# I. Library import

In [1]:
```
#Import the librairies
import ipywidgets as widgets
from ipywidgets import interact, interactive,HBox, VBox
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import geopandas as gpd
import geodatasets
import matplotlib.pyplot as plt
import geopandas as gpd
import pandas as pd
import folium
import matplotlib
import mapclassify
from folium.plugins import MarkerCluster
import os
```

```
c:\Users\maube\anaconda3\Lib\site-packages\paramiko\transport.py:219: CryptographyDeprecationWarning: Blowfish has been deprecated and will be removed in a future release
  "class": algorithms.Blowfish,
```

In [2]:
```
#Read the data from 2024 and 2023 and concat them
data2 = pd.read_csv("data.csv",low_memory=False)
data1 = pd.read_csv("data2023.csv",low_memory=False)
data = pd.concat([data1, data2], ignore_index=True)
```

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1584106 entries, 0 to 1584105
Data columns (total 18 columns):
 #   Column                     Non-Null Count    Dtype  
---  ------                     --------------    -----  
 0   date_mutation              1584106 non-null  object 
 1   nature_mutation            1584106 non-null  object 
 2   valeur_fonciere            1584106 non-null  float64
 3   code_commune               1584106 non-null  object 
 4   code_departement           1584106 non-null  object 
 5   id_parcelle                1584106 non-null  object 
 6   nombre_lots                1584106 non-null  int64  
 7   code_type_local            1584106 non-null  float64
 8   nombre_pieces_principales  1584106 non-null  float64
 9   surface_terrain            1584106 non-null  float64
 10  longitude                  1584106 non-null  float64
 11  latitude                   1584106 non-null  float64
 12  tranche_valeur             1584106 non-null  object 
 13  code_insee                 1584106 non-null  int64  
 14  region_name                1584106 non-null  object 
 15  plage_surface              1584106 non-null  object 
 16  prix_m2                    1584106 non-null  float64
 17  mois_annee                 1584106 non-null  object 
dtypes: float64(7), int64(2), object(9)
memory usage: 217.5+ MB
```

## II. Création des fonctions

In [4]:
```
def selection_filters_and_analyses(df):
    # Presentation of the filters
    type_local_map = {
        1: 'House',
        2: 'Apartment',
        3: 'Dependency (Isolated)',
        4: 'Industrial and commercial premises or similar'
    }
    df['type_local'] = df['code_type_local'].map(type_local_map)

    departments = df['code_departement'].unique()
    types_properties = df['type_local'].unique()

    # Widgets for the filters
    department_select = widgets.SelectMultiple(
        options=departments,
        value=[],
        description='Departments',
        layout=widgets.Layout(width='60%')
    )
    department_select.style.description_width = '100px'

    types_properties = df['type_local'].unique()
    types_properties_select = widgets.SelectMultiple(
        options=types_properties,
        value=[],
        description='Types of property',
        layout=widgets.Layout(width='60%')
    )
    types_properties_select.style.description_width = '100px'

    price_range = ["< 50k €", "50k - 100k €", "100k - 150k €", "150k - 200k €",

    price_range_select = widgets.SelectMultiple(
        options=price_range,
        value=[],
        description='Price range',
        layout=widgets.Layout(width='60%')
    )
    price_range_select.style.description_width = '100px'

    slice_surface = ['0-50 m²', '50-100 m²', '100-150 m²', '150-200 m²', '200-25
    slice_surface_select = widgets.SelectMultiple(
        options=slice_surface,
        value=[],
        description='Surface slice',
        layout=widgets.Layout(width='60%')
    )
    slice_surface_select.style.description_width = '100px'

    # Widget to sekect the analyse to execute
    disponible_analyses = {
        "Average price per m²": show_average_price_per_m2,
        "Average surface area" : show_average_surface_per_dept,
        "Price evolution" : evolution_average_price_per_m2_per_departement_month
        "Number of main parts" :display_number_of_room_department,
```

