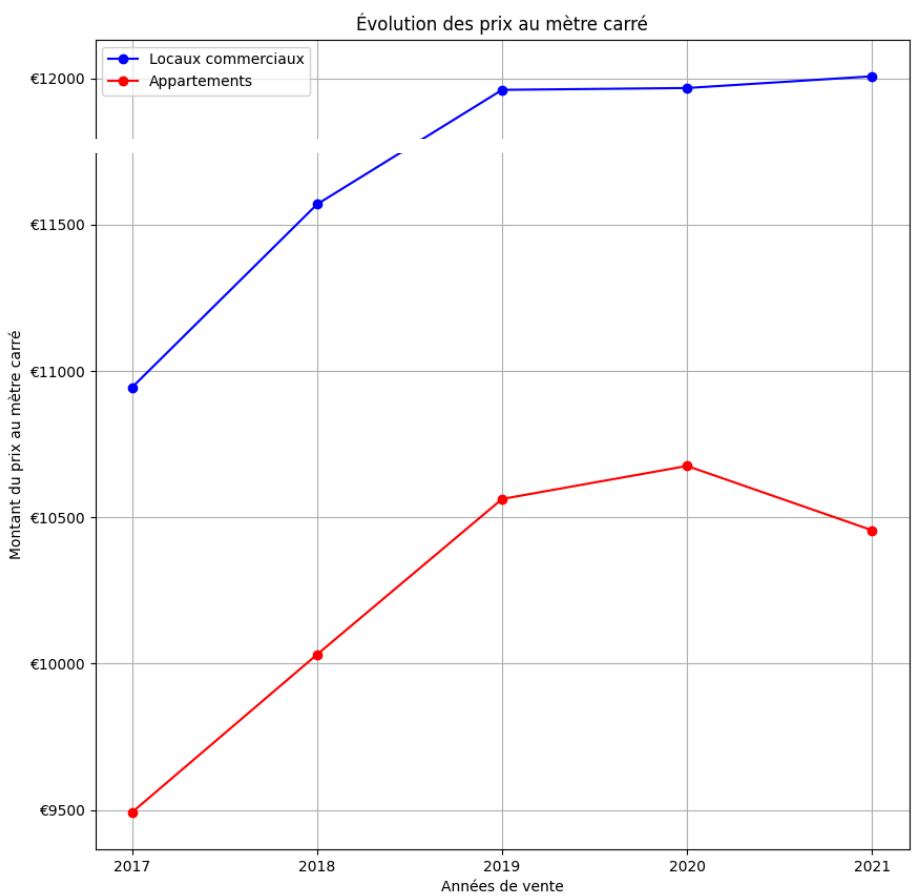
10. Analyse des locaux industriels, commerciaux et assimilés

```
#On crée un dataset qui ne contient que les locaux commerciaux
# Filtrer pour ne garder que les locaux commerciaux, industriels et assimilés
locaux_commerciaux = df_historique[df_historique['type_local'] == 'Local industriel. c


#Préparons un dataframe en regroupant les prix au metre carré moyens des ventes par ar



# S'assurer que 'date_mutation' est de type datetime
locaux_commerciaux['date_mutation'] = pd.to_datetime(locaux_commerciaux['date_mutatior

# Calculer le prix au mètre carré pour les locaux commerciaux
locaux_commerciaux['prix_metre_carre'] = locaux_commerciaux['valeur_fonciere'] / locau

# Extraire l'année de la date de mutation
locaux_commerciaux['annee'] = locaux_commerciaux['date_mutation'].dt.year

# Regrouper par année et calculer la moyenne des prix au mètre carré
prix_moyen_par_annee = locaux_commerciaux.groupby('annee')['prix_metre_carre'].mean().

# Arrondir les prix au mètre carré moyens à 2 chiffres après la virgule
prix_moyen_par_annee['prix_metre_carre'] = prix_moyen_par_annee['prix_metre_carre'].rc

# Afficher le DataFrame des prix moyens par année
print("Prix moyen au m² par année pour les locaux commerciaux :")
print(prix_moyen_par_annee)
```

```
Prix moyen au m² par année pour les locaux commerciaux :
   annee  prix_metre_carre
0   2017          10944.06
1   2018          11569.50
2   2019          11960.13
3   2020          11966.47
4   2021          12006.49
```

```
#Création d'un graphique pour visualiser la hausse de la moyenne des prix



# Filtrer pour ne garder que les appartements
appartements = df_historique[df_historique['type_local'] == 'Appartement'].copy()

# Calculer le prix au mètre carré pour les appartements
appartements['prix_metre_carre'] = appartements['valeur_fonciere'] / appartements['sur
```

```
# Extraire l'année de la date de mutation pour les appartements
appartements['annee'] = appartements['date_mutation'].dt.year

# Regrouper par année et calculer la moyenne des prix au mètre carré pour les apparten
prix_moyen_appartements = appartements.groupby('annee')['prix_metre_carre'].mean().res
prix_moyen_appartements['prix_metre_carre'] = prix_moyen_appartements['prix_metre_carr

# S'assurer que les données des deux types de biens couvrent la même période
annees = ['2017', '2018', '2019', '2020', '2021']

# Extraire les prix moyens pour ces années pour chaque type de bien
prix_locaux_commerciaux = prix_moyen_par_annee[prix_moyen_par_annee['annee'].isin(map(
prix_appartements = prix_moyen_appartements[prix_moyen_appartements['annee'].isin(map(

# Création du graphique
plt.figure(figsize=(10, 10))
plt.plot(annees, prix_locaux_commerciaux, label='Locaux commerciaux', color='blue', ma
plt.plot(annees, prix_appartements, label='Appartements', color='red', marker='o')

plt.xlabel('Années de vente')
plt.ylabel('Montant du prix au mètre carré')
plt.legend()
formatter = ticker.FormatStrFormatter('€%d')
plt.gca().yaxis.set_major_formatter(formatter)
plt.grid()
plt.title('Évolution des prix au mètre carré')
plt.xticks(annees)  # Assure que les années soient correctement placées
plt.show()
```



