Analyse des données historiques du marché immobilier parisien

Introduction

Cette analyse vise à explorer les données historiques du marché immobilier parisien entre 2017 et 2021. Nous allons examiner les tendances des prix, l'influence du type de bien et de la localisation, ainsi que les relations entre les différentes variables.

In [19]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [20]:
```
# Load the historical real estate data
file_path = 'C:\\Users\\pc\\OneDrive\\Documents\\projet n 8\\historique_immobilier_paris
historique_data = pd.read_excel(file_path)
historique_data.head()
```

Out[20]:

	date_mutation	valeur_fonciere	adresse_numero	adresse_nom_voie	code_postal	nom_commune	code_type_
0	2017-01-03	5.505597e+05	8	RUE DES COUTURES SAINT GERVAIS	75003	Paris 3e Arrondissement	
1	2017-01-12	1.576492e+06	32	AV MARCEAU	75008	Paris 8e Arrondissement	
2	2017-01-10	6.577574e+05	52	RUE DU FAUBOURG SAINT HONORE	75008	Paris 8e Arrondissement	
3	2017-01-10	2.500868e+05	64	RUE DU VERTBOIS	75003	Paris 3e Arrondissement	
4	2017-01-13	1.762667e+05	25	RUE DES LAVANDIERES STE OPPORT	75001	Paris 1er Arrondissement	

In [9]:
```
print(f"Nombre de lignes avant nettoyage : {historique_data.shape[0]}")

# Suppression des doublons
data_cleaned = historique_data.drop_duplicates()

# Affichage du nombre de lignes après suppression des doublons
print(f"Nombre de lignes après suppression des doublons : {data_cleaned.shape[0]}")
```

```
Nombre de lignes avant nettoyage : 26196
Nombre de lignes après suppression des doublons : 26180
```

Statistiques descriptives

Nous commençons par examiner les statistiques descriptives pour obtenir une vue d'ensemble des données.

In [21]:
```
historique_data.describe(include='all', datetime_is_numeric=True)
```

Out[21]:

	date_mutation	valeur_fonciere	adresse_numero	adresse_nom_voie	code_postal	nom_commune
count	26196	2.619600e+04	26196.000000	26196	26196.000000	26196
unique	NaN	NaN	NaN	2874	NaN	20
top	NaN	NaN	NaN	RUE DE VAUGIRARD	NaN	Paris 18e Arrondissement
freq	NaN	NaN	NaN	137	NaN	2925
mean	2019-06-09 14:09:14.099862784	4.916170e+05	47.449572	NaN	75012.716216	NaN
min	2017-01-02 00:00:00	8.519470e+04	1.000000	NaN	75001.000000	NaN
25%	2018-03-14 00:00:00	2.339439e+05	10.000000	NaN	75009.000000	NaN
50%	2019-05-23 00:00:00	3.545774e+05	27.000000	NaN	75014.000000	NaN
75%	2020-09-16 00:00:00	5.702197e+05	63.000000	NaN	75017.000000	NaN
max	2021-12-31 00:00:00	3.843359e+07	407.000000	NaN	75020.000000	NaN
std	NaN	5.713292e+05	55.733432	NaN	5.269150	NaN

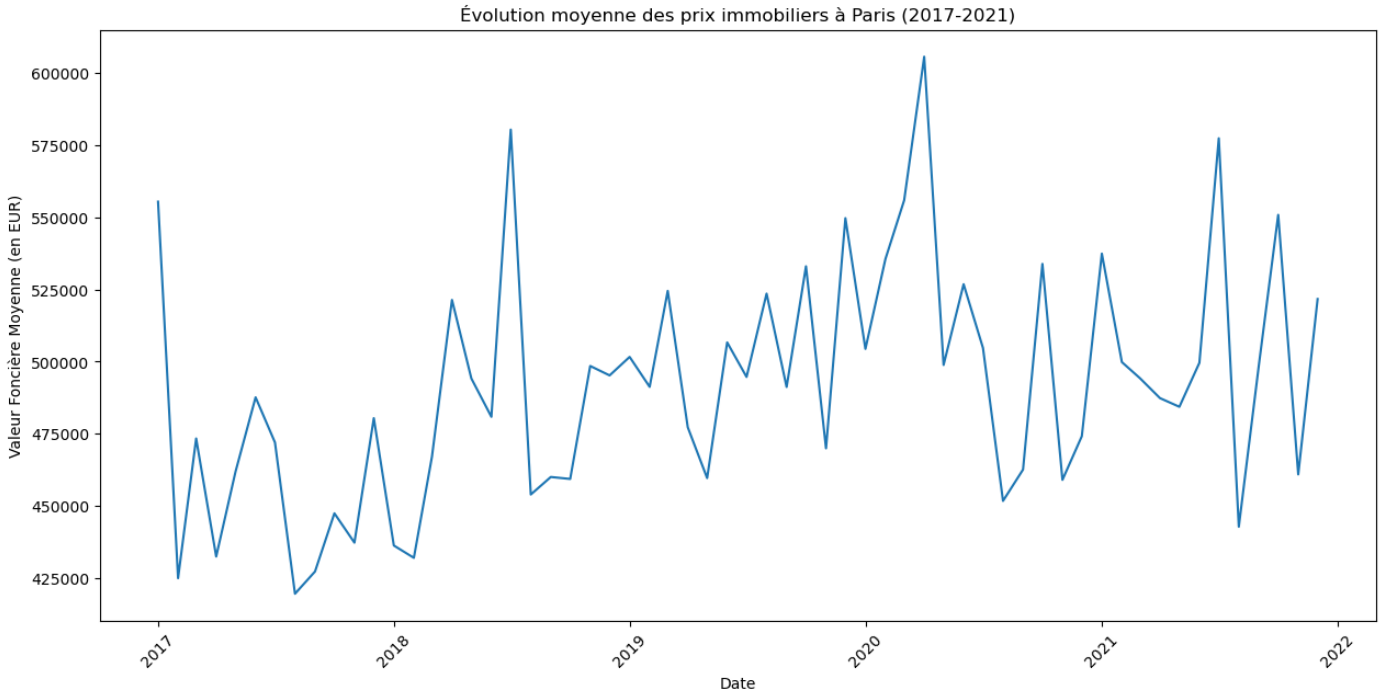
Analyse des tendances des prix

Nous allons maintenant étudier comment les prix immobiliers ont évolué au fil du temps.

In [22]:

```
# Grouping the data by the mutation date and calculating the average value fonciere
grouped_data = historique_data.groupby(historique_data['date_mutation'].dt.to_period('M'
# Resetting index to convert the PeriodIndex to DateTimeIndex for plotting
grouped_data.reset_index(inplace=True)
grouped_data['date_mutation'] = grouped_data['date_mutation'].dt.to_timestamp()
# Plotting the average value fonciere over time
plt.figure(figsize=(15, 7))
sns.lineplot(x='date_mutation', y='valeur_fonciere', data=grouped_data)
plt.title('Évolution moyenne des prix immobiliers à Paris (2017-2021)')
plt.xlabel('Date')
plt.ylabel('Valeur Foncière Moyenne (en EUR)')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\pc\AppData\Local\Temp\ipykernel_17304\3168675477.py:2: FutureWarning: The defau
lt value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, nu
meric_only will default to False. Either specify numeric_only or select only columns whi
ch should be valid for the function.
  grouped_data = historique_data.groupby(historique_data['date_mutation'].dt.to_period
('M')).mean()
```



In [36]:
```
historique_data['type_local'].unique()
```

Out[36]:
```
array(['Appartement', 'Local industriel. commercial ou assimilé'],
      dtype=object)
```

In [37]:
```
total_transactions = historique_data.shape[0]
transactions_by_type = historique_data['type_local'].value_counts()
```

In [39]:
```
date_range = historique_data['date_mutation'].agg(['min', 'max'])
```

In [40]:
```
apartments_data = historique_data[historique_data['type_local'] == 'Appartement']
apartments_data['prix_m2'] = apartments_data['valeur_fonciere'] / apartments_data['surfa
avg_price_per_m2_global = apartments_data.groupby(apartments_data['date_mutation'].dt.ye
```

```
C:\Users\pc\AppData\Local\Temp\ipykernel_17304\3093340506.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  apartments_data['prix_m2'] = apartments_data['valeur_fonciere'] / apartments_data['sur
face_reelle']
```

In [63]:
```
# Calcul du nombre total de transactions
total_transactions = len(historique_data)

# Calcul du nombre de transactions pour les appartements
apartments_transactions = len(historique_data[historique_data['type_local'] == 'Appartem

# Calcul du nombre de transactions pour les locaux commerciaux
commercial_transactions = len(historique_data[historique_data['type_local'] == 'Local co

print('Le nombre de transactions dans les données est :', total_transactions)
print('Le nombre de transactions pour les appartements dans les données est :', apartmen
print('Le nombre de transactions pour les locaux commerciaux dans les données est :', co
```

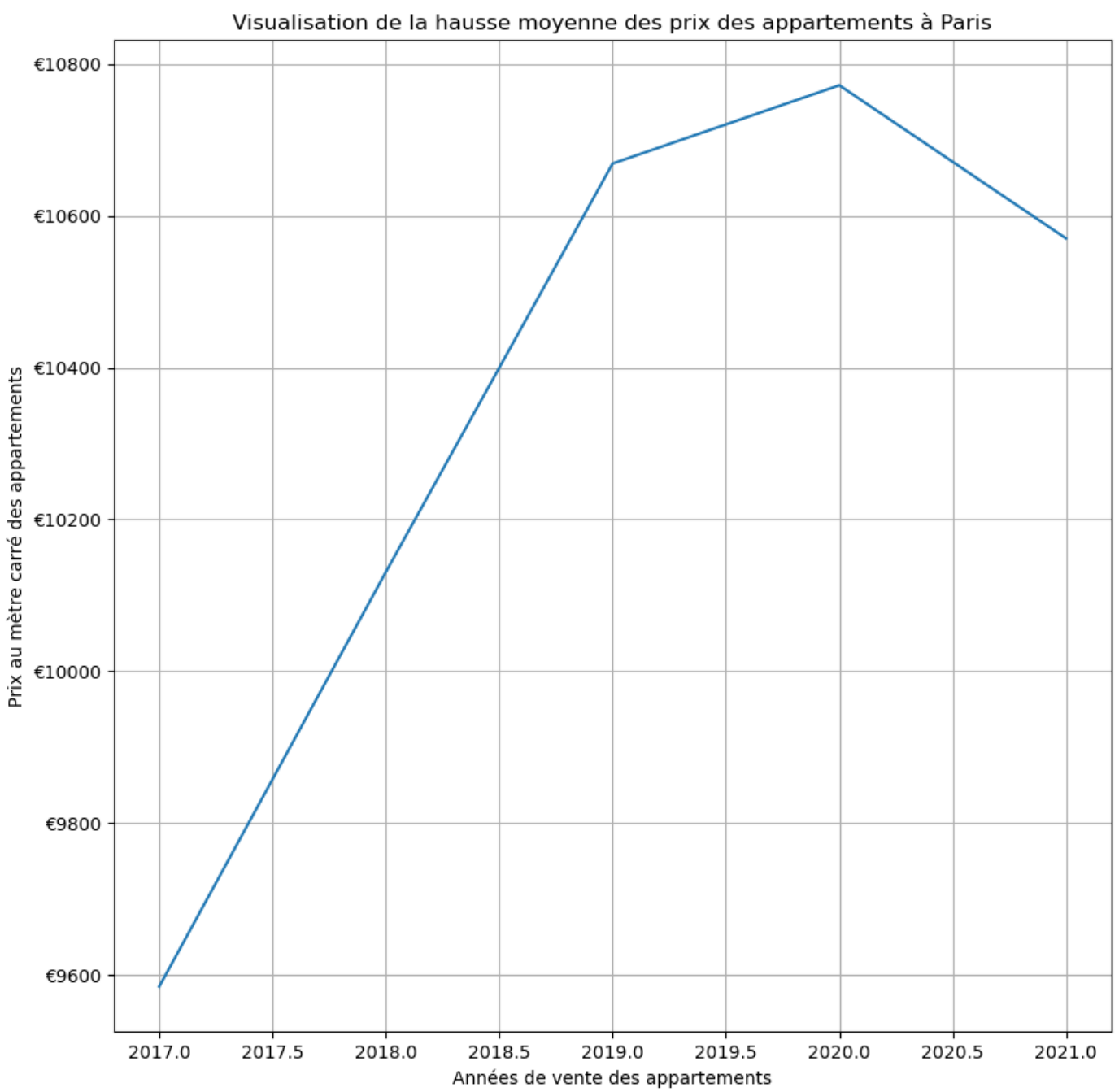
```
Le nombre de transactions dans les données est : 26196
Le nombre de transactions pour les appartements dans les données est : 24353
Le nombre de transactions pour les locaux commerciaux dans les données est : 0
```

In [64]:
```
#On vérifie également la plage de l'historique disponible
historique_data['date_mutation'] = pd.to_datetime(historique_data['date_mutation'], erro

# Trouver la première date de transaction
premiere_date = historique_data['date_mutation'].min()

# Trouver la dernière date de transaction
derniere_date = historique_data['date_mutation'].max()

print('La première date de transaction dans les données est le :', premiere_date.strftim
print('La dernière date de transaction dans les données est le :', derniere_date.strftim
```

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

In [65]:
```
historique_data['annee'] = historique_data['date_mutation'].dt.year
prix_moyen_par_annee = historique_data.groupby('annee')['prix_m2'].mean()
```

In [66]:
```
#Création d'un graphique pour visualiser la hausse de la moyenne des prix
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

plt.figure(figsize=(10, 10))
plt.plot(prix_moyen_par_annee.index, prix_moyen_par_annee.values)
plt.xlabel('Années de vente des appartements')
plt.ylabel('Prix au mètre carré des appartements')

formatter = ticker.FormatStrFormatter('€%d')
plt.gca().yaxis.set_major_formatter(formatter)

plt.grid()
plt.title('Visualisation de la hausse moyenne des prix des appartements à Paris')
plt.show()
```

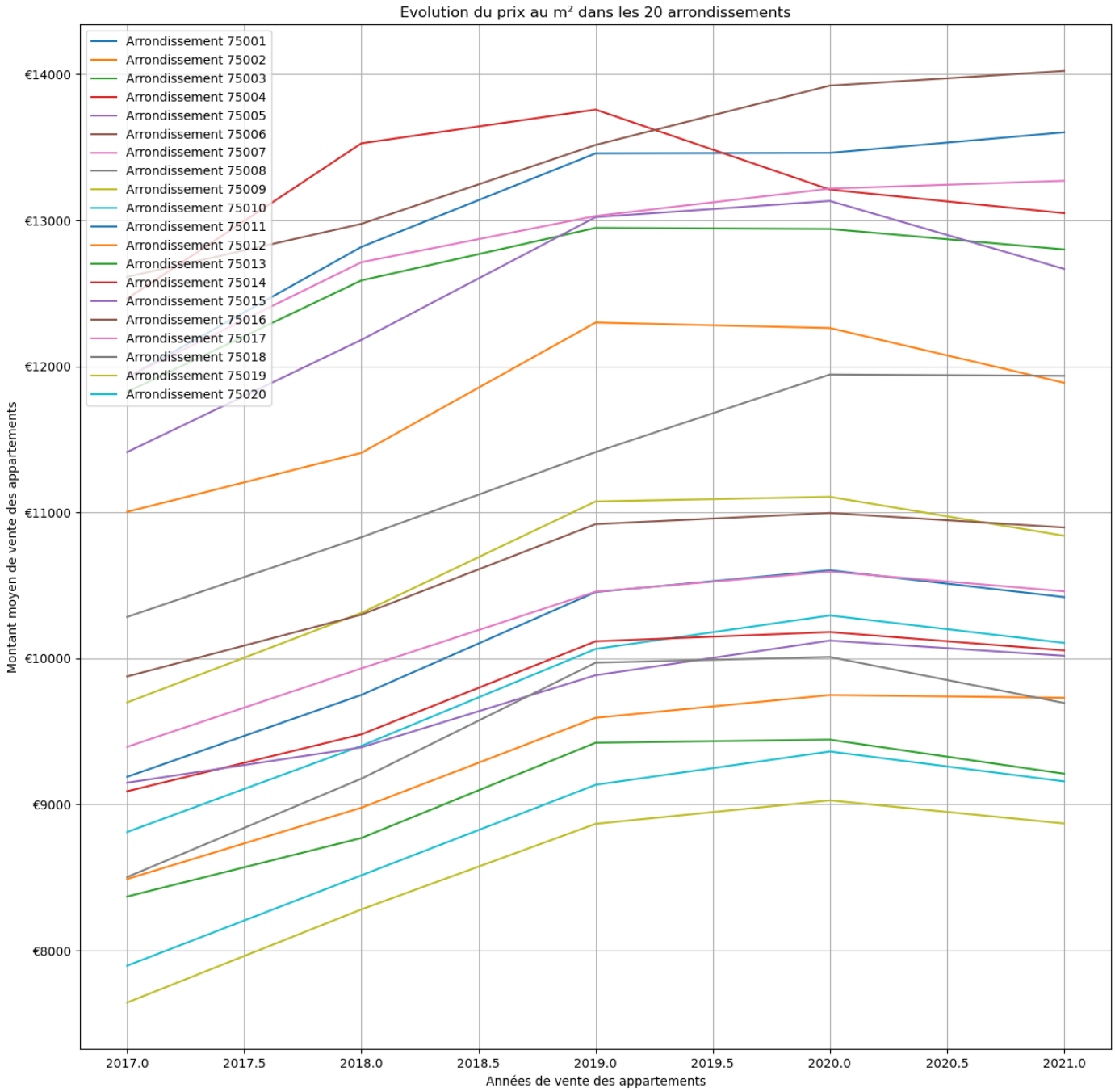


In [71]:

```
#On affiche l'évolution du prix au m² par arrondissement dans Paris
plt.figure(figsize=(15,15))
historique_data['annee'] = historique_data['date_mutation'].dt.year
prix_moyen_par_annee_arrondissement = historique_data.groupby(['annee', 'code_postal'])[
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

plt.figure(figsize=(15, 15))
interval_historique = ['2017', '2018', '2019', '2020', '2021']

# Tracer chaque ligne
for arrondissement in prix_moyen_par_annee_arrondissement.columns:
    plt.plot(prix_moyen_par_annee_arrondissement.index, prix_moyen_par_annee_arrondissem

plt.xlabel('Années de vente des appartements')
plt.ylabel('Montant moyen de vente des appartements')

formatter = ticker.FormatStrFormatter('€%d')
plt.gca().yaxis.set_major_formatter(formatter)

plt.grid()
volution du prix au m² dans les 20 arrondissements')
```

```
plt.legend(loc='upper left')
plt.show()
```

```
<Figure size 1500x1500 with 0 Axes>
```

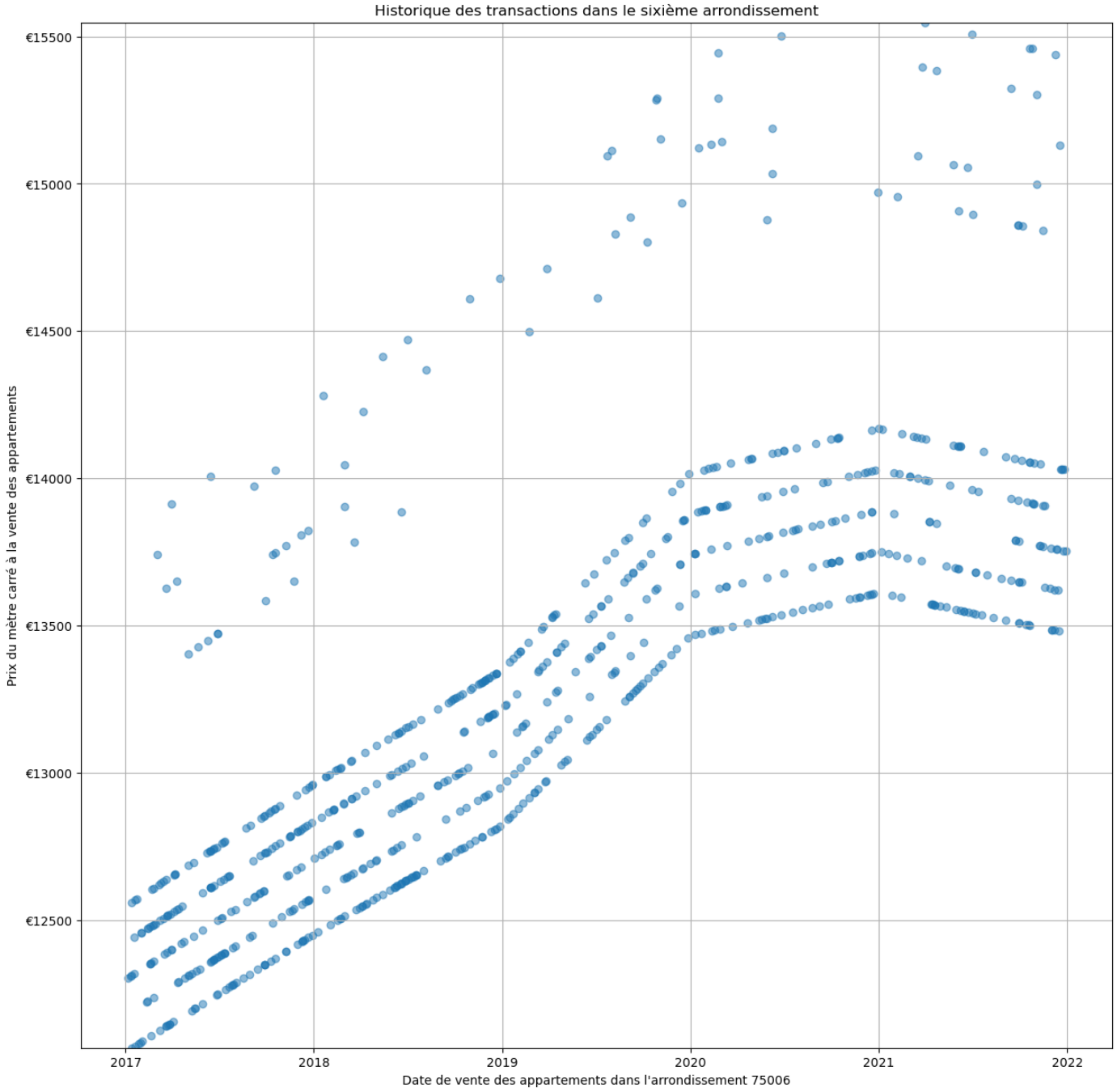


In [76]:

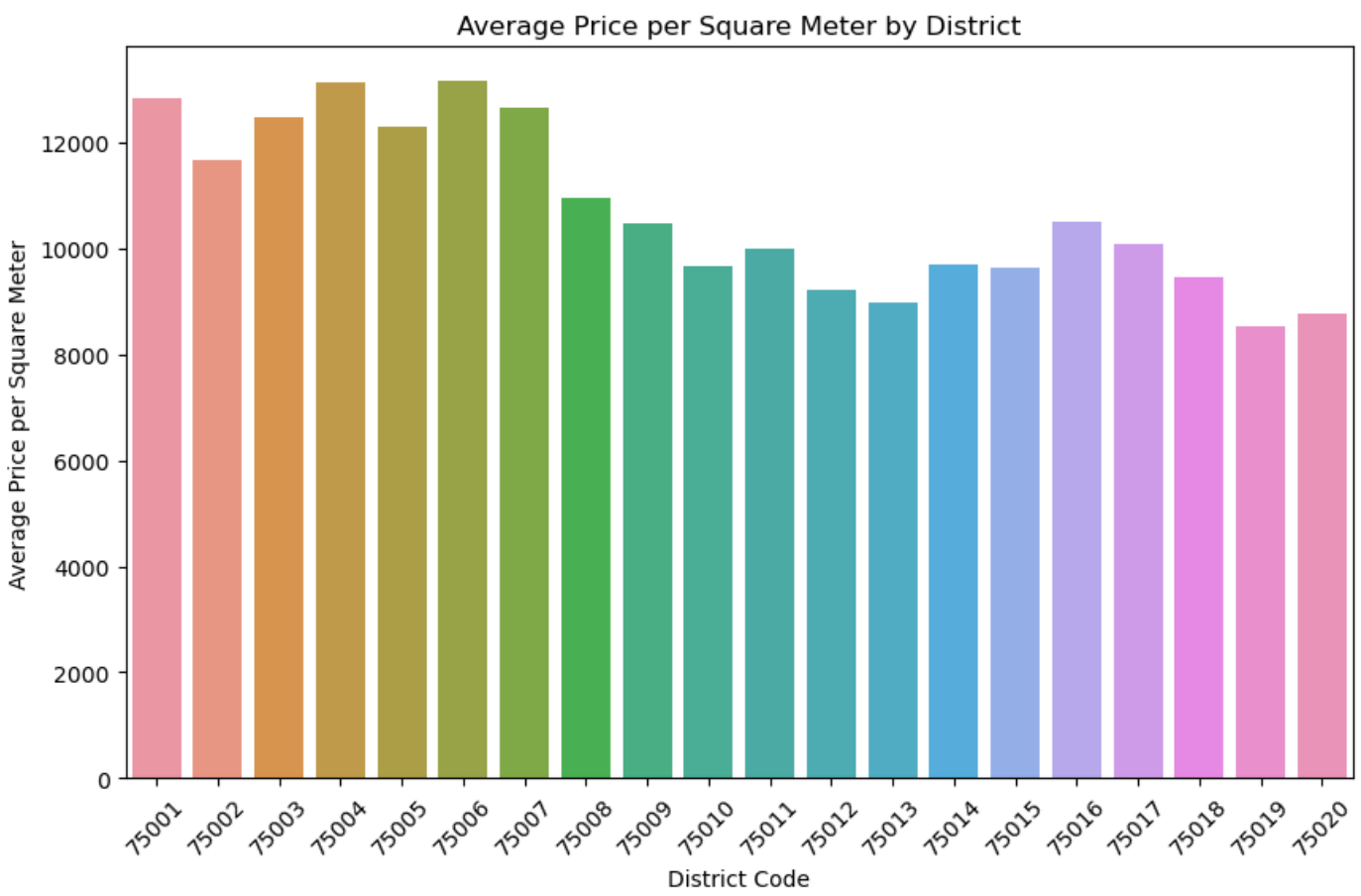
```
#Vérifions le nombre de transaction dans le 6ème car le prix semble élevé

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

import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker



# Filtrer les données pour le 6ème arrondissement
data_75006 = historique_data[historique_data['code_postal'] == 75006]

# Convertir 'date_mutation' en datetime si ce n'est pas déjà le cas
data_75006['date_mutation'] = pd.to_datetime(data_75006['date_mutation'], errors='coerce

# Créer un scatter plot
                                              gsize=(15, 15))
```

```
plt.scatter(data_75006['date_mutation'], data_75006['prix_m2'], alpha=0.5)  # alpha pour

# Définir les étiquettes et le titre du graphique
plt.xlabel("Date de vente des appartements dans l'arrondissement 75006")
plt.ylabel('Prix du mètre carré à la vente des appartements')

# Définir le formateur de l'axe Y pour les devises
formatter = ticker.FormatStrFormatter('€%d')
plt.gca().yaxis.set_major_formatter(formatter)

# Ajuster les limites de l'axe Y en fonction des données réelles
# Vous devez remplacer ces limites par les valeurs appropriées pour vos données
plt.ylim(data_75006['prix_m2'].min(), data_75006['prix_m2'].max())

# Ajouter une grille pour une meilleure lisibilité
plt.grid(which='both')

# Ajouter un titre
plt.title("Historique des transactions dans le sixième arrondissement")

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

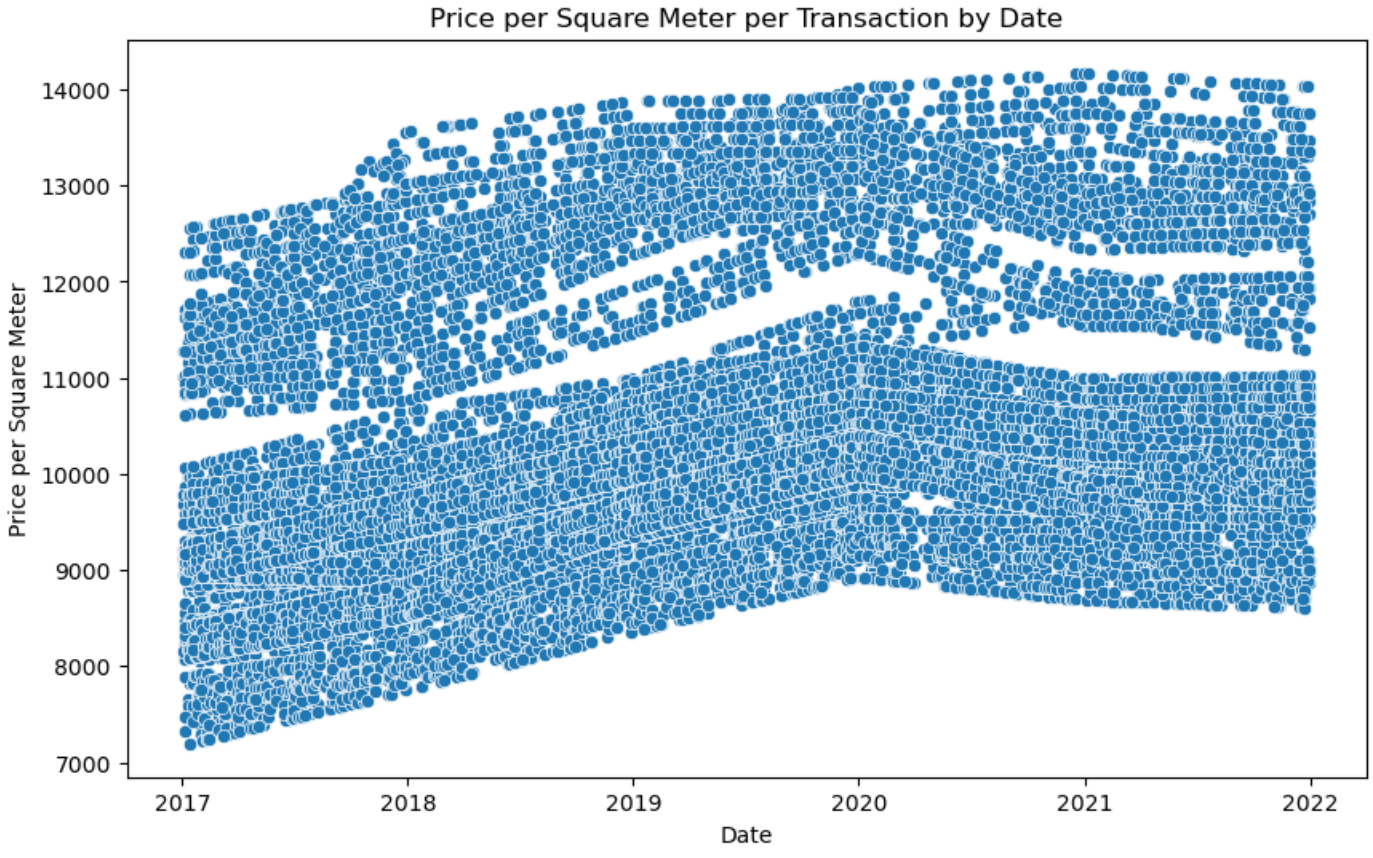
```
C:\Users\pc\AppData\Local\Temp\ipykernel_17304\1223943792.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  data_75006['date_mutation'] = pd.to_datetime(data_75006['date_mutation'], errors='coer
ce')
```



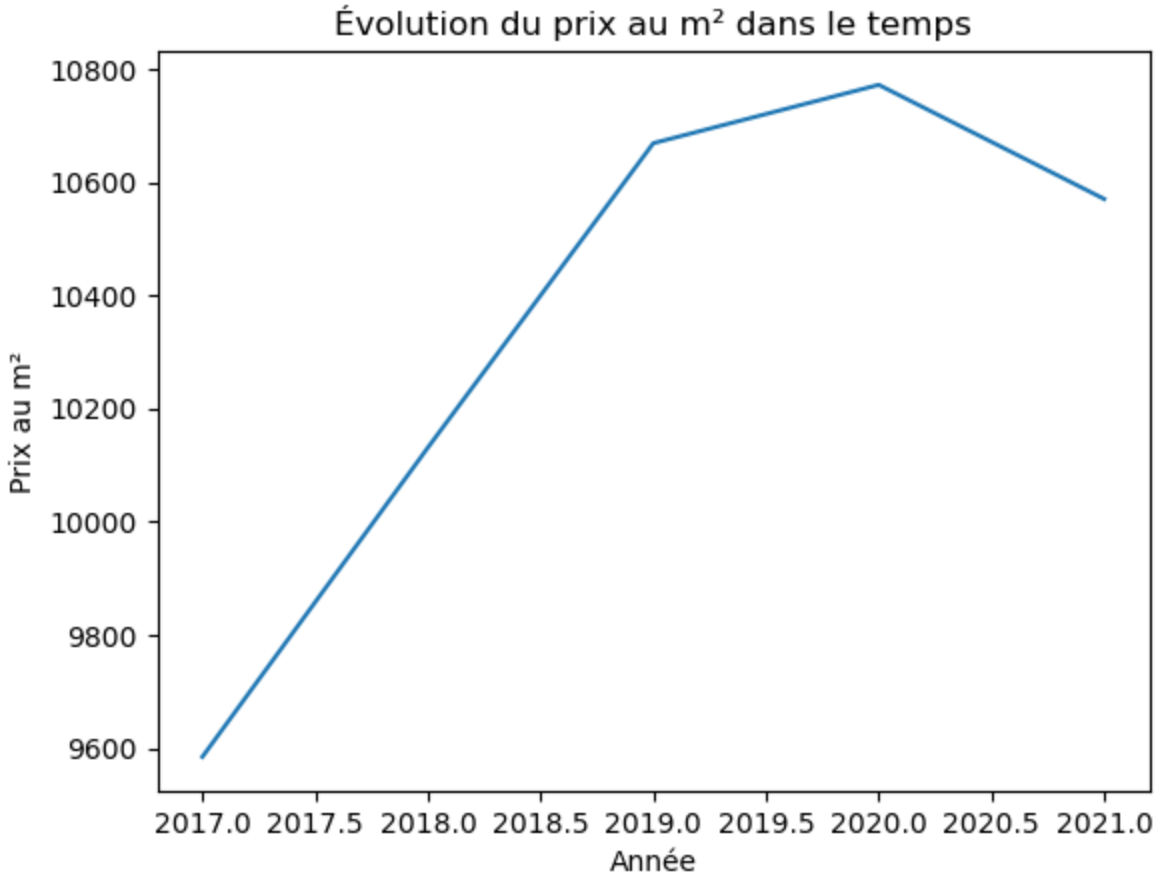
In [49]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x=avg_price_per_m2_by_district.index, y=avg_price_per_m2_by_district.values)
plt.title('Average Price per Square Meter by District')
plt.xlabel('District Code')
plt.ylabel('Average Price per Square Meter')
plt.xticks(rotation=45)
plt.show()
```



In [50]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='date_mutation', y='prix_m2', data=apartments_data)
plt.title('Price per Square Meter per Transaction by Date')
plt.xlabel('Date')
plt.ylabel('Price per Square Meter')
plt.show()
```



In [61]:
```
# Pour voir l'évolution du prix moyen au m² par année
historique_data['prix_m2'] = historique_data['valeur_fonciere'] / historique_data['surfa

historique_data['annee'] = historique_data['date_mutation'].dt.year
evolution_prix = historique_data.groupby('annee')['prix_m2'].mean()

import matplotlib.pyplot as plt
import seaborn as sns

sns.lineplot(data=evolution_prix)
plt.title('Évolution du prix au m² dans le temps')
plt.xlabel('Année')
plt.ylabel('Prix au m²')
plt.show()
```

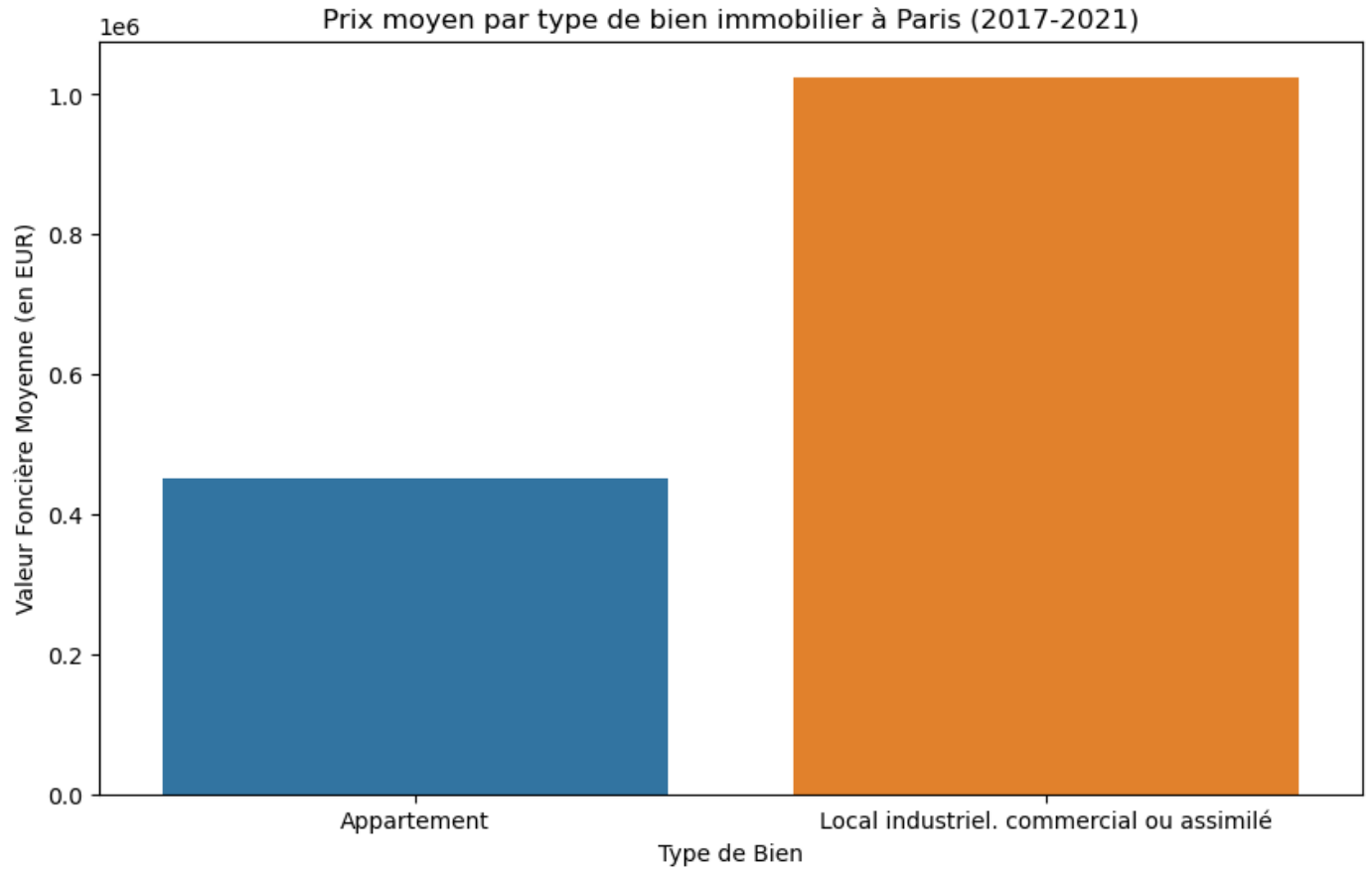


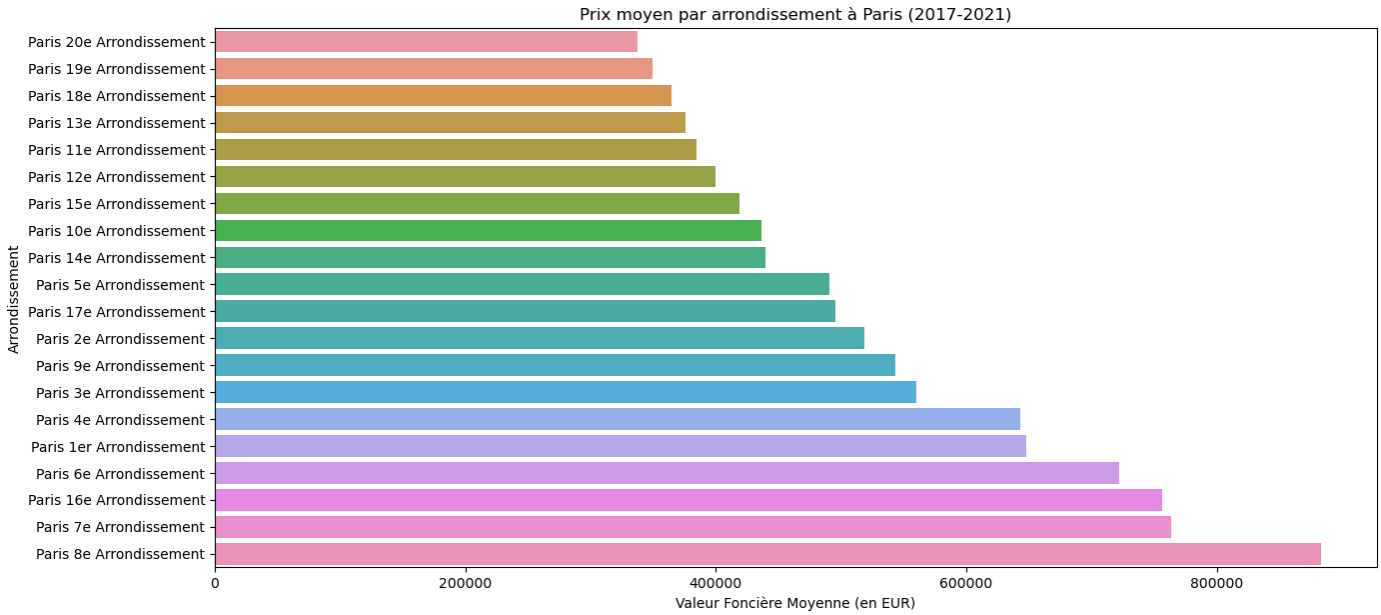
Analyse de l'impact du type de bien et de la localisation sur les prix

Dans cette section, nous examinerons comment différents types de biens et la localisation (arrondissements de Paris) affectent les prix immobiliers.

In [45]:
```
# Analyzing the impact of property type on the prices
type_price_data = historique_data.groupby('type_local')['valeur_fonciere'].mean().reset_
# Plotting the average value fonciere by property type
plt.figure(figsize=(10, 6))
sns.barplot(x='type_local', y='valeur_fonciere', data=type_price_data)
plt.title('Prix moyen par type de bien immobilier à Paris (2017-2021)')
plt.xlabel('Type de Bien')
plt.ylabel('Valeur Foncière Moyenne (en EUR)')
plt.show()
# Analyzing the impact of location (Parisian arrondissements) on the prices
t_price_data = historique_data.groupby('nom_commune')['valeur_fonciere'].me
```