```
        "Type of property by department" : distribution_type_property_by_departm
        "Distribution of land values" : distribution_of_land_value_by_tranche,
        "Price per m² depending on the surface area" : price_m2_per_surface,
        "Explore the map" : explore_france

    }

    analyse_select = widgets.SelectMultiple(
        options=list(disponible_analyses.keys()),
        description='Analyse',
        layout=widgets.Layout(width='60%')
    )

    analyse_select.style.description_width = '100px'

    # Button to apply the filters
    bouton_apply = widgets.Button(description="Apply the filters", button_style=

    # Function to take the filters and execute the analyse
    def apply_filter(_):
        selected_analyses = list(analyse_select.value)

        # We execute each analyse selected
        for analyse in selected_analyses:
            func = disponible_analyses[analyse]
            func(df, department_select, types_properties_select,price_range_sele

    # Link the button to the function
    bouton_apply.on_click(apply_filter)

    # Show widgets
    display(department_select, types_properties_select,price_range_select,slice_
```

In [5]:

```
def show_average_price_per_m2(df,department_select,types_properties_select,price

    # We take the department,price range, slice surface and types of properties
    selected_departments = list(department_select.value)
    selected_types_properties = list(types_properties_select.value)
    selected_price_range = list(price_range_select.value)
    selected_slice_surface = list(slice_surface_select.value)

    # If nothing is selected we take all the values for each point
    if not selected_departments:
        selected_departments = list(df['code_departement'].unique())
    if not selected_types_properties:
        selected_types_properties = list(df['type_local'].unique())
    if not selected_price_range:
        selected_price_range = ["< 50k €", "50k - 100k €", "100k - 150k €", "150
    if not selected_slice_surface:
        selected_slice_surface = ['0-50 m²', '50-100 m²', '100-150 m²', '150-200

    # Filter data based with our filter
    df_filtered = df[(df['code_departement'].isin(selected_departments)) &
                        (df['type_local'].isin(selected_types_properties))&
                        (df['tranche_valeur'].isin(selected_price_range))&
                        (df['plage_surface'].isin(selected_slice_surface))].copy

    df_filtered_national = df[
                        (df['type_local'].isin(selected_types_properties))&
                        (df['tranche_valeur'].isin(selected_price_range))&
```

```
                        (df['plage_surface'].isin(selected_slice_surface))].copy

    # Calculate the average price per m² by department
    average_price_par_dept = df_filtered.groupby('code_departement')['prix_m2'].

    # Calculate the average price of the selected departments
    average_price_select = average_price_par_dept.mean()

    # Calculate the national average price per m²
    national_average_price = df_filtered_national[df_filtered_national['type_loc

    # we create the graph
    fig, ax = plt.subplots(figsize=(20, 6))

    # We draw a bar to show the average price for each department
    ax.bar(average_price_par_dept.index, average_price_par_dept)

    # Added a bar for the average price selected
    ax.bar('Selected Average', average_price_select, alpha=0.7)

    # Added a bar for the national average price
    ax.bar('National Average', national_average_price, alpha=0.7)

    # And this is the parameters of the graphics
    ax.set_ylabel('Average price per m² (€)')
    ax.set_xlabel('Departments')
    ax.set_title("Average price per m² per department, selected and glogal avera
    ax.grid(True, linestyle='--', alpha=0.6)
    plt.xticks(rotation=90)
    plt.tight_layout()  # To avoid labels overlapping
    plt.show()
```

In [6]:

```
def show_average_surface_per_dept(df,department_select,types_properties_select,p

    # We take the department,price range, slice surface and types of properties
    selected_departments = list(department_select.value)
    selected_types_properties = list(types_properties_select.value)
    selected_price_range = list(price_range_select.value)
    selected_slice_surface = list(slice_surface_select.value)