Le prix au metre carré des locaux commerciaux est ?

Le prix au mètre carré des locaux commerciaux est supérieur à celui des appartements et suit la même courbe d'évolution des prix au fil du temps excepté en 2020 et 2021 ou la courbe d'évolution des prix chute plus rapidement pour les appartements que pour les locaux commerciaux.

Après ces analyses, nous pouvons conclure que les dimensions à utiliser pour prédire le prix au m² sont :

- la surface du bien immobilier,
- la date considérée,
- la localisation (code_postal),
- le type de bien.

Milestone 2 - Entraînement de l'algorithme

Dans cette section nous allons maintenant entrâiner un algorithme à prédire la valeur fonciere d'un bien immobilier. Pour cela nous allons utiliser l'algorithme de régression linéaire.

On commence par préparer nos données en transformant les colonnes catégoriques du code postal et du type de local grâce au one hot encoder (sklearn) / get_dummies (pandas)

```
# Importation de OneHotEncoder
from sklearn.preprocessing import OneHotEncoder
```

```
# Extraire les composantes temporelles de la date_mutation
df_historique.loc[:, 'année'] = df_historique['date_mutation'].dt.year
df_historique.loc[:, 'mois'] = df_historique['date_mutation'].dt.month # Le mois est c
df_historique.loc[:, 'timestamp'] = df_historique['date_mutation'].apply(lambda x: x.t
```

```
# Filtrage des colonnes d'intérêt
colonnes_interet = ['code_postal', 'type_local', 'surface_reelle', 'année', 'mois', 'v
df_model = df_historique[colonnes_interet].dropna()
```

```
# Encodage one-hot des variables catégorielles (code_postal et type_local)
df_encoded = pd.get_dummies(df_model, columns=['code_postal', 'type_local'])
```