```
# Plotting the average value fonciere by arrondissement
plt.figure(figsize=(15, 7))
sns.barplot(x='valeur_fonciere', y='nom_commune', data=arrondissement_price_data)
plt.title('Prix moyen par arrondissement à Paris (2017-2021)')
plt.xlabel('Valeur Foncière Moyenne (en EUR)')
plt.ylabel('Arrondissement')
plt.show()
```



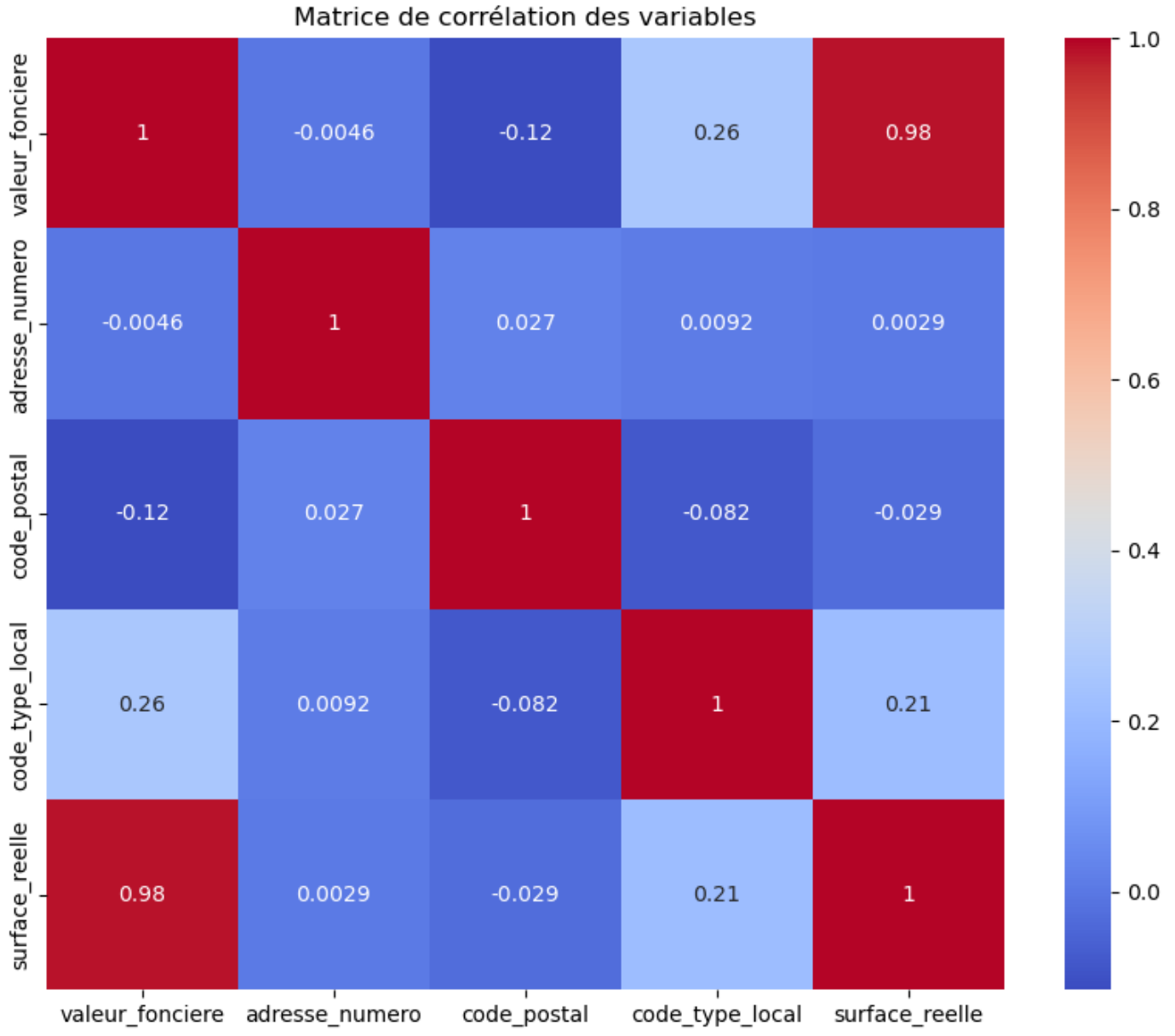


Analyse des corrélations

Nous allons maintenant examiner les corrélations entre différentes variables pour identifier les facteurs les plus importants.

In [14]:
```
# Correlation matrix
correlation_matrix = historique_data.corr()
# Plotting the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Matrice de corrélation des variables')
plt.show()
```

```
C:\Users\pc\AppData\Local\Temp\ipykernel_17304\1136189660.py:2: FutureWarning: The defau
lt value of numeric_only in DataFrame.corr is deprecated. In a future version, it will d
efault to False. Select only valid columns or specify the value of numeric_only to silen
ce this warning.
  correlation_matrix = historique_data.corr()
```



Entraînement d'un algorithme de régression linéaire

Dans cette section, nous allons préparer les données pour la modélisation, entraîner un modèle de régression linéaire et évaluer sa performance.

In [23]:
```
# Préparation des données pour le modèle de régression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

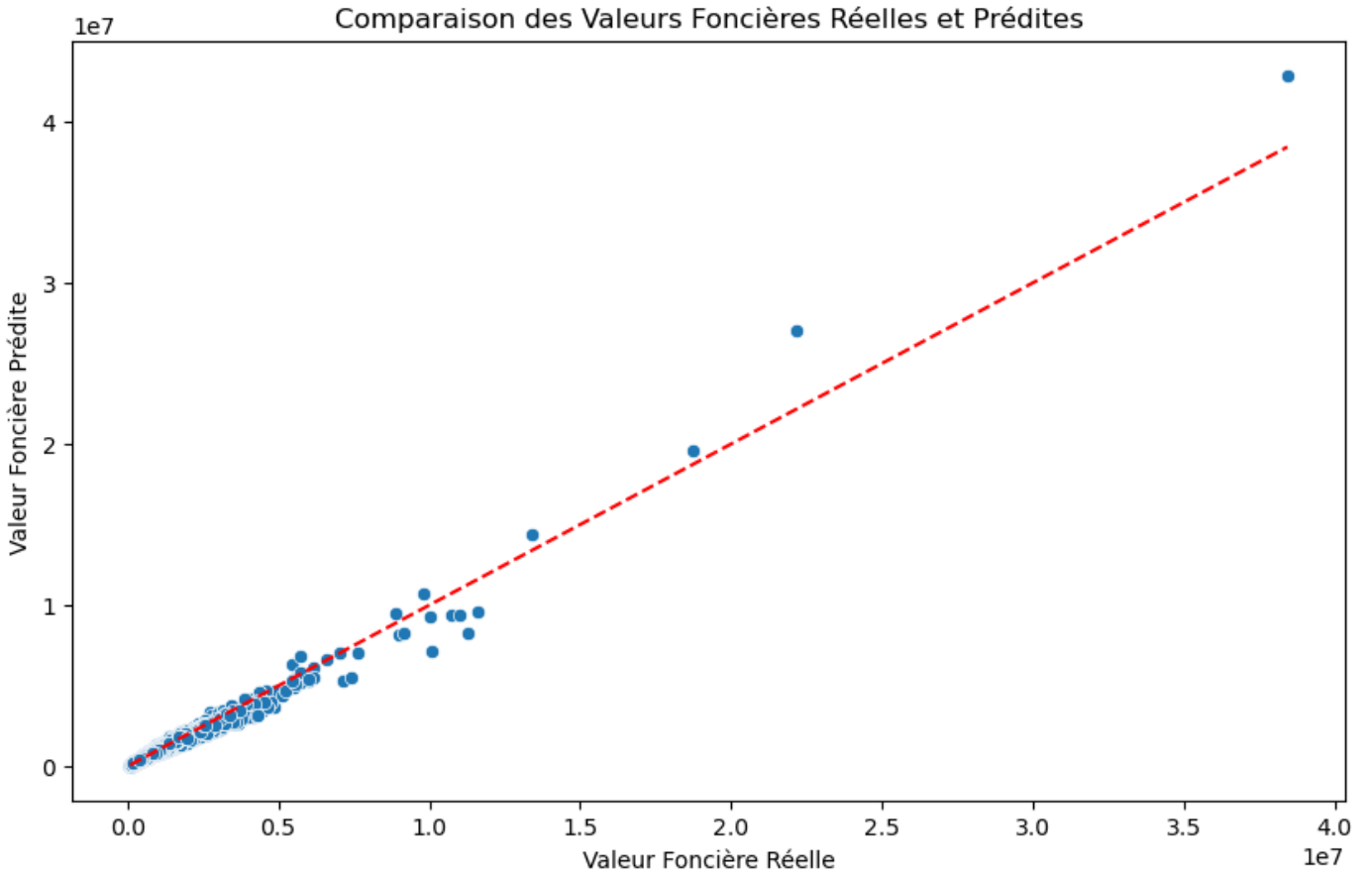
```
# Sélection des variables pertinentes et suppression des valeurs manquantes
regression_data = historique_data[['valeur_fonciere', 'surface_reelle', 'code_postal']].

# Séparation des variables indépendantes (X) et de la variable dépendante (y)
X = regression_data[['surface_reelle', 'code_postal']]
y = regression_data['valeur_fonciere']

# Division des 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=0)

# Entraînement du modèle de régression linéaire
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Prédiction sur l'ensemble de test
y_pred = regressor.predict(X_test)

# Évaluation de la performance du modèle
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
'MSE: {:.2f}, R²: {:.2f}'.format(mse, r2)
```

Out[23]:
```
'MSE: 5044188833.46, R²: 0.98'
```

In [25]:
```
# Sélection des caractéristiques et de la variable cible
X = historique_data[['surface_reelle', 'code_postal']]  # Exemple de caractéristiques
y = historique_data['valeur_fonciere']  # Variable cible

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
from sklearn.linear_model import LinearRegression

# Création du modèle de régression linéaire
model = LinearRegression()

# Entraînement du modèle avec les données d'entraînement
model.fit(X_train, y_train)
# Prédiction des valeurs foncières
new_predictions = model.predict(X_test)
 # Création d'un DataFrame pour les résultats de test avec les prédictions
test_results = X_test.copy()
test_results['valeur_fonciere_reelle'] = y_test
test_results['valeur_fonciere_predite'] = new_predictions

# Affichage des premières lignes du DataFrame avec les prédictions
print(test_results.head())
```

```
       surface_reelle  code_postal  valeur_fonciere_reelle  \
7178               24        75011           228793.952877
10733              16        75004           213326.691507
20031              29        75018           290498.700685
9698               80        75018           704732.602740
18238              80        75012           869560.789041

       valeur_fonciere_predite
7178              258766.368479
10733             236767.278972
20031             248129.961266
9698              802933.405505
18238             858672.464144
```

In [31]:
```
# Supposons que 'model' est le modèle de prédiction déjà entraîné
# et que vous utilisez 'surface_reelle' et 'code_postal' comme variables pour prédire 'v

# Prédictions
predictions = model.predict(historique_data[['surface_reelle', 'code_postal']])

# Ajout des prédictions au DataFrame
historique_data['valeur_fonciere_predite'] = predictions

# visualiser les données
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(x='valeur_fonciere', y='valeur_fonciere_predite', data=historique_data)

plt.plot([historique_data['valeur_fonciere'].min(), historique_data['valeur_fonciere'].m
         [historique_data['valeur_fonciere'].min(), historique_data['valeur_fonciere'].m
         color='red', linestyle='--')

plt.title('Comparaison des Valeurs Foncières Réelles et Prédites')
plt.xlabel('Valeur Foncière Réelle')
plt.ylabel('Valeur Foncière Prédite')
plt.show()
```



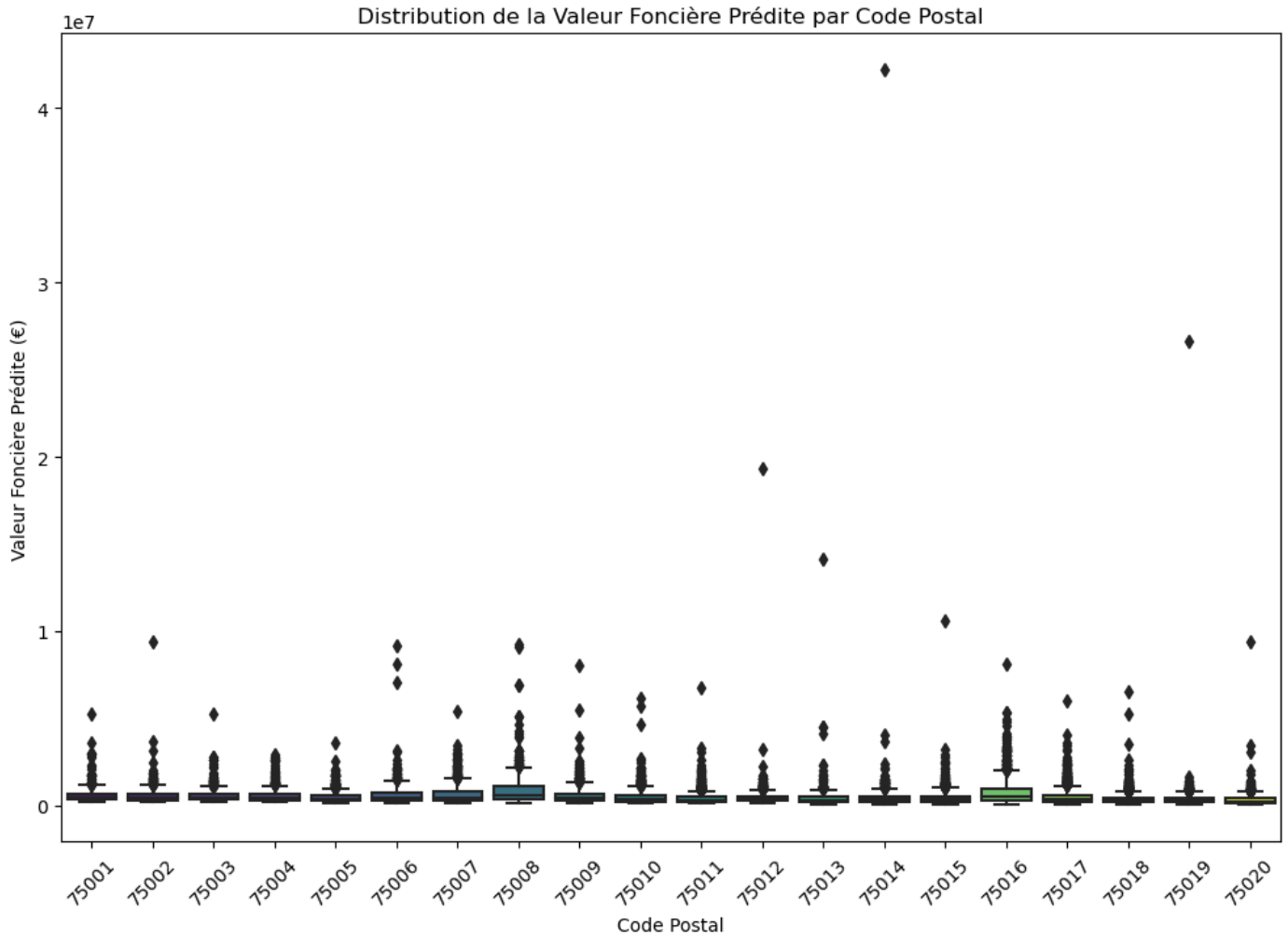
In [33]:
```
from sklearn.linear_model import LinearRegression

# Préparation des données
X = historique_data[['surface_reelle', 'code_postal']].dropna()
y = historique_data['valeur_fonciere'].dropna()  # Remplacez par la colonne appropriée s


X = X.loc[y.index]

# entraînement du modèle de régression
```

```
model = LinearRegression()
model.fit(X, y)

# Génération des prédictions
new_predictions = model.predict(X)

# Ajout des prédictions au DataFrame original
historique_data.loc[y.index, 'valeur_fonciere_predite'] = new_predictions

# Affichage des premières lignes du DataFrame avec les prédictions
print(historique_data[['surface_reelle', 'code_postal', 'valeur_fonciere_predite']].head
```

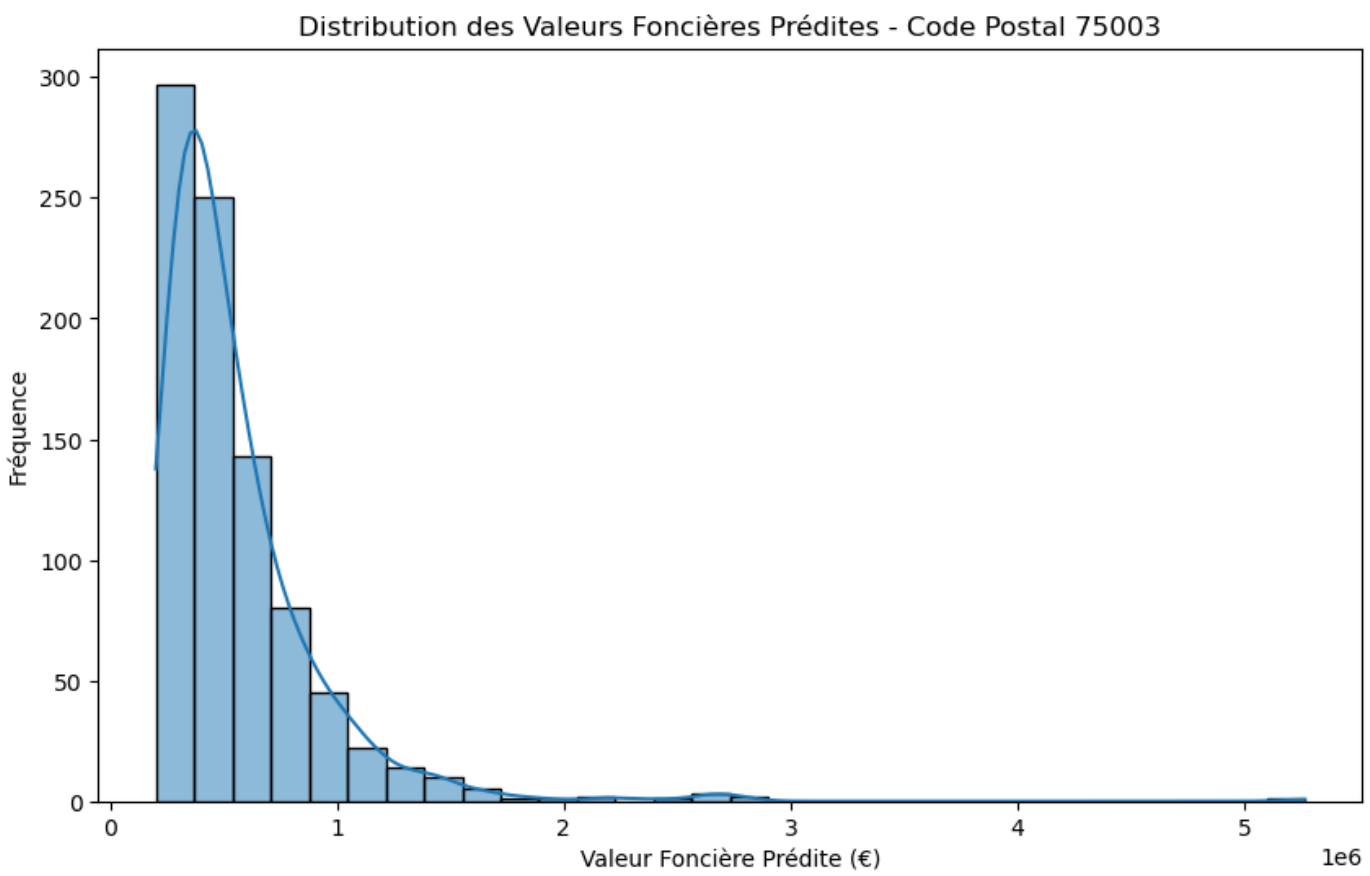
```
   surface_reelle  code_postal  valeur_fonciere_predite
0              50        75003             6.164236e+05
1             163        75008             1.778013e+06
2              66        75008             7.405993e+05
3              22        75003             3.169639e+05
4              15        75001             2.608767e+05
```