    # If nothing is selected we take all the values for each point
    if not selected_departments:
        selected_departments = list(df['code_departement'].unique())
    if not selected_types_properties:
        selected_types_properties = list(df['type_local'].unique())
    if not selected_price_range:
        selected_price_range = ["< 50k €", "50k - 100k €", "100k - 150k €", "150
    if not selected_slice_surface:
        selected_slice_surface = ['0-50 m²', '50-100 m²', '100-150 m²', '150-200

    # Filter data based with our filter
    df_filtered = df[(df['code_departement'].isin(selected_departments)) &
                        (df['type_local'].isin(selected_types_properties))&
                        (df['tranche_valeur'].isin(selected_price_range))&
                        (df['plage_surface'].isin(selected_slice_surface))].copy

    df_filtered_national = df[
                        (df['type_local'].isin(selected_types_properties))&
                        (df['tranche_valeur'].isin(selected_price_range))&
                        (df['plage_surface'].isin(selected_slice_surface))].copy
```

```
    # We calculate the average surface area for each selected department
    average_area_par_dept = df_filtered.groupby('code_departement')['surface_ter

    # Calculate the average surface area of the selected departments
    surface_moyenne_selectionnee = average_area_par_dept.mean()

    # Calculate the national average surface area (average of all selected depar
    national_average_area = df_filtered_national[df_filtered_national['type_loca

    fig, ax = plt.subplots(figsize=(20, 6))

    # We plot the average area for each selected department
    ax.bar(average_area_par_dept.index, average_area_par_dept, label='Average Su

    # Add a bar for the average area of the selected departments
    ax.bar('Selected Average', surface_moyenne_selectionnee, label='Selected Ave

    # Add a bar for the national average area
    ax.bar('National average', national_average_area, label='National average',

    # parameters of the graphics
    ax.set_ylabel('Average surface area in m²')
    ax.set_xlabel('Departments')
    ax.set_title("Average surface area in m² by department, selection and global
    ax.grid(True, linestyle='--', alpha=0.6)
    ax.legend(title="Legend", bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()
```

In [7]:

```
def evolution_average_price_per_m2_per_departement_monthly(df,department_select,
    # We take the department,price range, slice surface and types of properties
    selected_departments = list(department_select.value)
    selected_types_properties = list(types_properties_select.value)
    selected_price_range = list(price_range_select.value)
    selected_slice_surface = list(slice_surface_select.value)

    # If nothing is selected we take all the values for each point
    if not selected_departments:
        selected_departments = list(df['code_departement'].unique())
    if not selected_types_properties:
        selected_types_properties = list(df['type_local'].unique())
    if not selected_price_range:
        selected_price_range = ["< 50k €", "50k - 100k €", "100k - 150k €", "150
    if not selected_slice_surface:
        selected_slice_surface = ['0-50 m²', '50-100 m²', '100-150 m²', '150-200

   # Filter data based with our filter
    df_filtered = df[(df['code_departement'].isin(selected_departments)) &
                     (df['type_local'].isin(selected_types_properties))&
                     (df['tranche_valeur'].isin(selected_price_range))&
                     (df['plage_surface'].isin(selected_slice_surface))].copy

    df_filtered_national = df[
                     (df['type_local'].isin(selected_types_properties))&
                     (df['tranche_valeur'].isin(selected_price_range))&
                     (df['plage_surface'].isin(selected_slice_surface))].copy
```

```
    # we verify that the date column is in the correct format
    df_filtered['date_mutation'] = pd.to_datetime(df_filtered['date_mutation'])

    # we take the average price per department for each month
    average_price_par_dept_mois = df_filtered.groupby(['code_departement', 'mois


    plt.figure(figsize=(15, 8))

    # Add the curve for the global average
    df_dept_global = df_filtered.groupby('mois_annee')['prix_m2'].mean().reset_i
    plt.plot(df_dept_global['mois_annee'].astype(str), df_dept_global['prix_m2']