On utilise le train_test_split pour prélever un tiers de nos données (33%) et les garder de côté. Nous allons entraîner notre algorithme sur le reste des données et puis mesurer notre erreur moyenne en pourcentage de la valeur foncière

```
# On importe le train test split de la librairie sk_learn
from sklearn.model_selection import train_test_split
# La valeur y à trouver est la valeur foncière
```

```
# Séparer les caractéristiques (X) et la cible (y)
X = df_encoded.drop('valeur_fonciere', axis=1)
y = df_encoded['valeur_fonciere']

# 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=0.33, random_state

# Vérifier la taille des ensembles
print("Taille de l'ensemble d'entraînement :", X_train.shape)
print("Taille de l'ensemble de test :", X_test.shape)

# Afficher un aperçu des données d'entraînement
# print("\nAperçu des données d'entraînement :")
# print(X_train.head())
```

```
Taille de l'ensemble d'entraînement : (17540, 25)
Taille de l'ensemble de test : (8640, 25)
```

```
#conversion des données en str
```

```
from sklearn.linear_model import LinearRegression
#On entraîne l'algorithme ci-dessous et on effectue la prédiction
```

```
# Initialiser et entraîner le modèle de régression linéaire
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# Prédire les valeurs foncières sur l'ensemble de test
y_pred = model.predict(X_test)


# Évaluer la performance du modèle
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
mape = mean_absolute_percentage_error(y_test, y_pred)

print("\nErreur quadratique moyenne (MSE) :", mse)
print("Erreur quadratique moyenne racine (RMSE) :", rmse)
print("Erreur absolue moyenne en pourcentage (MAPE) :", round(mape*100, 2) , "%" )
```

```
Erreur quadratique moyenne (MSE) : 4269297854.984127
Erreur quadratique moyenne racine (RMSE) : 65339.864210022104
Erreur absolue moyenne en pourcentage (MAPE) : 9.06 %
```

Notre algorithme fait donc 9 % d'erreur en moyenne sur la prédiction de la valeur foncière.

Mes conclusions sur ce résultat et comment j'aurais pu aller plus loin :
Un MAPE de 9% indique une prédiction fiable et conforme aux attentes du client. Pour aller plus loin, nous aurions par exemple pu tenter l'intégration de la variable "adresse_nom_voie" dans le modèle. Cependant, cette option apporterait de nombreux points de vigilances (granularité de l'information, beaucoup de valeurs uniques, difficultés d'encodage).

Milestone 3 - Prédiction définitive pour le client

Nous avons récupéré le fichier avec le portefeuille des actifs de la société. Nous allons l'importer puis effectuer la prédiction et statuer sur la branche qui, selon notre prédiction, aura le plus de valeur à la date demandée c'est à dire au 31 décembre 2022.
Petite précision, nous souhaitons continuer à utiliser la surface réelle pour faire les calculs et pas la surface carrez.

```
#On importe les données dans un dataframe
# df_portefeuille # importé plus haut


# Production du rapport de Profiling du df_portefeuille
profile = ProfileReport(df_portefeuille)
profile.to_notebook_iframe() # commenter / désactiver pour accélérer le rechargement c
```

Summarize dataset: 100% 85/85 [00:19<00:00, 3.15it/s, Completed]

Generate report structure: 100% 1/1 [00:07<00:00, 7.67s/it]

Render HTML: 100% 1/1 [00:03<00:00, 3.25s/it]

Overview

Brought to you by YData (https://ydata.ai/?utm_source=opensource&utm_medium=ydataprofiling&utm_campaign=report)

Dataset statistics

Number of variables	12
Number of observations	275
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	2
Duplicate rows (%)	0.7%
Total size in memory	25.9 KiB
Average record size in memory	96.5 B

Variable types

Numeric	8
Text	1
Categorical	3

Alerts

Dataset has 2 (0.7%) duplicate rows	Duplicates

D'après le rapport ci-dessus, il ressort que nous avons 2 lignes dupliquées. Nous allons donc nettoyer le dataframe pour supprimer les doublons.

```
# Suppression des doublons dans le DataFrame df_portefeuille
# En supprimant les doublons, seules les lignes uniques sont conservées
df_portefeuille = df_portefeuille.drop_duplicates()
```

```
df_portefeuille.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 273 entries, 0 to 274
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   adresse_numero             273 non-null    int64
 1   adresse_nom_voie           273 non-null    object
 2   code_postal                273 non-null    int64
 3   code_commune               273 non-null    int64
 4   nom_commune                273 non-null    object
 5   surface_carrez             273 non-null    float64
 6   code_type_local            273 non-null    int64
 7   type_local                 273 non-null    object
 8   surface_reelle_bati        273 non-null    int64
 9   nombre_pieces_principales  273 non-null    int64
 10  longitude                  273 non-null    float64
 11  latitude                   273 non-null    float64
dtypes: float64(3), int64(6), object(3)
memory usage: 27.7+ KB
```