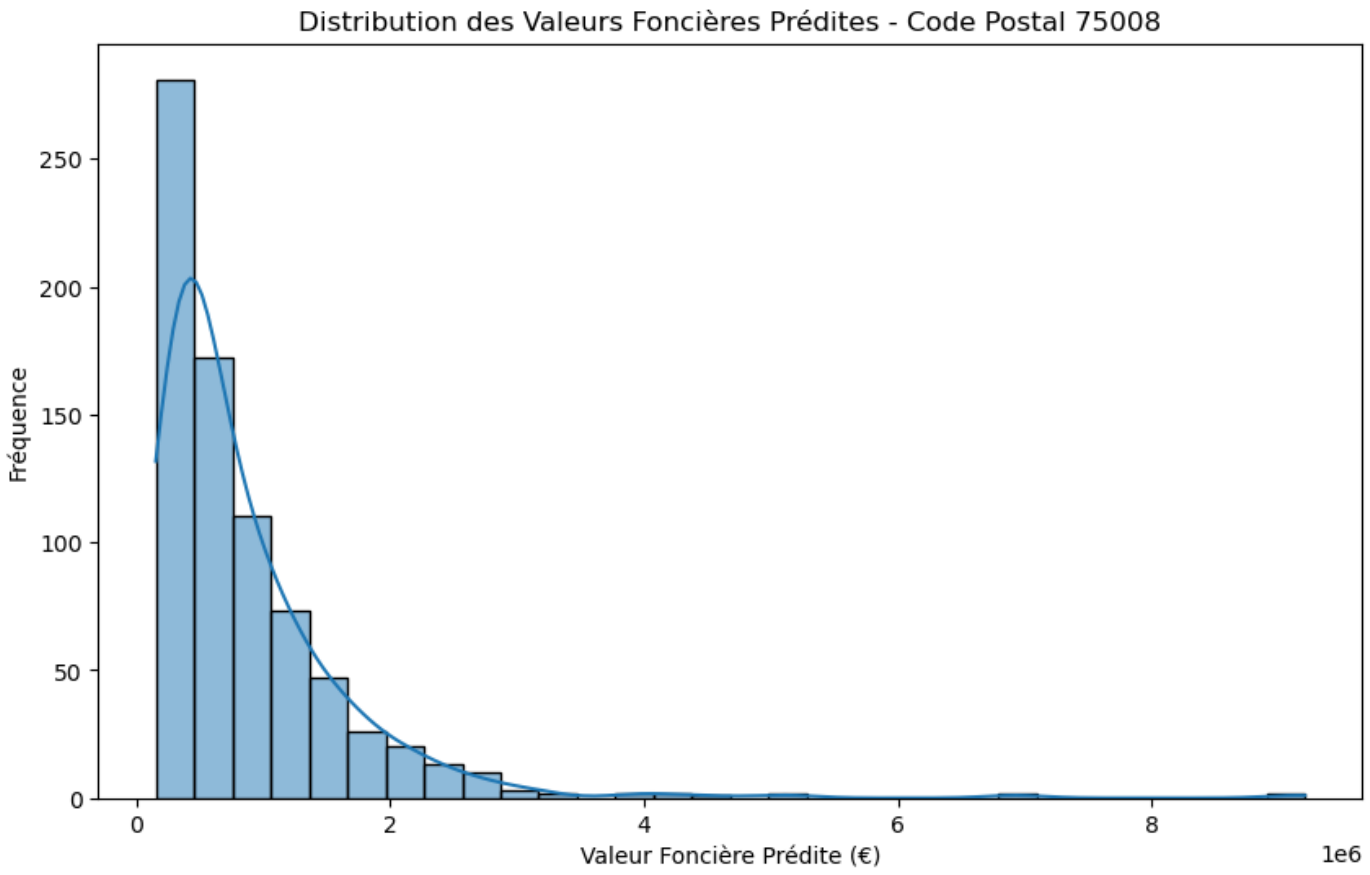
In [34]:

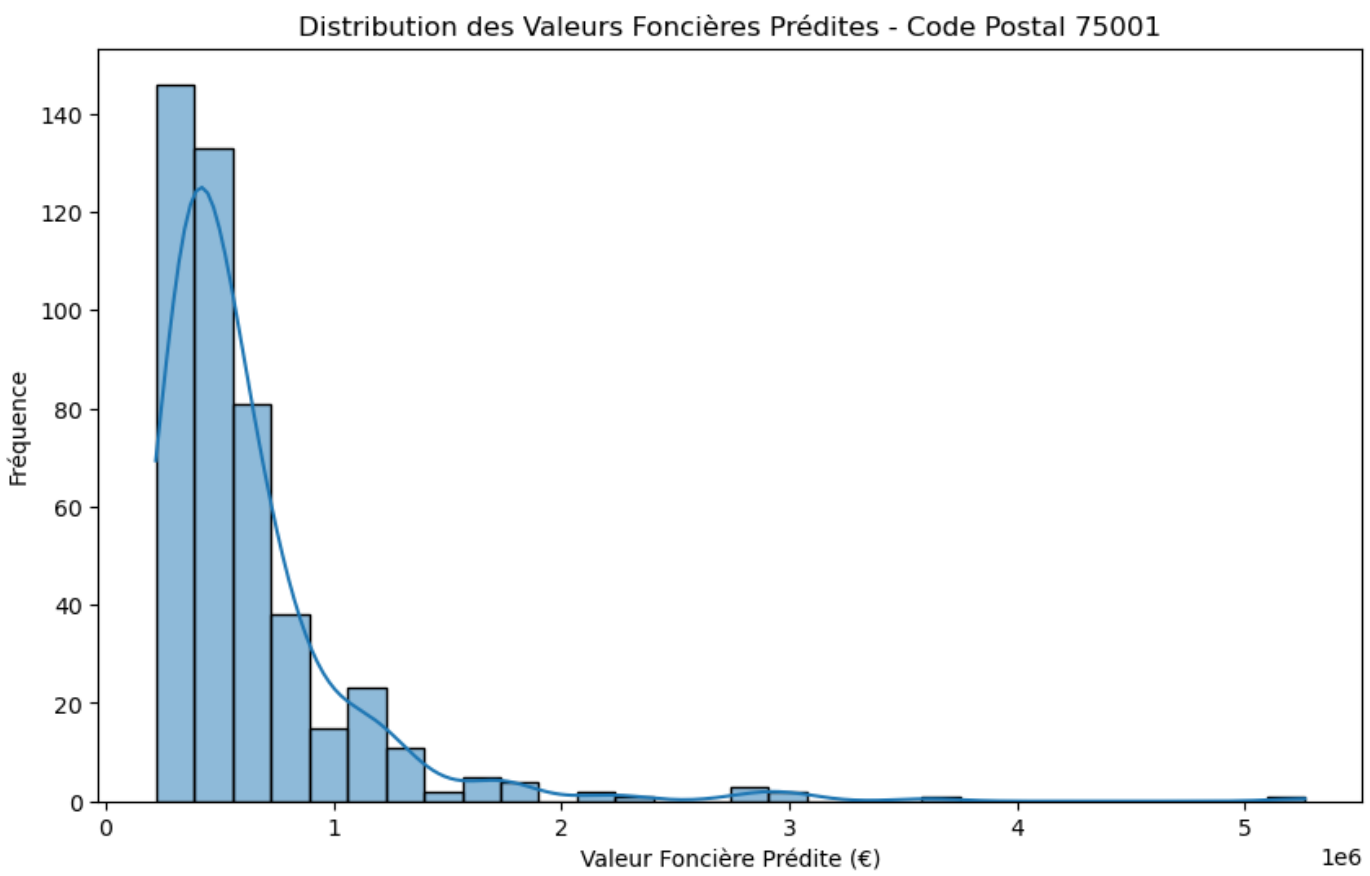
```
# Création de la variable visualization_data à partir de new_data
visualization_data = historique_data[['surface_reelle', 'code_postal', 'valeur_fonciere_

# visualization_data pour la visualisation
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
sns.boxplot(data=visualization_data, x='code_postal', y='valeur_fonciere_predite', palet

plt.title('Distribution de la Valeur Foncière Prédite par Code Postal')
plt.xlabel('Code Postal')
plt.ylabel('Valeur Foncière Prédite (€)')
plt.xticks(rotation=45)  # Rotation des étiquettes pour une meilleure lisibilité
plt.show()
```

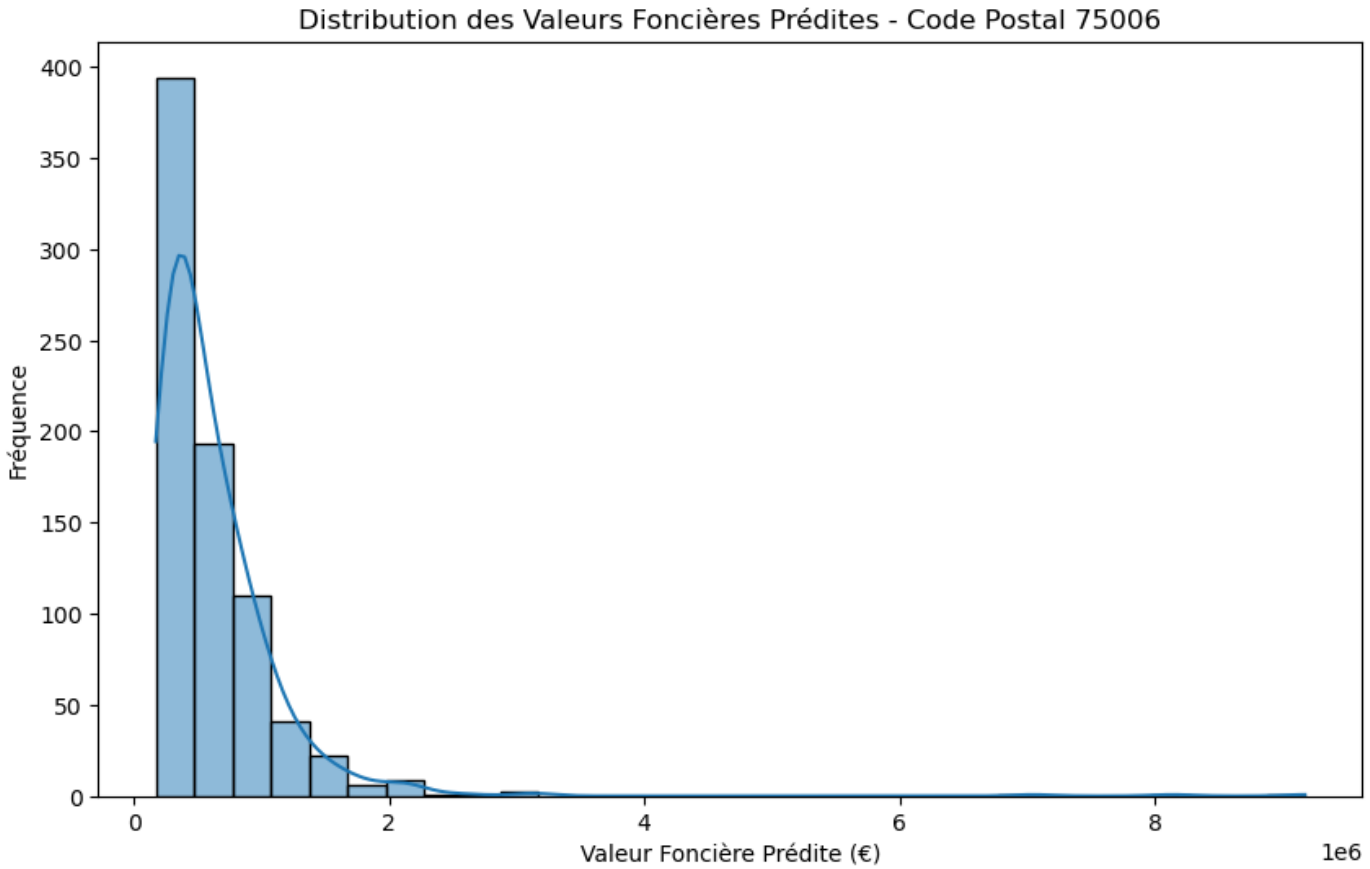


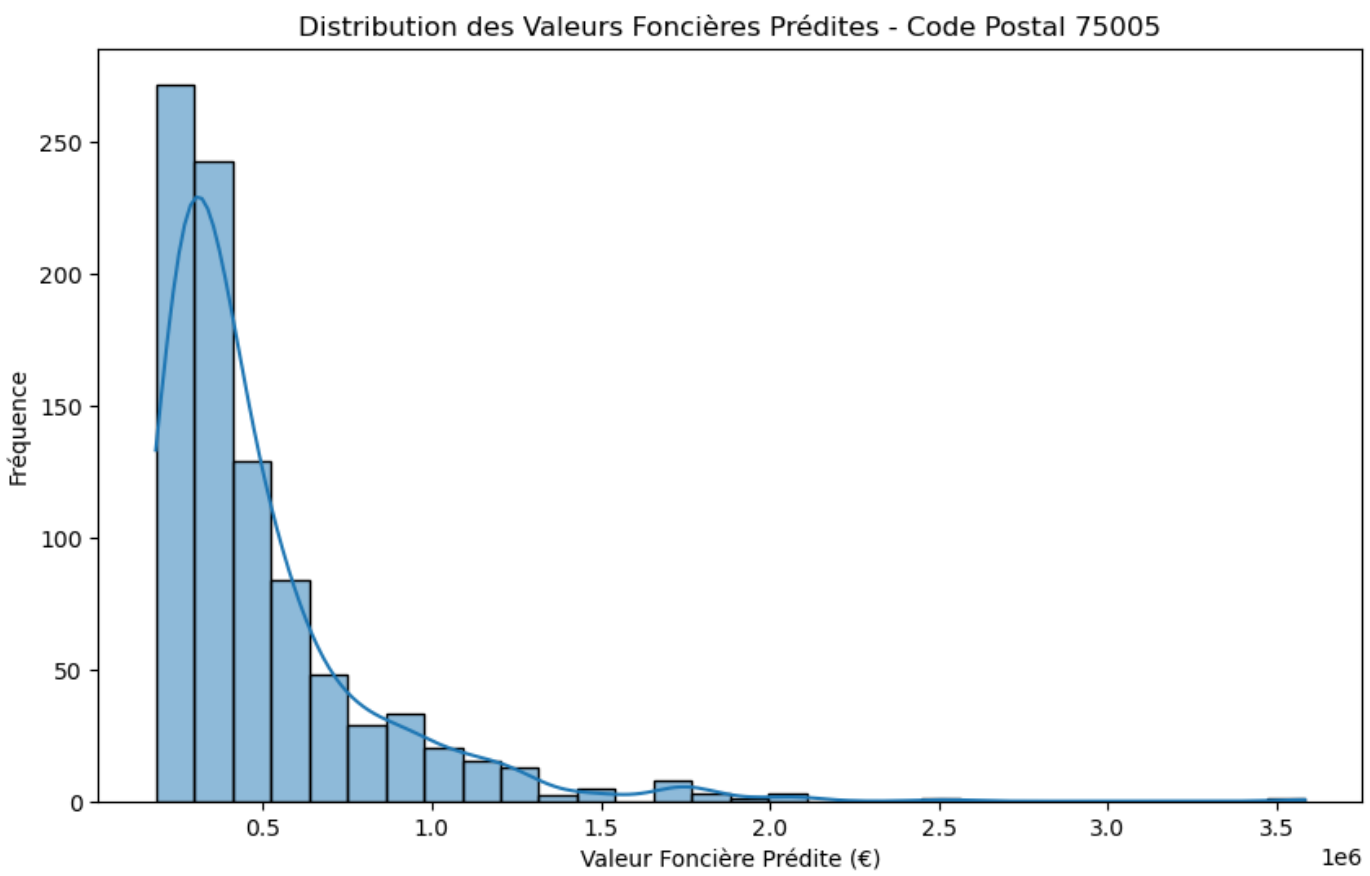
In [38]:
```python
# Obtention des codes postaux uniques
unique_postal_codes = visualization_data['code_postal'].unique()

# Création d'un histogramme pour chaque code postal
for postal_code in unique_postal_codes:
    subset = visualization_data[visualization_data['code_postal'] == postal_code]

    plt.figure(figsize=(10, 6))
    sns.histplot(subset['valeur_fonciere_predite'], kde=True, bins=30)

    plt.title(f'Distribution des Valeurs Foncières Prédites - Code Postal {postal_code}'
    plt.xlabel('Valeur Foncière Prédite (€)')
    plt.ylabel('Fréquence')
    plt.show()
```