    # Add the curve for the national average (based on the entire data)
    df_national_global = df_filtered_national.groupby('mois_annee')['prix_m2'].m
    plt.plot(df_national_global['mois_annee'].astype(str), df_national_global['p

    # Add the curves for each selected department
    for departement in selected_departments:
        df_dept = average_price_par_dept_mois[average_price_par_dept_mois['code_
        plt.plot(df_dept['mois_annee'].astype(str), df_dept['prix_m2'], marker='

    plt.title("Evolution of the average price per m² by department (monthly)")
    plt.xlabel("Month and year")
    plt.ylabel("Average price per m² (€)")
    plt.legend(title="Types of property", bbox_to_anchor=(1.05, 1), loc='upper l
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

In [8]:

```
def display_number_of_room_department(df,department_select,types_properties_sele
    # We take the department,price range, slice surface and types of properties
    selected_departments = list(department_select.value)
    selected_types_properties = list(types_properties_select.value)
    selected_price_range = list(price_range_select.value)
    selected_slice_surface = list(slice_surface_select.value)

    # If nothing is selected we take all the values for each point
    if not selected_departments:
        selected_departments = list(df['code_departement'].unique())
    if not selected_types_properties:
        selected_types_properties = list(df['type_local'].unique())
    if not selected_price_range:
        selected_price_range = ["< 50k €", "50k - 100k €", "100k - 150k €", "150
    if not selected_slice_surface:
        selected_slice_surface = ['0-50 m²', '50-100 m²', '100-150 m²', '150-200

     # Filter data based with our filter
    df_filtered = df[(df['code_departement'].isin(selected_departments)) &
                     (df['type_local'].isin(selected_types_properties))&
                     (df['tranche_valeur'].isin(selected_price_range))&
                     (df['plage_surface'].isin(selected_slice_surface))].copy

    df_filtered_national = df[
                     (df['type_local'].isin(selected_types_properties))&
                     (df['tranche_valeur'].isin(selected_price_range))&
                     (df['plage_surface'].isin(selected_slice_surface))].copy

    # We take the number of main room per department
```

```
    number_piece_per_dept = df_filtered.groupby('code_departement')['nombre_piec

    # Ajouter la moyenne générale des départements sélectionnés
    average_selection = df_filtered['nombre_pieces_principales'].mean()
    national_average = df_filtered_national['nombre_pieces_principales'].mean()

    # Ajouter une barre pour la moyenne générale
    departments = list(number_piece_per_dept['code_departement'])
    number_room = list(number_piece_per_dept['nombre_pieces_principales'])

    plt.figure(figsize=(15, 8))
    plt.bar(departments, number_room, color='skyblue')
    #We plot line for average and national average of main rooms
    plt.axhline(average_selection, color='red', linestyle='dashed', linewidth=2,
    plt.axhline(national_average, color='green', linestyle='dashed', linewidth=2

    plt.title("Average number of rooms per department and per type of property s
    plt.xlabel("Department")
    plt.ylabel("Average number of rooms")
    plt.xticks(rotation=90)
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.tight_layout()
    plt.show()
```

In [9]:

```
def distribution_type_property_by_department(df,department_select,types_properti
    # We take the department,price range, slice surface and types of properties
    selected_departments = list(department_select.value)
    selected_types_properties = list(types_properties_select.value)
    selected_price_range = list(price_range_select.value)
    selected_slice_surface = list(slice_surface_select.value)

   # If nothing is selected we take all the values for each point
    if not selected_departments:
        selected_departments = list(df['code_departement'].unique())
    if not selected_types_properties:
        selected_types_properties = list(df['type_local'].unique())
    if not selected_price_range:
        selected_price_range = list(df['tranche_valeur'].unique())
    if not selected_slice_surface:
        selected_slice_surface = list(df['plage_surface'].unique())

     # Filter data based with our filter
    df_filtered = df[(df['code_departement'].isin(selected_departments)) &
                     (df['type_local'].isin(selected_types_properties))&
                     (df['tranche_valeur'].isin(selected_price_range))&
                     (df['plage_surface'].isin(selected_slice_surface))].copy

    # Calculate the distribution of property types by department
    distribution = df_filtered.groupby(['code_departement', 'type_local']).size(

    # Normalize to obtain proportions (in %)
    distribution_norm = distribution.div(distribution.sum(axis=1), axis=0) * 100

    # creating a Stacked Bar Chart
    distribution_norm.plot(kind='bar', figsize=(15, 8), stacked=True, colormap='

    plt.title("Distribution of transactions by type of property and by departmen
    plt.xlabel("Department")
    plt.ylabel("Proportion of transactions (%)")
```

```
    plt.legend(title="Types of properties", bbox_to_anchor=(1.05, 1), loc='upper
    plt.grid(axis='y', linestyle='--', alpha=0.6)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

In [10]:

```
def distribution_of_land_value_by_tranche(df, department_select, types_propertie
    # Liste ordonnée des plages de prix
    price_order = [
        "< 50k €", "50k - 100k €", "100k - 150k €", "150k - 200k €", "200k - 250
        "250k - 300k €", "300k - 350k €", "350k - 400k €", "400k - 450k €", "450
        "500k - 550k €", "550k - 600k €", "600k - 650k €", "650k - 700k €", "700
        "750k - 800k €", "800k - 850k €", "850k - 900k €", "900k - 950k €", "950
        "1M - 1.5M €", "1.5M - 2M €", "2M - 2.5M €", "2.5M - 3M €", "3M - 4M €",
    ]

    # Transformer la colonne 'tranche_valeur' en catégorie ordonnée
    df['tranche_valeur'] = pd.Categorical(df['tranche_valeur'], categories=price

    # We take the department,price range, slice surface and types of properties
    selected_departments = list(department_select.value)
    selected_types_properties = list(types_properties_select.value)
    selected_price_range = list(price_range_select.value)
    selected_slice_surface = list(slice_surface_select.value)

    # If nothing is selected we take all the values for each point
    if not selected_departments:
        selected_departments = list(df['code_departement'].unique())
    if not selected_types_properties:
        selected_types_properties = list(df['type_local'].unique())
    if not selected_price_range:
        selected_price_range = price_order
    if not selected_slice_surface:
        selected_slice_surface = ['0-50 m²', '50-100 m²', '100-150 m²', '150-200
                                  '250-300 m²', '300-350 m²', '350-400 m²', '400
                                  '500-600 m²', '600-700 m²', '700-800 m²', '800

    # Filter data based with our filter
    df_filtered = df[(df['code_departement'].isin(selected_departments)) &
                     (df['type_local'].isin(selected_types_properties)) &
                     (df['tranche_valeur'].isin(selected_price_range)) &
                     (df['plage_surface'].isin(selected_slice_surface))].copy()

    # Count the number of transactions by tranche and by department
    distribution = df_filtered.groupby(['code_departement', 'tranche_valeur']).s
    # Normalize to obtain proportions (in %)
    distribution_norm = distribution.div(distribution.sum(axis=1), axis=0) * 100
    # Create a graphic
    distribution_norm.T.plot(kind='bar', figsize=(15, 8), stacked=True, colormap
    plt.title("Distribution of transactions by land value bracket and by departm
    plt.xlabel("Land value range (€)")
    plt.ylabel("Proportion of transactions (%)")
    plt.legend(title="Department", bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.grid(axis='y', linestyle='--', alpha=0.6)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