Nous avons la liste des biens immobiliers de l'entreprise. Pour effectuer une prédiction, nous devons mettre ce fichier au même format que le dataframe que nous avons utilisé lors de l'entraînement de

l'algorithme.

```
# Renommer la colonne 'surface_reelle_bati' en 'surface_reelle' pour correspondre au m
df_portefeuille = df_portefeuille.copy()
df_portefeuille.rename(columns={'surface_reelle_bati': 'surface_reelle'}, inplace=True
```

```
#On réutilise les mêmes fonctions pour faire le one hot encoding des variables catégor
```

```
#On importe la librairie datetime pour pouvoir créer une colonne timestamp avec la dat
import datetime as dt
```

```
# Extraire l'année, le mois (valeurs significatives pour l'estimation)
df_portefeuille.loc[:,'année'] = 2022
df_portefeuille.loc[:,'mois'] = 12
```

```
# Filtrage des colonnes d'intérêt
colonnes_interet = ['code_postal', 'type_local', 'surface_reelle', 'année', 'mois']
df_portefeuille_select = df_portefeuille[colonnes_interet].dropna()
```

```
print(df_portefeuille.columns)
```

```
Index(['adresse_numero', 'adresse_nom_voie', 'code_postal', 'code_commune',
       'nom_commune', 'surface_carrez', 'code_type_local', 'type_local',
       'surface_reelle_bati', 'nombre_pieces_principales', 'longitude',
       'latitude', 'année', 'mois'],
      dtype='object')
```

```
# Encodage one-hot des variables catégorielles (code_postal et type_local)
df_portefeuille_encoded = pd.get_dummies(df_portefeuille_select, columns=['code_postal
```

```
#df_portefeuille_encoded.head()
```

```
#Conversion du dataframe en str
```

Notre dataframe est prêt à être utilisé par notre algorithme de prédiction.

```
# Récupérer les colonnes de surface et les nouvelles colonnes encodées
X_portefeuille = df_portefeuille_encoded
```

```
#On effectue la prédiction
valeurs_predites = model.predict(X_portefeuille)
```

```
# Ajouter les prédictions au DataFrame pour l'analyse
df_portefeuille.loc[:,'valeur_fonciere_predite'] = valeurs_predites.round(2)
```

```
#On vérifie les 10 premières valeurs
df_portefeuille.head(10)
```

Alexandre Duvernay
16 août 2024

D'après l'analyse de prix précédente, le prix au M2 des locaux commerciaux sur Paris est plus élevé que celui des appartements. Le label = 0 serait donc plutôt attribué aux appartements et le label = 1 aux locaux commerciaux.

	adresse_numero	adresse_nom_voie	code_postal	code_commune	nom_commune	surfac
0	127	RUE SAINT-DENIS	75001	75101	Paris 1er Arrondissement	
1	62	RUE NOTRE-DAME DE NAZARETH	75003	75103	Paris 3e Arrondissement	
2	62	RUE DE TURENNE	75003	75103	Paris 3e Arrondissement	
3	32	AV GEORGE V	75008	75108	Paris 8e Arrondissement	
4	152	BD HAUSSMANN	75008	75108	Paris 8e Arrondissement	
5	152	BD HAUSSMANN	75008	75108	Paris 8e Arrondissement	
6	52	RUE DES GRAVILLIERS	75003	75103	Paris 3e Arrondissement	
7	208	RUE SAINT MAUR	75010	75110	Paris 10e Arrondissement	
8	142	RUE DE COURCELLES	75017	75117	Paris 17e Arrondissement	
9	52	RUE LAMARCK	75018	75118	Paris 18e Arrondissement	

Maintenant nous allons comparer la valorisation prédite pour les deux segments.

Commencez à coder ou à générer avec l'IA.

```
#Valorisation du portefeuille sur le segment des particuliers
# Calcul de la somme de la valeur foncière prédite pour les appartements
somme_valeur_fonciere_appartement = df_portefeuille[df_portefeuille['type_local'] == '
# Conversion en millions d'euros
somme_valeur_fonciere_appartement_millions = somme_valeur_fonciere_appartement / 1_000
print('la valorisation du segment particulier est (en millions deuros):', round(somme_
```

```
la valorisation du segment particulier est (en millions deuros): 71 M€
```

```
#Valorisation du portefeuille sur le segment corporate
# Calcul de la somme de la valeur foncière prédite pour les locaux industriels, commer
somme_valeur_fonciere_commercial = df_portefeuille[df_portefeuille['type_local'] == 'L
# Conversion en millions d'euros
somme_valeur_fonciere_commercial_millions = somme_valeur_fonciere_commercial / 1_000_0
print('la valorisation du segment corporate est (en millions deuros):', round(somme_va
```

```
la valorisation du segment corporate est (en millions deuros): 97 M€
```

Mes conclusions sur le segment avec la plus grande valorisation et sur les limites de cette estimation :

Le segment qui aura la plus grande valorisation au 31/12/2022 est le segment corporate avec 97M€ contre 71M€ pour le particulier.