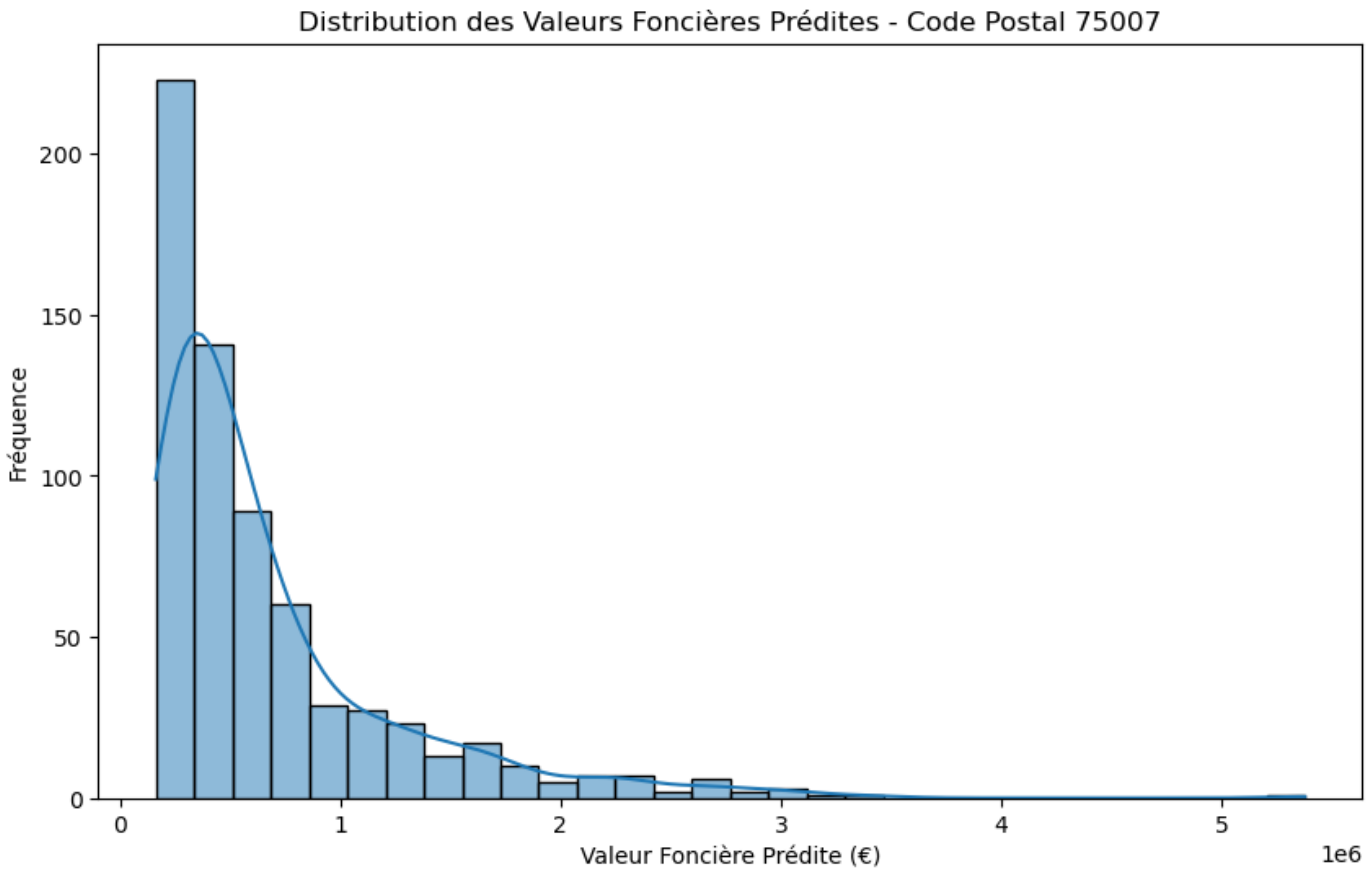


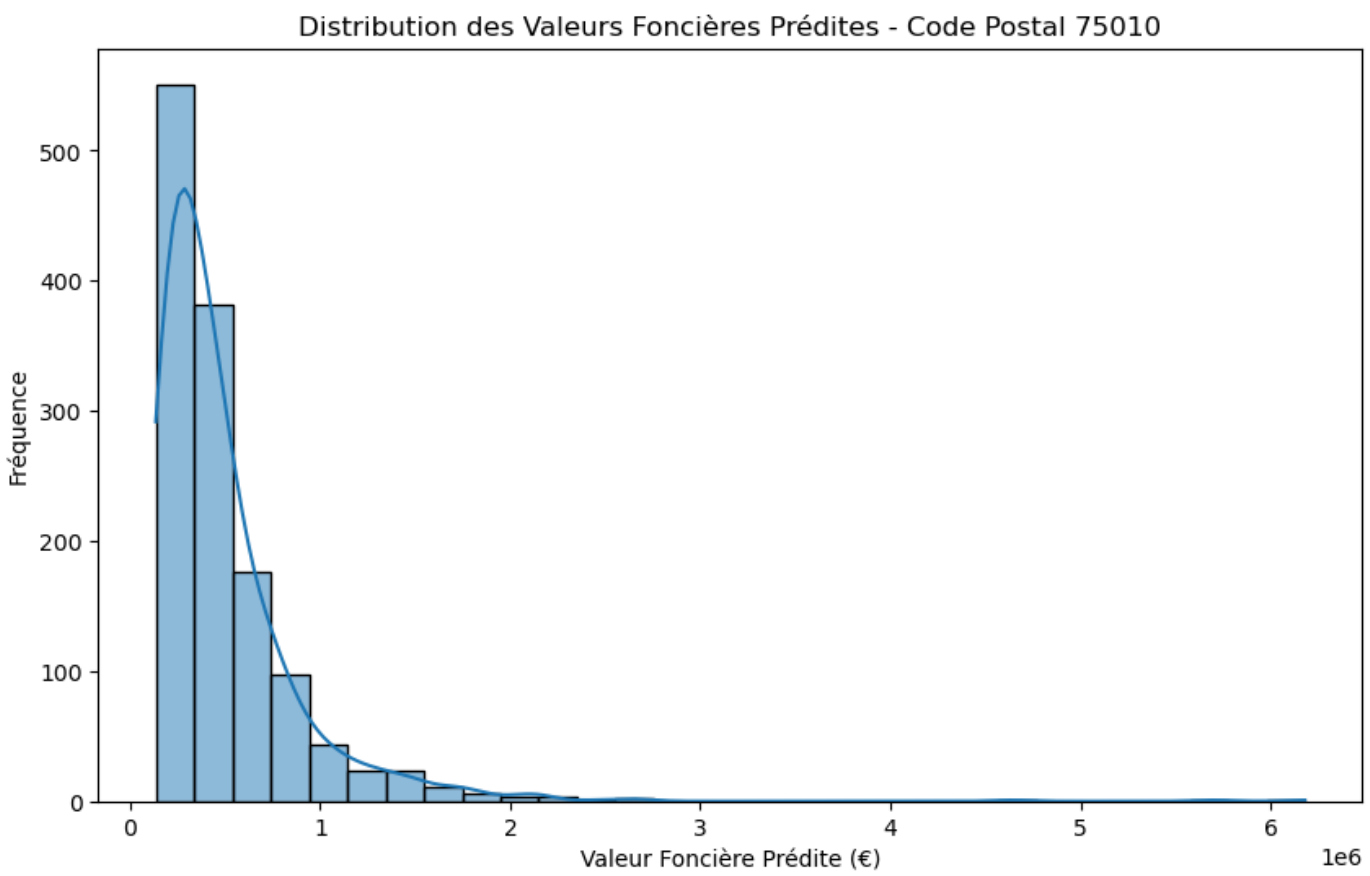


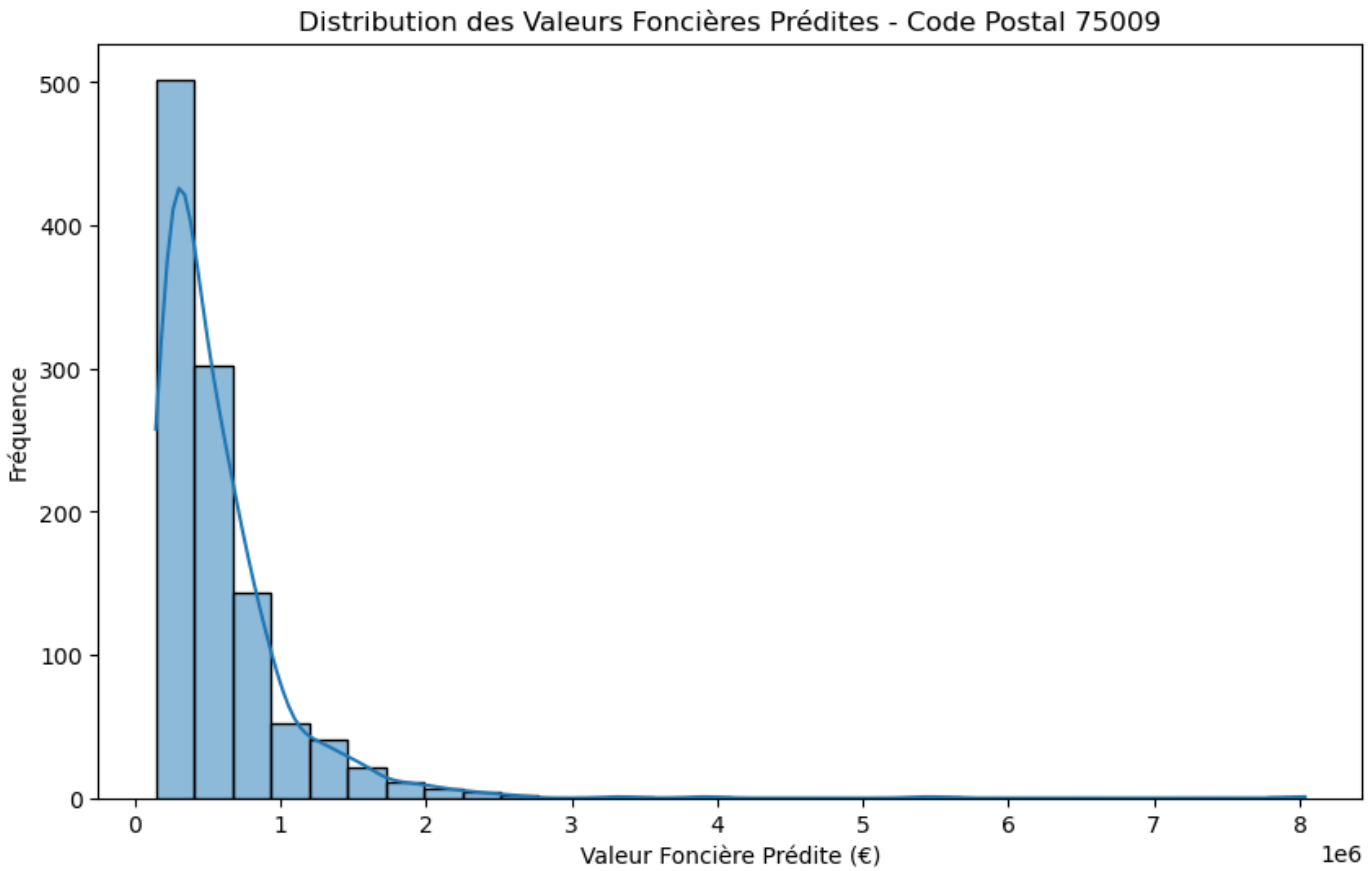


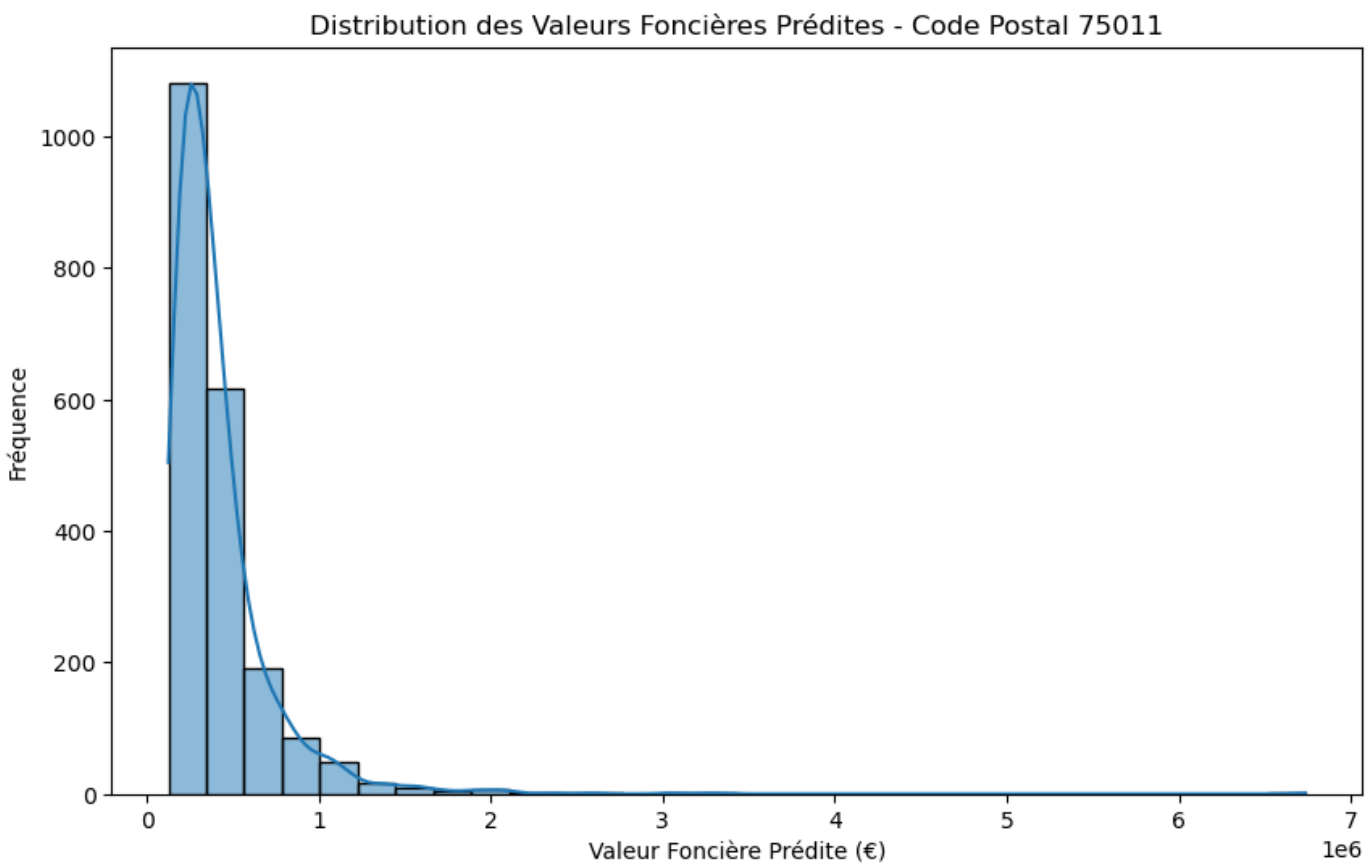


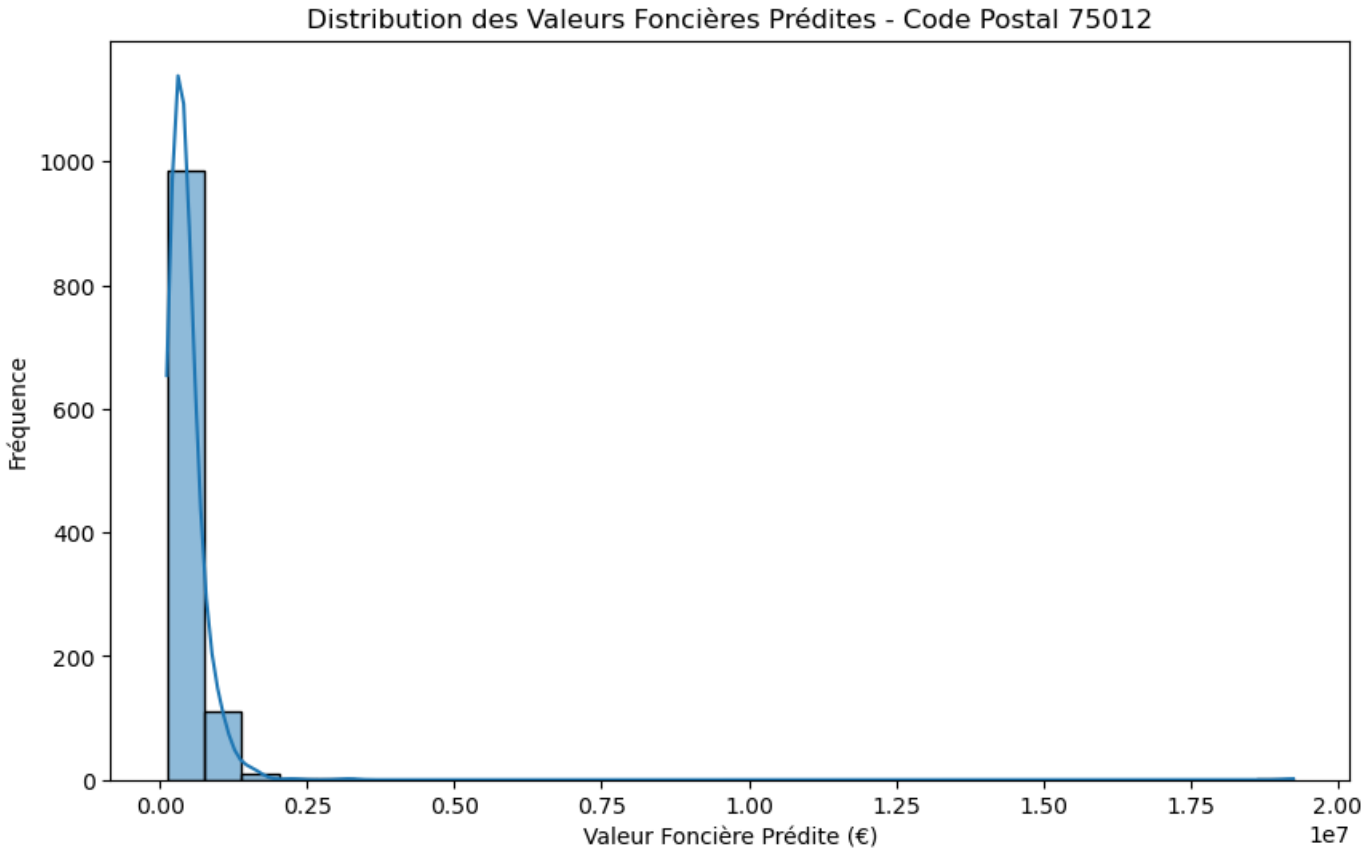


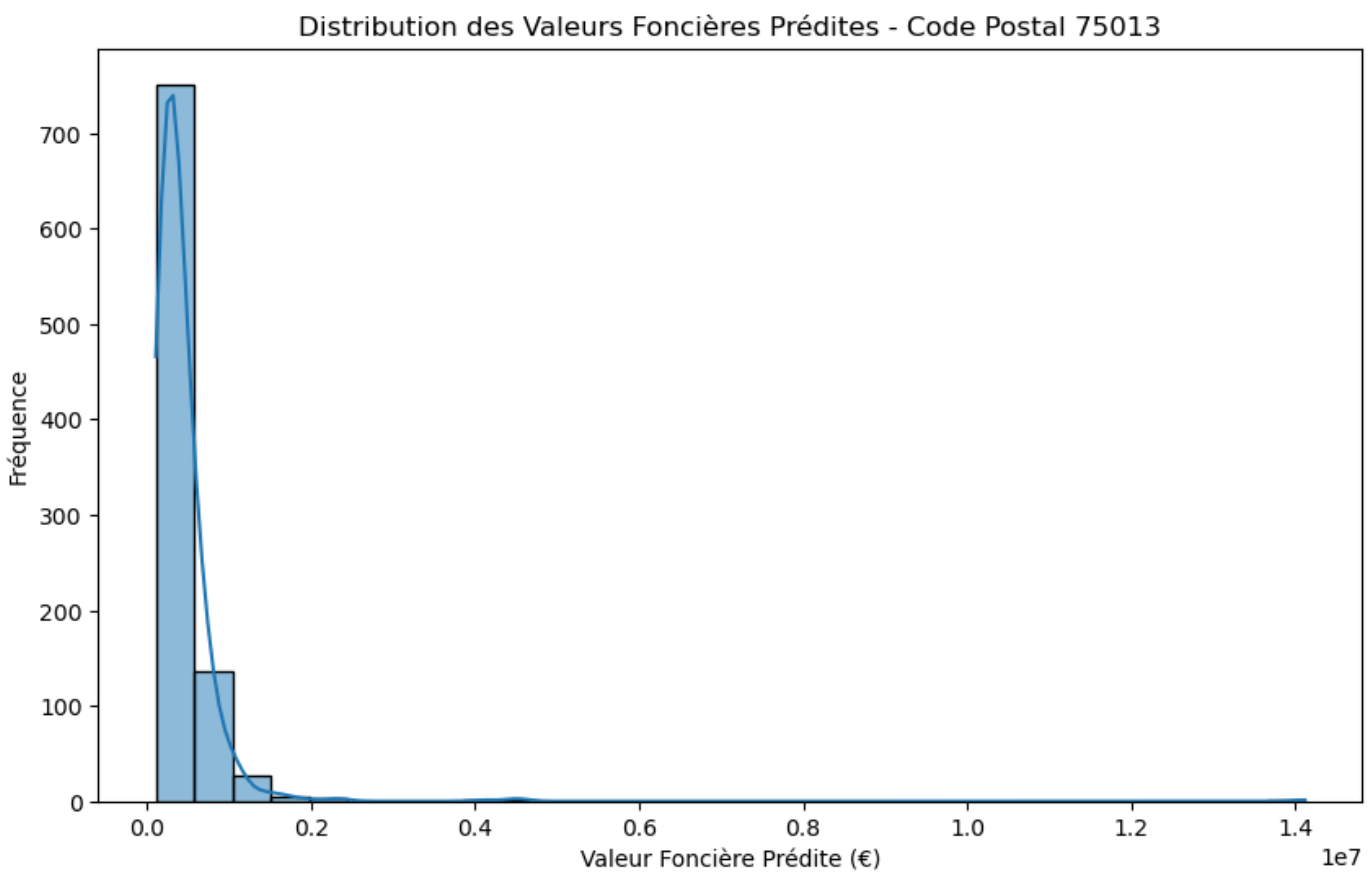


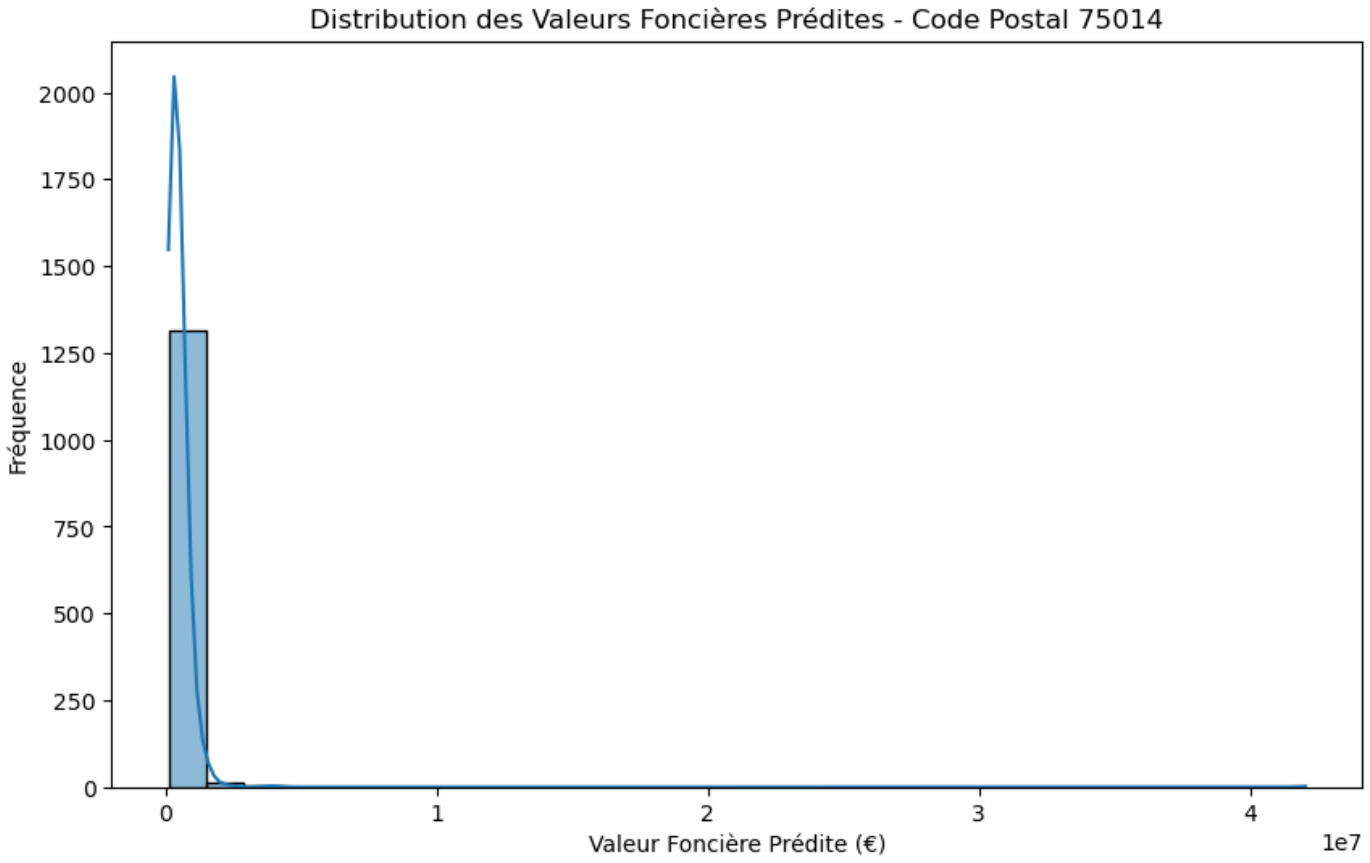


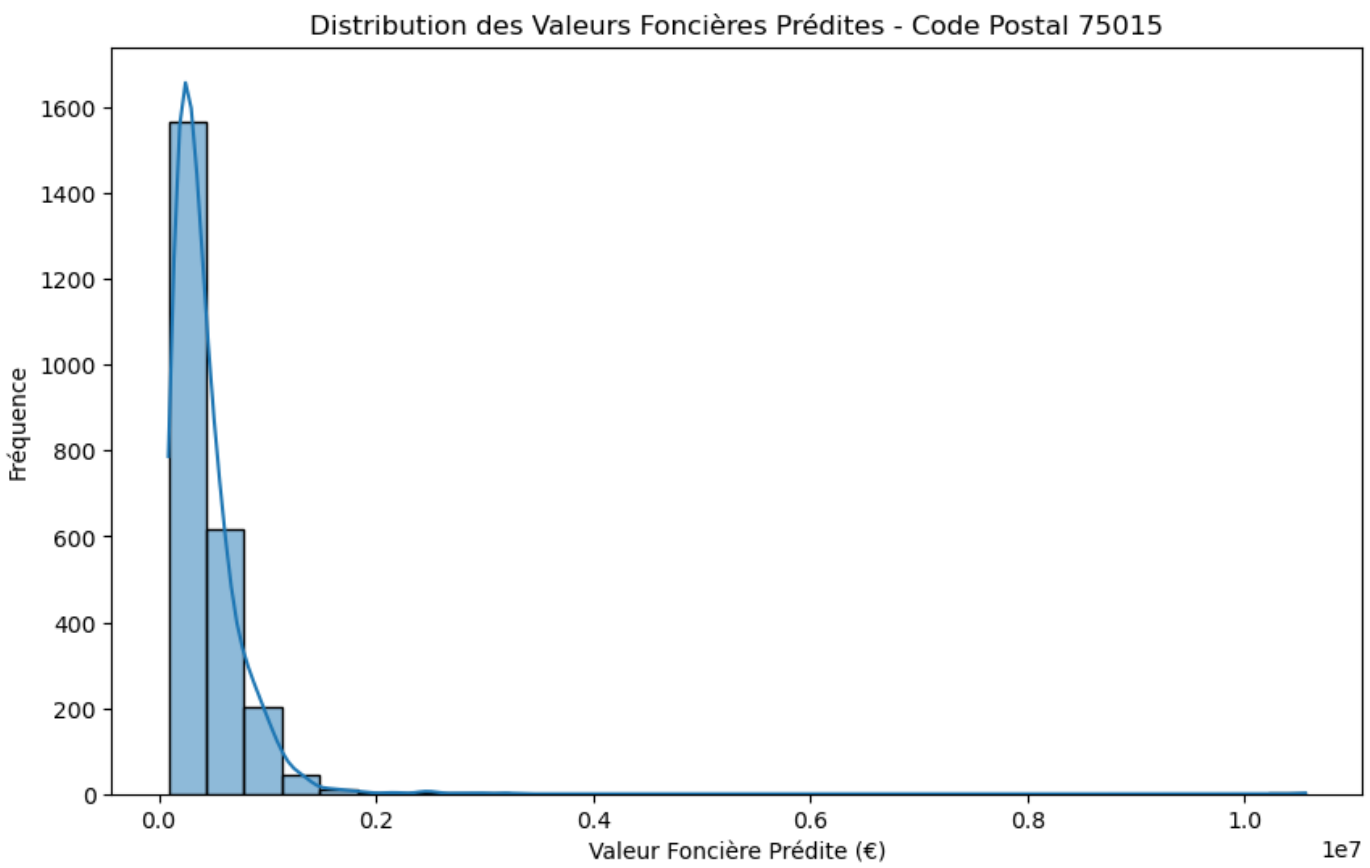


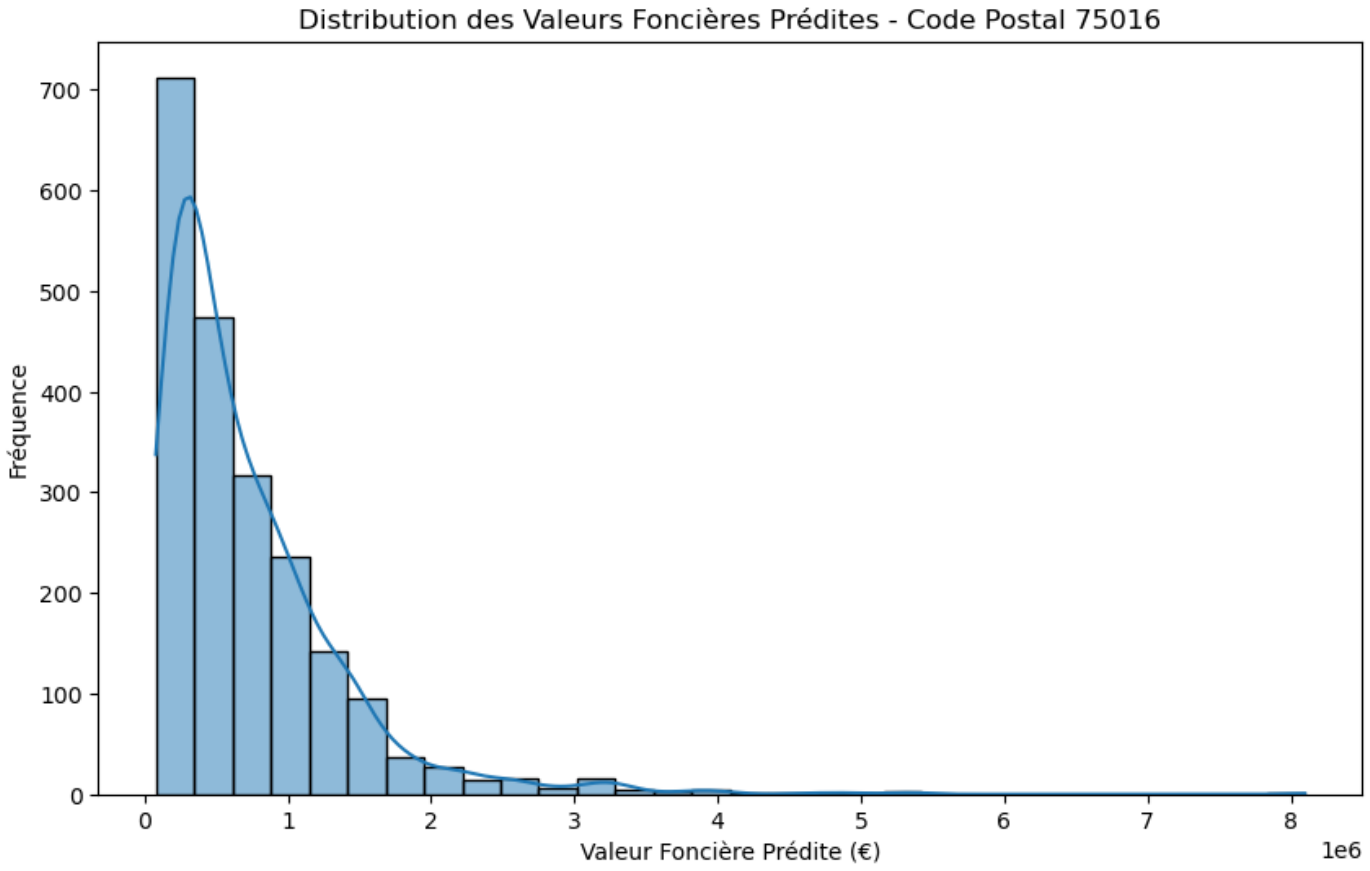


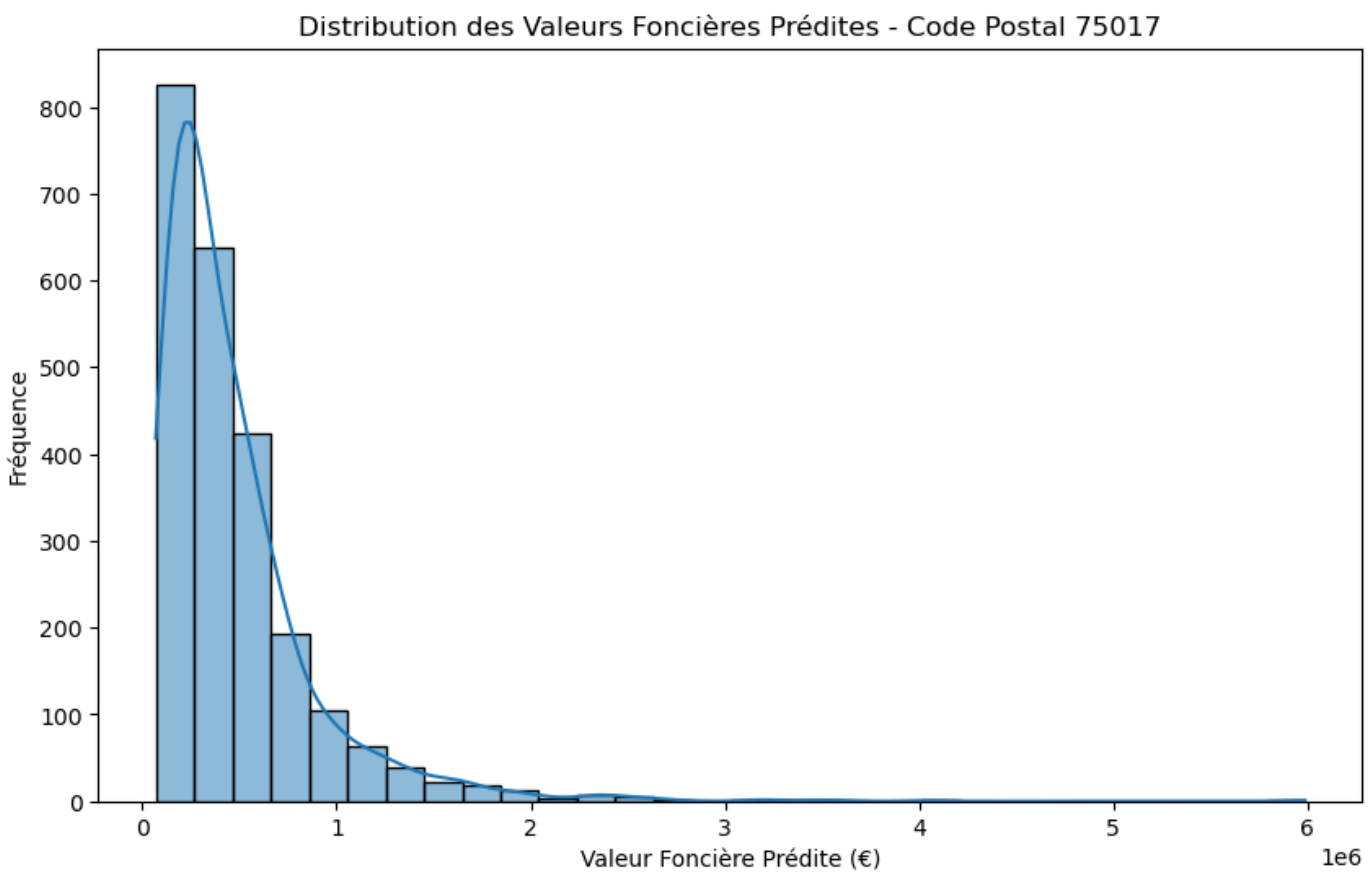


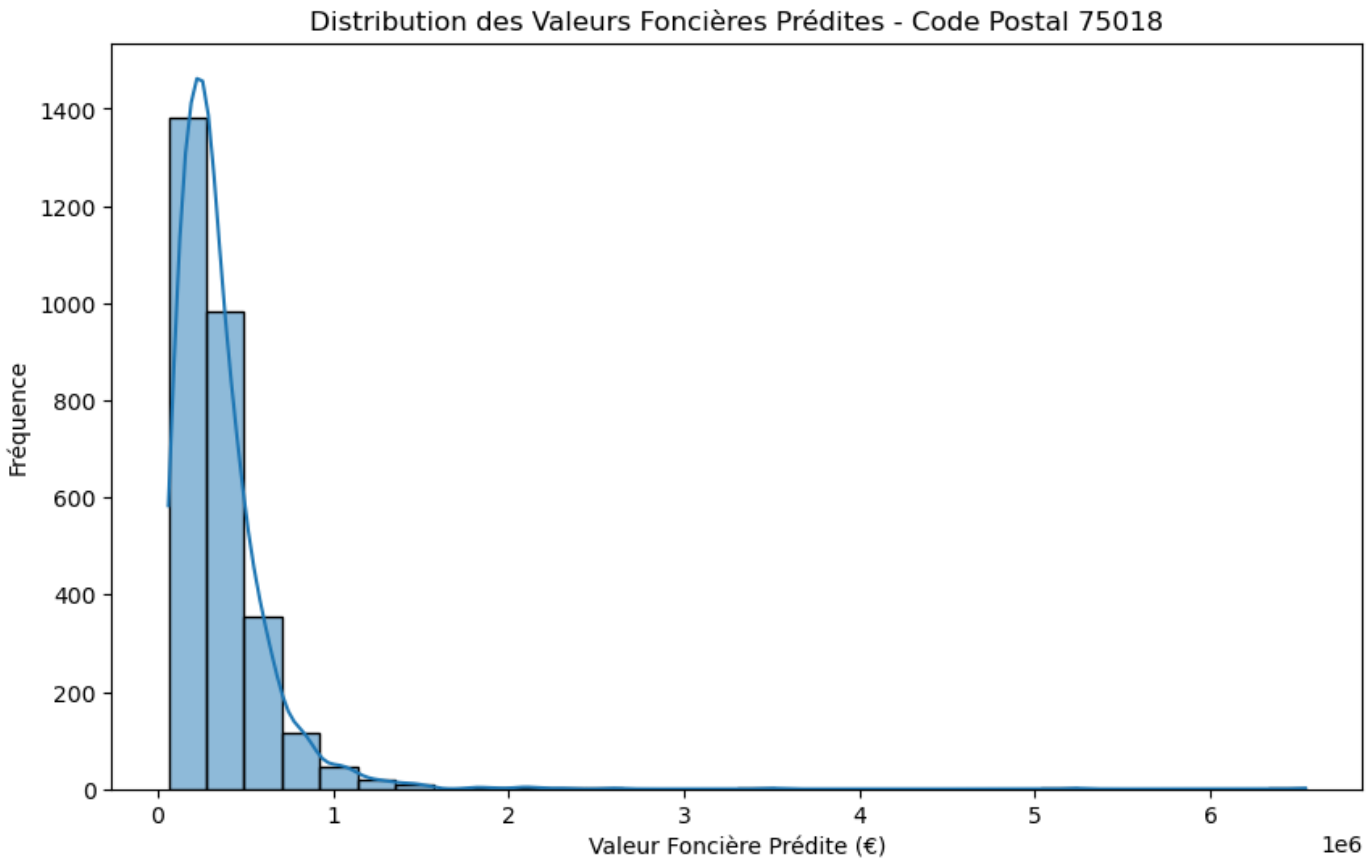


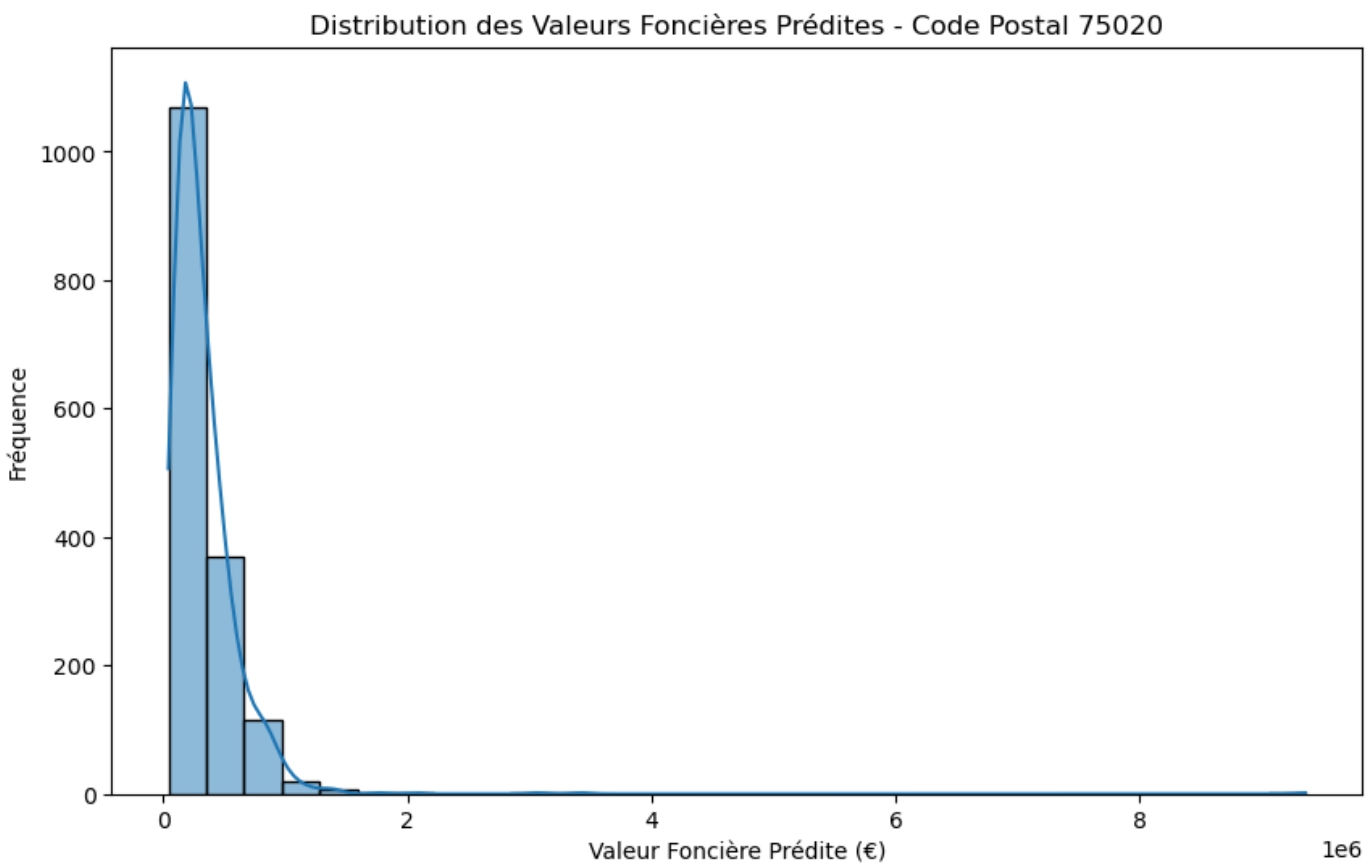


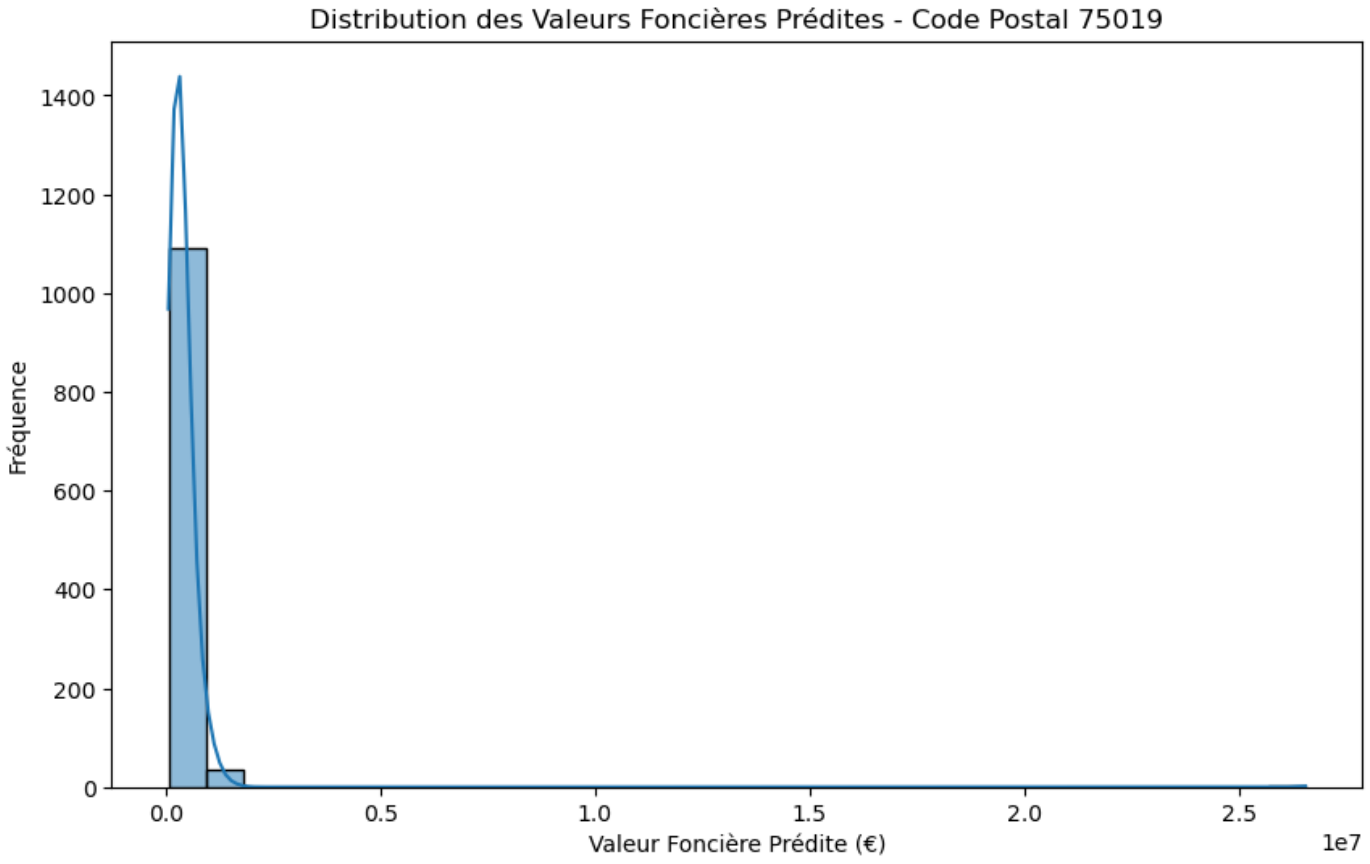


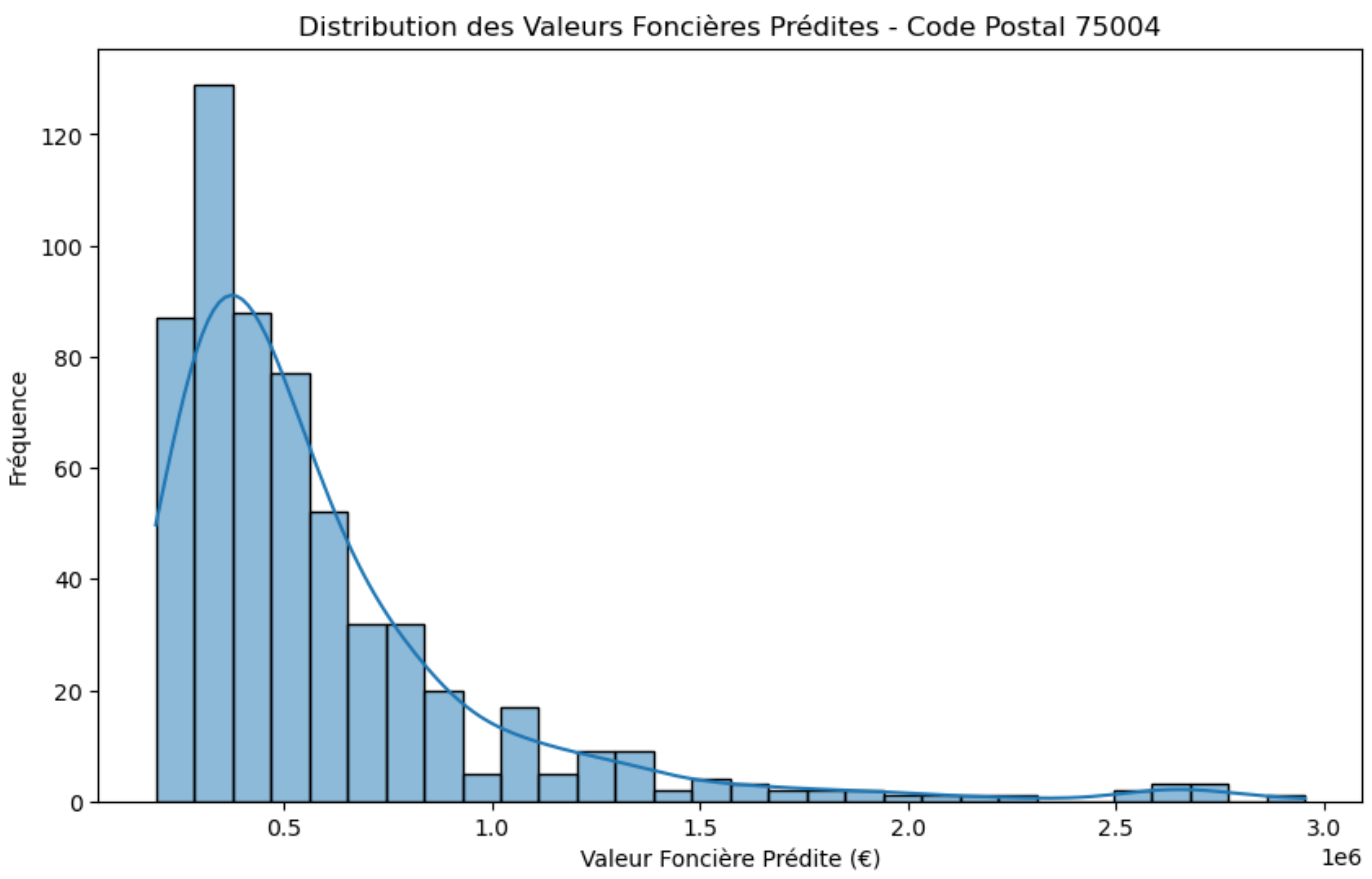


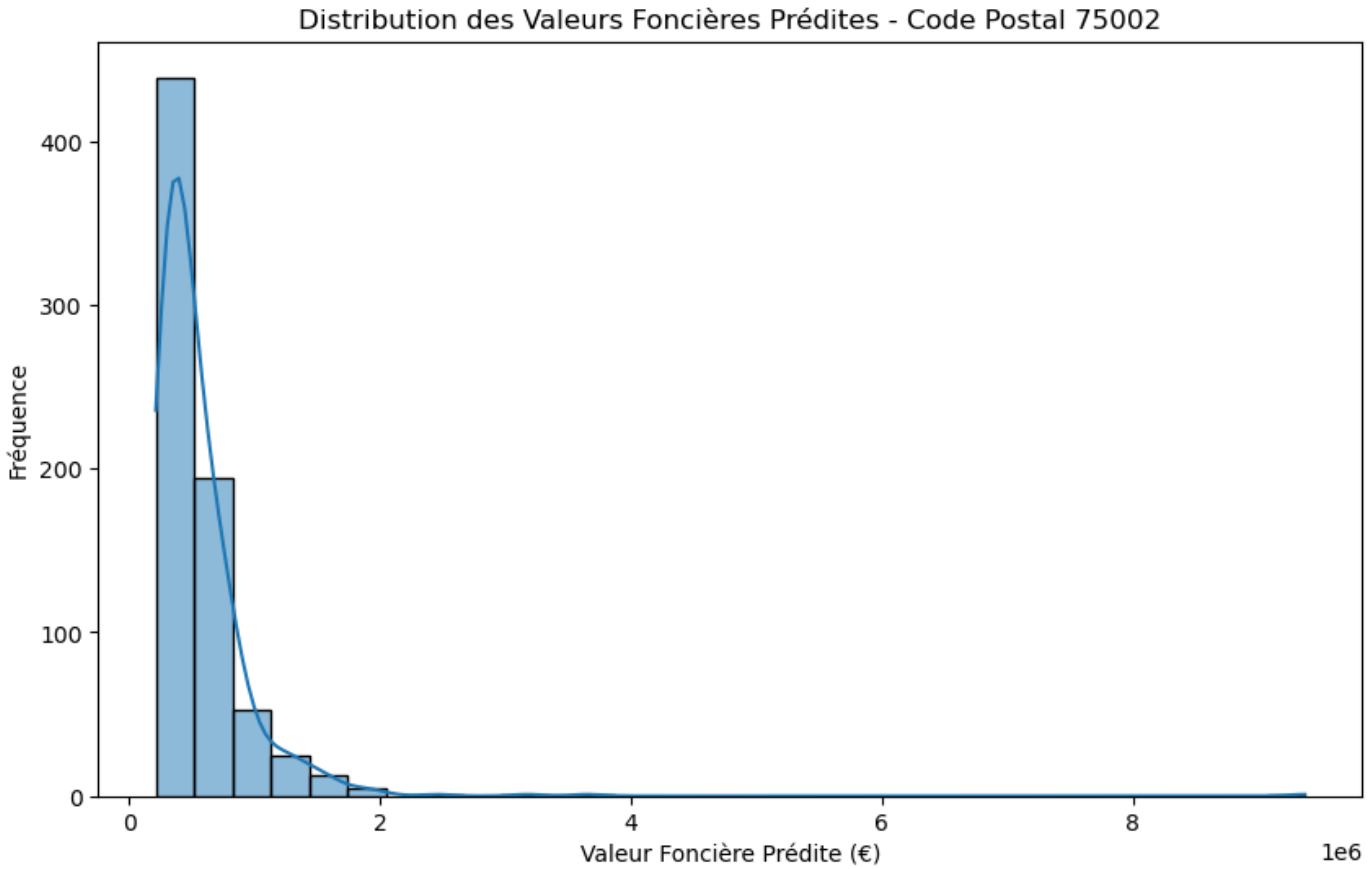












In [16]:
```
# Chargement des données
portfolio_file_path = 'C:\\Users\\pc\\OneDrive\\Documents\\projet n 8\\portefeuille_acti
portfolio_data = pd.read_excel(portfolio_file_path)

# Préparation des données pour la régression
# Utilisation de 'surface_reelle_bati' comme prédicteur
X_portfolio = portfolio_data[['surface_reelle_bati']]  # Prédicteur