In [11]:
```
def price_m2_per_surface(df, department_select, types_properties_select,price_ra
    # We take the department,price range, slice surface and types of properties
    selected_departments = list(department_select.value)
    selected_types_properties = list(types_properties_select.value)
    selected_price_range = list(price_range_select.value)
    selected_slice_surface = list(slice_surface_select.value)

    # If nothing is selected we take all the values for each point
    if not selected_departments:
        selected_departments = list(df['code_departement'].unique())
    if not selected_types_properties:
        selected_types_properties = list(df['type_local'].unique())
    if not selected_price_range:
        selected_price_range = ["< 50k €", "50k - 100k €", "100k - 150k €", "150
    if not selected_slice_surface:
        selected_slice_surface = ['0-50 m²', '50-100 m²', '100-150 m²', '150-200

     # Filter data based with our filter
    df_filtered = df[(df['code_departement'].isin(selected_departments)) &
                     (df['type_local'].isin(selected_types_properties))&
                     (df['tranche_valeur'].isin(selected_price_range))&
                     (df['plage_surface'].isin(selected_slice_surface))].copy

    df_filtered_national = df[
                     (df['type_local'].isin(selected_types_properties))&
                     (df['tranche_valeur'].isin(selected_price_range))&
                     (df['plage_surface'].isin(selected_slice_surface))].copy

    # Calculate the average price by surface area and department
    average_price_per_surface = df_filtered.groupby(['plage_surface', 'code_depa

     # Calculate average by surface range for selection
    average_selection = df_filtered.groupby('plage_surface')['prix_m2'].mean().r

    # Calculate the national average by surface area
    national_average = df_filtered_national.groupby('plage_surface')['prix_m2'].

    # Creating the graph
    plt.figure(figsize=(12, 8))
    #For each department selected, we take the average price per suface and then
    for department in selected_departments:
        df_department = average_price_per_surface[average_price_per_surface['cod
        if not df_department.empty:
            plt.plot(
                df_department['plage_surface'],
                df_department['prix_m2'],
                marker='o',
                label=f'Department {department}'
            )
    plt.plot(
        average_selection['plage_surface'],
        average_selection['prix_m2'],
        marker='o',
        linestyle='--',
        label='Selected average',
        color='black'
    )

    # Then we add the National average line
```

```
    plt.plot(
        national_average['plage_surface'],
        national_average['prix_m2'],
        marker='o',
        linestyle=':',
        label='National average',
        color='red'
    )

    # graph configurations
    plt.title("Average price per m² depending on surface area ranges")
    plt.xlabel("Surface range (m²)")
    plt.ylabel("Average price per m² (€)")
    plt.legend(title="Departement", bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

In [12]:
```
def add_region_name_column(df):
    """
    Add a column 'region_name' to the dataframe with the name of the region corr

    :param df: DataFrame with the column 'code_insee' containing the INSEE codes

    :return: DataFrame with the added column 'region_name'
    """
    # Dictionary of INSEE codes for regions and their names
    regions_names = {
        "01": "Guadeloupe", "02": "Martinique", "03": "Guyane", "04": "La Réunio
        "24": "Centre-Val de Loire", "27": "Bourgogne-Franche-Comté", "28": "Nor
        "44": "Grand Est", "52": "Pays de la Loire", "53": "Bretagne", "75": "No
        "84": "Auvergne-Rhône-Alpes", "93": "Provence-Alpes-Côte d'Azur", "94":
    }
    df['region_name'] = df['code_insee'].map(regions_names)
    return df
```

In [13]:
```
def explore_france(data, department_select=None, types_properties_select=None, p
    """
    Displays a map of France with the average price per m² of land by region, an

    Parameters:
    - data : pd.DataFrame : The DataFrame containing the data of real estate tra
    - department_select : ipywidgets.SelectMultiple : The widget for selecting d
    - types_properties_select : ipywidgets.SelectMultiple : The widget for selec
    - price_range_select : ipywidgets.SelectMultiple : The widget for selecting
    - slice_surface_select : ipywidgets.SelectMultiple : The widget for selectin

    Returns:
    - Map of France with the average price per m² of land by region, and the loc

    """
    data_general = data.copy()

    #We take the department,price range, slice surface and types of properties s
    selected_departments = list(department_select.value)
    selected_types_properties = list(types_properties_select.value)
    selected_price_range = list(price_range_select.value)
    selected_slice_surface = list(slice_surface_select.value)
```