Cette estimation est fondée sur des données historiques. Si les conditions économiques changent, les prédictions peuvent ne pas refléter correctement les nouvelles réalitées. De même, si les données utilisées pour l'entrainement contiennent des erreurs, cela peut affecter la précision des estimations.

```
# Exportation du DataFrame `df_portefeuille` au format CSV
from datetime import datetime # import de la librairie pour l'heure
# Obtenir la date et l'heure actuelles et les formater
timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
# Exporter le DF au format CSV
output_path = f'/content/drive/MyDrive/Datasets/LPBLP/df_portefeuille{timestamp}.csv'
df_portefeuille.to_csv(output_path, index=False)
print(f"DataFrame exporté vers '{output_path}' avec succès.")
```

```
DataFrame exporté vers '/content/drive/MyDrive/Datasets/LPBLP/df_portefeuille20240
```

Milestone 4 - Classification des données issues du jeu de test

Dans cette partie nous allons labelliser automatiquement les biens immobiliers comme étant :

- soit des Appartements
- soit des Local industriel. commercial ou assimilé Pour cela nous allons utiliser l'algorithme du KMeans sur le jeu de données partagé par l'entreprise.

Pour que l'algorithme fonctionne, il faut que nous préparions les données en supprimant les dimensions inutiles et en nous concentrant sur le facteur discriminant entre les appartements et les locaux commerciaux : la différence dans le prix au mètre carré tel que nous l'avons vu avant.

```
#On importe les données dans un dataframe
#Importation du fichier echantillon_a_classer.xlsx
df_echantillon = pd.read_excel("/content/drive/MyDrive/Datasets/LPBLP/echantillon_a_cl
```

On applique les transformations nécessaires. Tout d'abord nous allons calculer le prix au mètre carré en divisant la valeur foncière par la surface. Ensuite nous allons retirer ces colonnes car nous avons déjà l'information qu'elles contiennent dans la dimension prix au mètre carré désormais.

Enfin toutes nos données sont de l'année 2021. Nous allons retirer cette dimension qui ne devrait pas être discriminante dans le regroupement des données.

```
# Calcul du prix au mètre carré
df_echantillon['prix_m2'] = df_echantillon['valeur_fonciere'] / df_echantillon['surfac
```

```
# Suppression des colonnes inutiles
df_echantillon = df_echantillon.drop(columns=['valeur_fonciere', 'surface_reelle'])
```

```
df_echantillon.head()
```

	code_postal	nom_commune	prix_m2
0	75019	Paris 19e Arrondissement	9871.444128
1	75019	Paris 19e Arrondissement	10045.572493
2	75019	Paris 19e Arrondissement	9194.697790
3	75019	Paris 19e Arrondissement	9469.142168
4	75019	Paris 19e Arrondissement	7463.610005

Nous observons dans les données que nous avons des valeurs différentes de prix au mètre carré pour un même arrondissement (ici le 19ème arrondissement). Il se peut fort que cela soit notre dimension à utiliser pour attribuer les prix au mètre carré les plus élévé dans un département aux locaux commerciaux, et les prix les plus bas aux appartements.

Pour effectuer cette opération, nous allons utiliser l'algorithme du Kmeans qui va rechercher 2 centroïdes à travers les données.

```
from sklearn.cluster import KMeans
```

```
# Préparation des données pour KMeans (on n'utilise que la colonne 'prix_m2')
X = df_echantillon[['prix_m2']].values
```

```
# Appliquer l'algorithme KMeans pour classifier en 2 clusters
kmeans = KMeans(n_clusters=2, random_state=0, n_init=10).fit(X)
```

```
# Ajouter les labels aux données
df_echantillon['label'] = kmeans.labels_
```

```
# On vérifie les données de la prédiction
```

```
# Calcul des statistiques descriptives pour chaque label
stats = df_echantillon.groupby('label')['prix_m2'].describe()
print(stats)
```

```
       count         mean         std         min          25%          50%  \
label
0       20.0  7408.775030  141.329995  7207.21763  7304.332833  7374.091721
1       20.0  9806.924674  232.955000  9194.69779  9705.214437  9842.717300

                75%           max
label
0      7512.112911   7666.071700
1      9979.592847  10113.195822
```