y_portfolio = portfolio_data['surface_carrez']         # Cible (ici, surface carrez comme
```

```
# Division des données en ensembles d'entraînement et de test
X_train_portfolio, X_test_portfolio, y_train_portfolio, y_test_portfolio = train_test_sp

# Création et entraînement du modèle de régression linéaire
model_portfolio = LinearRegression()
model_portfolio.fit(X_train_portfolio, y_train_portfolio)

# Prédiction des valeurs pour l'ensemble de test
y_pred_portfolio = model_portfolio.predict(X_test_portfolio)

# Calcul de l'erreur quadratique moyenne et du coefficient de détermination (R²)
mse_portfolio = mean_squared_error(y_test_portfolio, y_pred_portfolio)
r2_portfolio = r2_score(y_test_portfolio, y_pred_portfolio)

# Affichage des résultats
model_coefficient_portfolio = model_portfolio.coef_[0]
model_intercept_portfolio = model_portfolio.intercept_
mse_portfolio, r2_portfolio, model_coefficient_portfolio, model_intercept_portfolio
```

Out[16]:
```
(83.27339232660852, 0.937472428806704, 0.93903241673834, -2.6008966229588637)
```

In [17]:
```
# Affichage des résultats de manière structurée

# Affichage des coefficients du modèle
print("Coefficient de régression (pente) :", model_coefficient_portfolio)
print("Intercept (ordonnée à l'origine) :", model_intercept_portfolio)

# Affichage de la performance du modèle
print("\nPerformance du modèle :")
print("Erreur Quadratique Moyenne (MSE) :", mse_portfolio)
print("Coefficient de Détermination (R²) :", r2_portfolio)
```

```
Coefficient de régression (pente) : 0.93903241673834
Intercept (ordonnée à l'origine) : -2.6008966229588637

Performance du modèle :
Erreur Quadratique Moyenne (MSE) : 83.27339232660852
Coefficient de Détermination (R²) : 0.937472428806704
```

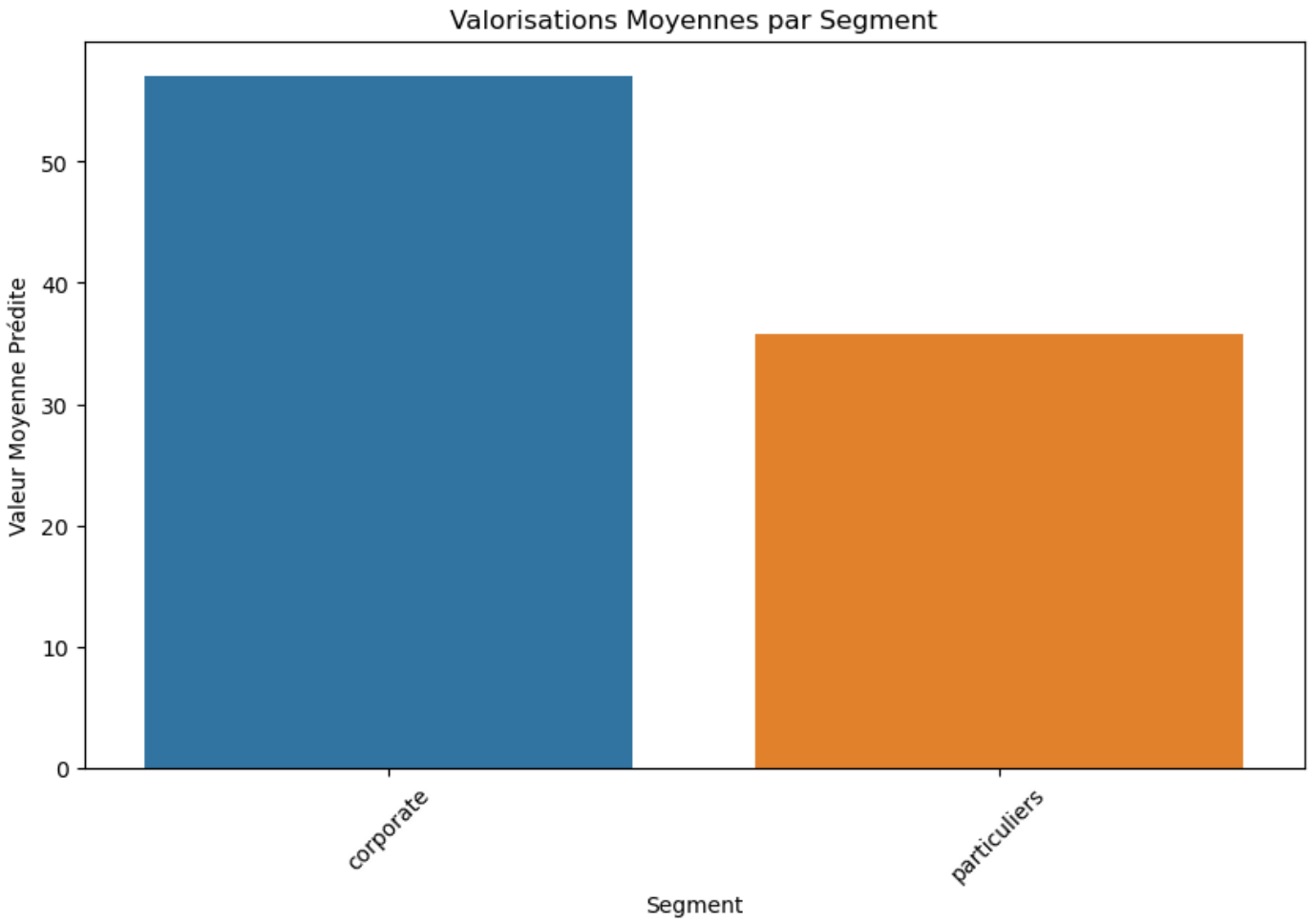
Prédiction de la valorisation future du portefeuille

Utilisation du modèle de régression pour estimer la valeur des biens immobiliers dans le portefeuille de l'entreprise, et analyse comparative entre les segments 'particuliers' et 'corporate'.

In [22]:
```
portefeuille_data['segment'] = portefeuille_data['type_local'].apply(lambda x: 'particul

# Continuation du code pour la prédiction et l'analyse
X_portefeuille = portefeuille_data[['surface_reelle_bati']]
predictions = model_portfolio.predict(X_portefeuille)
portefeuille_data['valeur_predite'] = predictions

# Analyse comparative des valorisations entre les segments
segment_analysis = portefeuille_data.groupby('segment')['valeur_predite'].mean()

# Affichage des résultats
print("Analyse des Valorisation Moyennes par Segment :")
print(segment_analysis)
```

```
Analyse des Valorisation Moyennes par Segment :
segment
corporate       57.124669
particuliers    35.728699
Name: valeur_predite, dtype: float64
```

In [23]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Création d'un graphique à barres pour visualiser les valorisations moyennes par segmen
plt.figure(figsize=(10, 6))
sns.barplot(x=segment_analysis.index, y=segment_analysis.values)

plt.title('Valorisations Moyennes par Segment')
plt.xlabel('Segment')
plt.ylabel('Valeur Moyenne Prédite')
plt.xticks(rotation=45)  # Rotation des étiquettes pour une meilleure lisibilité
plt.show()
```



Classification non supervisée des biens immobiliers

Mise en œuvre d'une méthode de classification non supervisée pour regrouper les biens immobiliers et comparaison avec le classement existant.

In [20]:
```python
# Importation de l'algorithme de clustering
from sklearn.cluster import KMeans

# Préparation des données pour la classification
# Sélection des variables pertinentes (par exemple, surface, localisation)
classification_data = portefeuille_data[['surface_reelle_bati', 'code_postal']]
```

```
# Application de l'algorithme de clustering K-means
kmeans = KMeans(n_clusters=3, random_state=0)  # Le nombre de clusters est à choisir
portefeuille_data['cluster'] = kmeans.fit_predict(classification_data)

# Analyse des clusters créés
# (Exemple de visualisation et d'analyse, à adapter en fonction des besoins spécifiques)
cluster_analysis = portefeuille_data.groupby('cluster')['surface_reelle_bati', 'code_pos
cluster_analysis
```

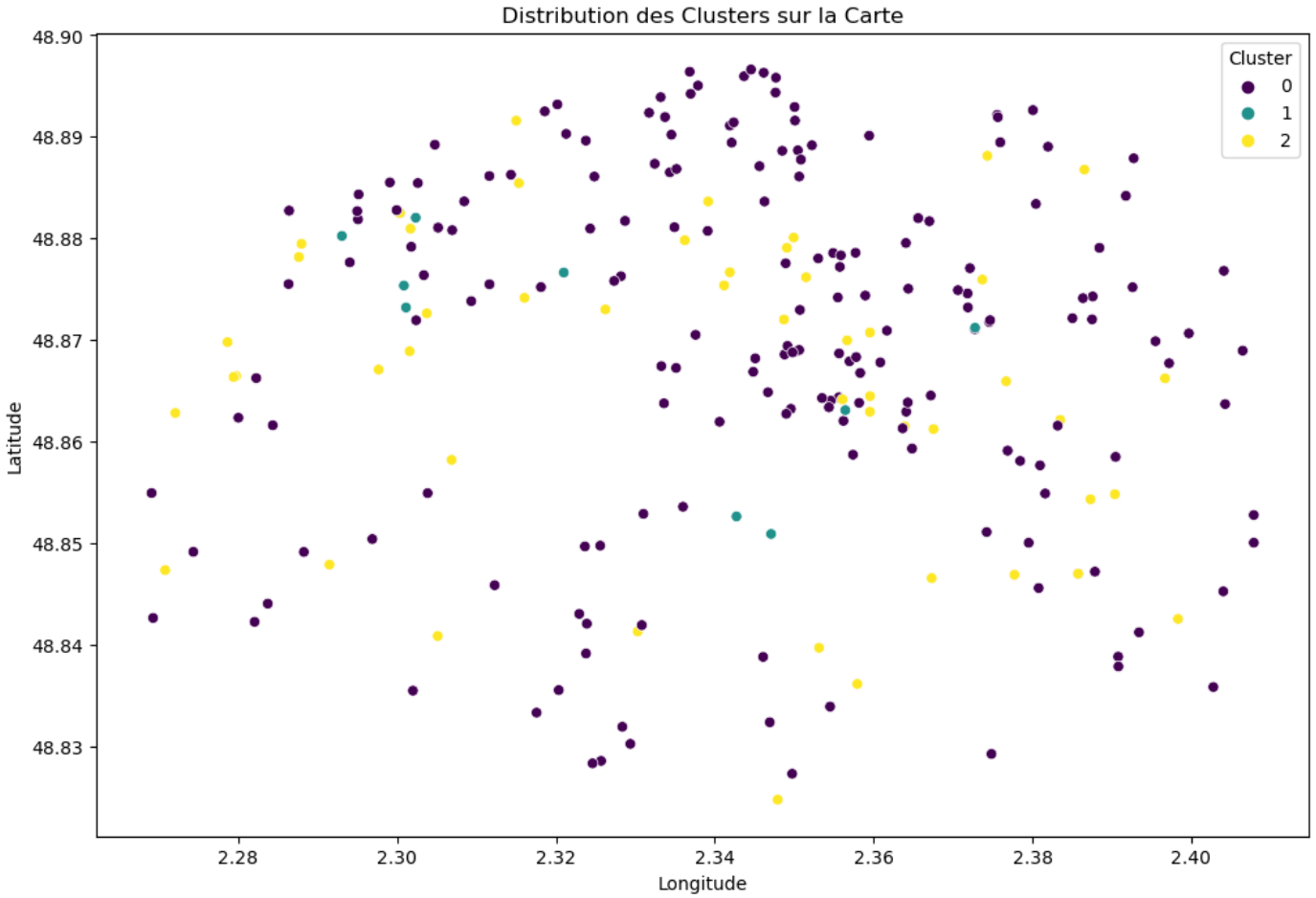
```
C:\Users\pc\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning:
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_
init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\pc\AppData\Local\Temp\ipykernel_18124\2268591799.py:14: FutureWarning: Indexing
with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a l
ist instead.
  cluster_analysis = portefeuille_data.groupby('cluster')['surface_reelle_bati', 'code_p
ostal'].mean()
```

Out[20]:

cluster	surface_reelle_bati	code_postal
0	33.597156	75012.611374
1	207.777778	75009.111111
2	91.327273	75012.254545

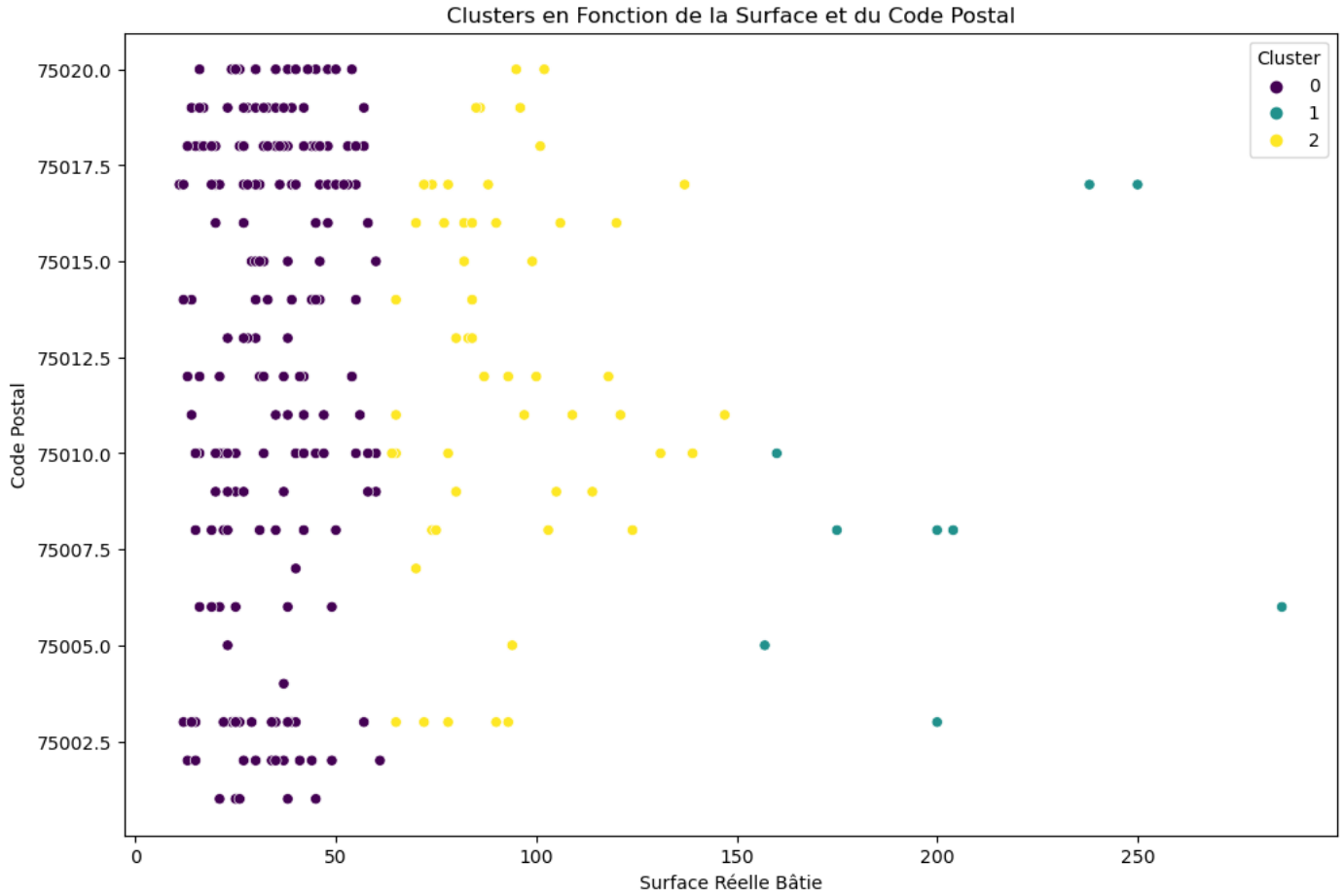
In [24]:

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

# Vérification de DataFrame 'portefeuille_data' contient des colonnes 'longitude' et 'la
plt.figure(figsize=(12, 8))

# Scatter plot avec longitude et latitude, coloré par cluster
sns.scatterplot(x='longitude', y='latitude', hue='cluster', palette='viridis', data=port

plt.title('Distribution des Clusters sur la Carte')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend(title='Cluster')
plt.show()
```



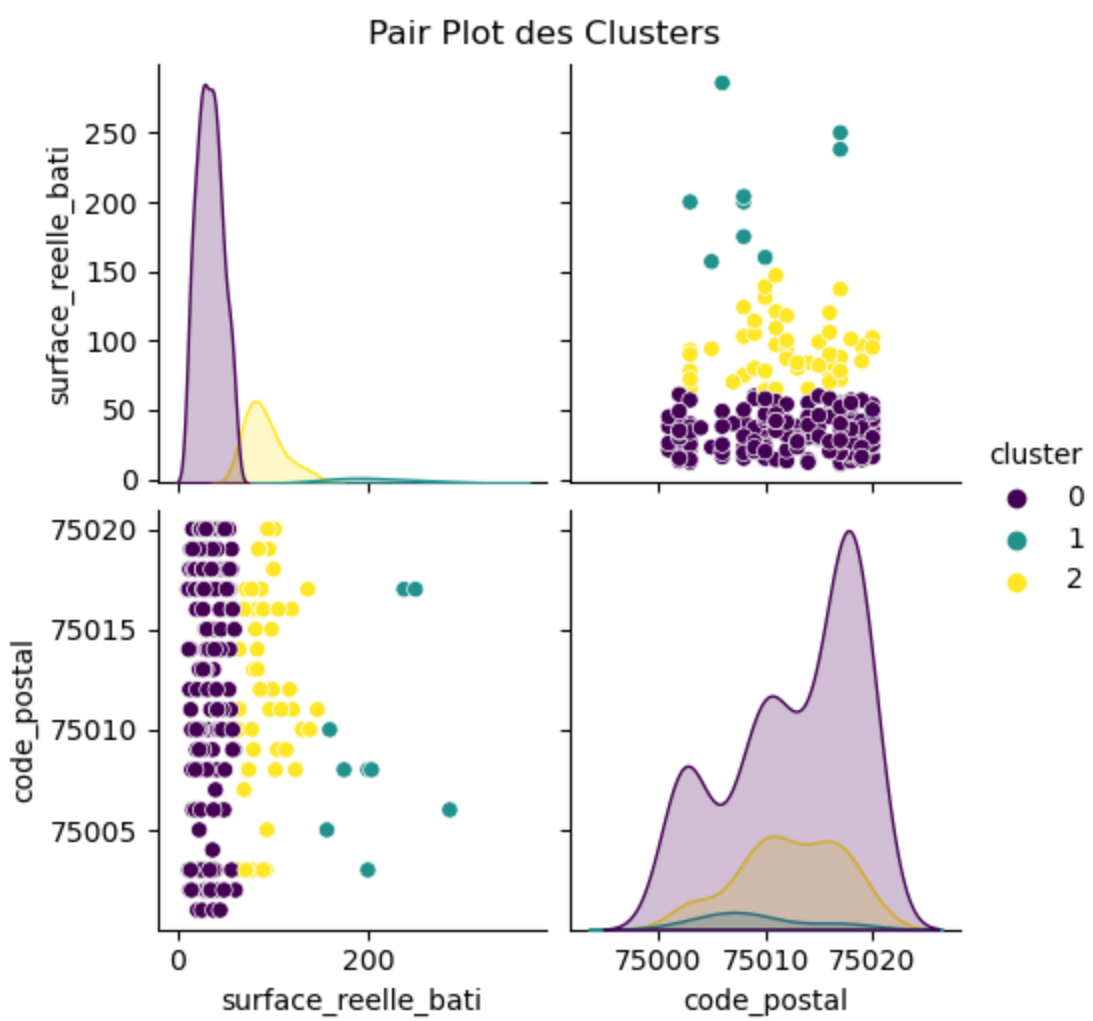
In [25]:

```
plt.figure(figsize=(12, 8))

# Scatter plot avec surface_reelle_bati et code_postal
sns.scatterplot(x='surface_reelle_bati', y='code_postal', hue='cluster', palette='viridi

plt.title('Clusters en Fonction de la Surface et du Code Postal')
plt.xlabel('Surface Réelle Bâtie')
plt.ylabel('Code Postal')
plt.legend(title='Cluster')
plt.show()
```



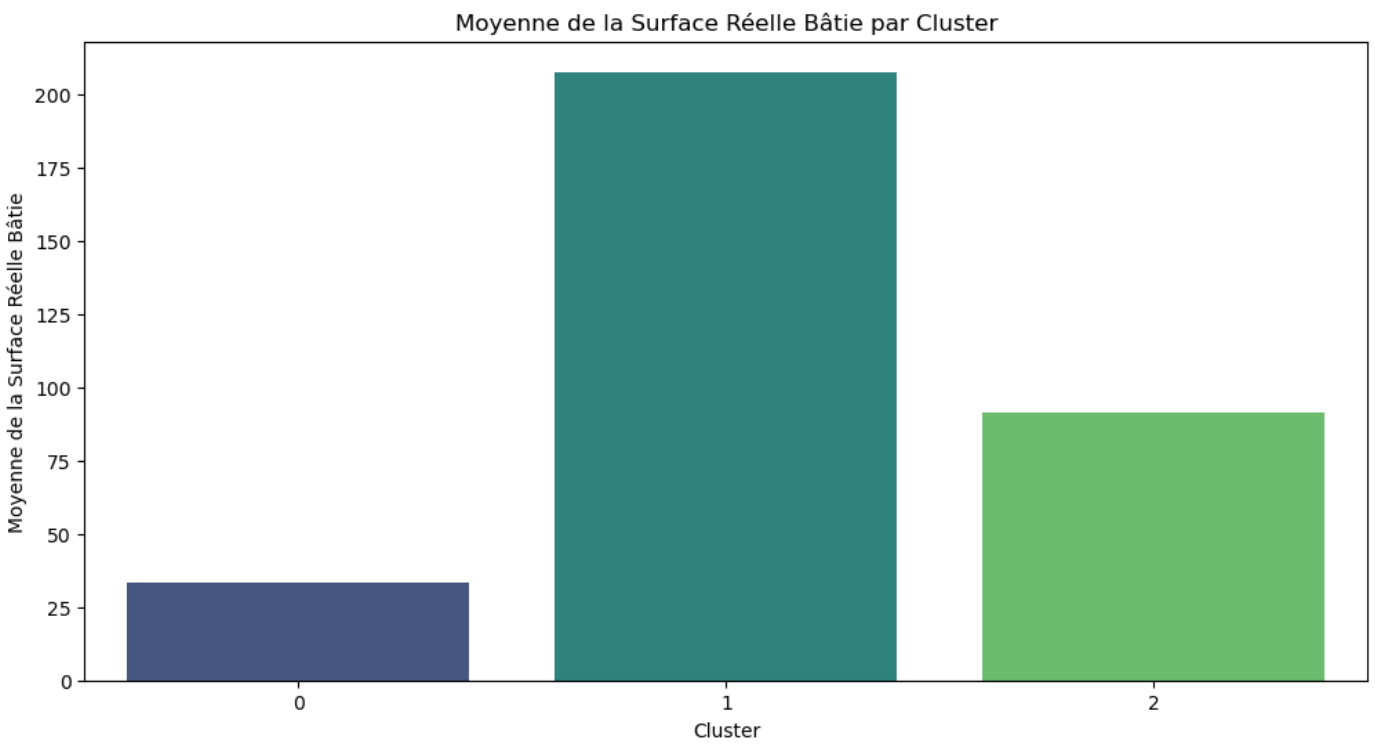
In [26]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Sélection de quelques variables pour le pair plot
selected_variables = ['surface_reelle_bati', 'code_postal', 'cluster']
pair_plot_data = portefeuille_data[selected_variables]

# Création d'un pair plot
sns.pairplot(pair_plot_data, hue='cluster', palette='viridis')
plt.suptitle('Pair Plot des Clusters', y=1.02)  # Ajout d'un titre et ajustement de la p
```

Out[26]:
```
Text(0.5, 1.02, 'Pair Plot des Clusters')
```



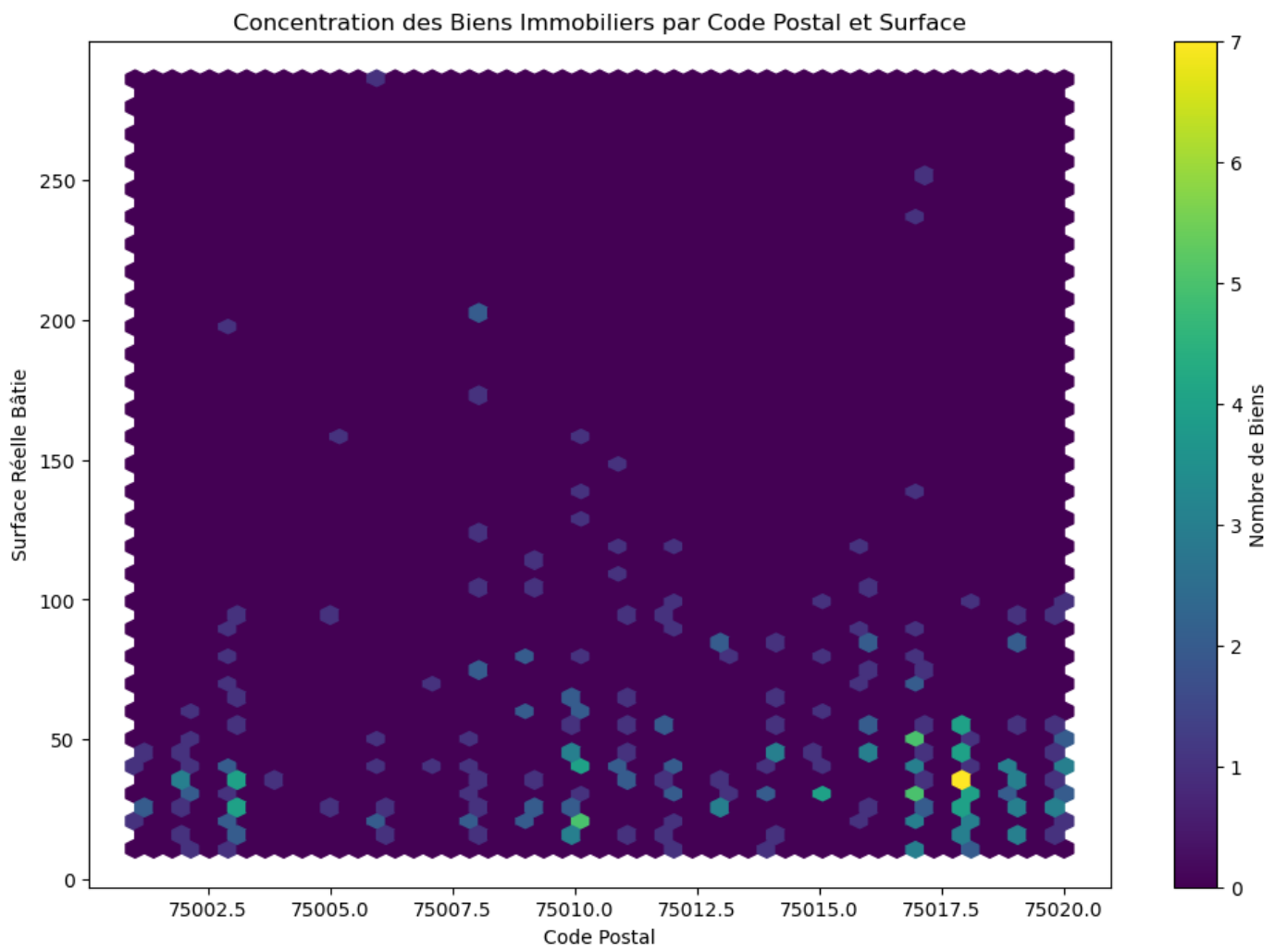
In [28]:

```
# Calcul de la moyenne de la surface réelle bâtie pour chaque cluster
mean_surface_by_cluster = portefeuille_data.groupby('cluster')['surface_reelle_bati'].me

plt.figure(figsize=(12, 6))
sns.barplot(x='cluster', y='surface_reelle_bati', data=mean_surface_by_cluster, palette=
plt.title('Moyenne de la Surface Réelle Bâtie par Cluster')
plt.xlabel('Cluster')
plt.ylabel('Moyenne de la Surface Réelle Bâtie')
plt.show()
```



In [29]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
plt.hexbin(portefeuille_data['code_postal'], portefeuille_data['surface_reelle_bati'], g
plt.colorbar(label='Nombre de Biens')
plt.title('Concentration des Biens Immobiliers par Code Postal et Surface')
plt.xlabel('Code Postal')
plt.ylabel('Surface Réelle Bâtie')
plt.show()
```



In [30]:
```python
import seaborn as sns

# Calcul de la matrice de corrélation
correlation_matrix = portefeuille_data[['code_postal', 'surface_reelle_bati', 'cluster']]

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='viridis')
plt.title('Heatmap de Corrélation entre Code Postal, Surface et Clusters')
plt.show()
```