```
    # If nothing is selected we take all the values for each point
    if not selected_departments:
        selected_departments = list(data['code_departement'].unique())
    if not selected_types_properties:
        selected_types_properties = list(data['type_local'].unique())
    if not selected_price_range:
        selected_price_range = ["< 50k €", "50k - 100k €", "100k - 150k €", "150
    if not selected_slice_surface:
        selected_slice_surface = ['0-50 m²', '50-100 m²', '100-150 m²', '150-200

    # Filter data based with our filter
    data = data[
        (data['code_departement'].isin(selected_departments)) &
        (data['type_local'].isin(selected_types_properties)) &
        (data['tranche_valeur'].isin(selected_price_range)) &
        (data['plage_surface'].isin(selected_slice_surface))
    ].copy()

    # Load shapefile
    shp_file_path = "regions.shx"
    map_france = gpd.read_file(shp_file_path)

    # Define the CRS
    if map_france.crs is None:
        map_france = map_france.set_crs(epsg=4326, allow_override=True)

    #We verify that the column 'code_insee' is in the correct format and correct
    data_general["code_insee"] = data_general["code_insee"].astype(str).str.zfil

    #Average price per m² by region
    ratio_average = data_general.groupby('code_insee')['prix_m2'].mean()

    # Add the average land value by region stylized in euros
    map_france['Average_land_value'] = map_france['code_insee'].map(ratio_averag
    map_france['Average_land_value'] = map_france['Average_land_value'].apply(la


    # Normalize the values
    ratio_average = (ratio_average - ratio_average.min()) / (ratio_average.max()

    # Logarithmic transformation to accentuate differences
    ratio_average = np.log1p(ratio_average * 10)

    map_france = map_france.set_index('code_insee').join(ratio_average, how='lef
    map_france = add_region_name_column(map_france)

    sample_data = data.copy() if department_select and types_properties_select a

    geometry = gpd.GeoSeries.from_xy(sample_data['longitude'], sample_data['lati
    points_gdf = gpd.GeoDataFrame(sample_data, geometry=geometry)

    # Creating the map
    map = folium.Map(location=[46.603354, 1.888334], zoom_start=6)
    map_france["prix_m2"].fillna(0, inplace=True)
    # We add the regions to the map
    folium.GeoJson(
        map_france,
        name='geojson',
        style_function=lambda feature: {
            'fillColor':  matplotlib.colors.rgb2hex(plt.cm.plasma(feature['prope
```

```
            'color': matplotlib.colors.rgb2hex(plt.cm.plasma(feature['properties
            'weight': 2,
            'fillOpacity': 0.5,
        },
        tooltip=folium.GeoJsonTooltip(fields=['region_name', 'Average_land_value

    ).add_to(map)

    # Add some points
    marker_cluster = MarkerCluster().add_to(map)
    #We added popup to each properties to display information about the property
    for idx, row in points_gdf.iterrows():
        folium.Marker(
            location=[row['latitude'], row['longitude']],
            popup=f"{row.get('region_name', 'N/A')} \n Type : {row.get('type_loc
            icon=folium.Icon(color='red', icon='home')
        ).add_to(marker_cluster)

    display(map)
    return map
```

# III. Execution :

## Présentation of the case :

The investor is a family head searching for a home for himself, his wife, and their two children. The need for space is critical to ensure family comfort, hence the requirement for a minimum of 100 $m^2$.

- Property type: Apartment only (houses are excluded).
- Preferred location: Paris or the western part of the Île-de-France region (departments 75, 78, 92, 94, 91, 95).
- Maximum budget: €350,000 (including all purchase-related costs and taxes).
- Priority: Floor area is the key criterion, and the investor is willing to compromise on location to acquire a larger property within budget constraints.
- Seeking an excellent price-to-floor-area ratio, potentially exploring properties on the outskirts of high-demand areas to maximize livable space.
- This purchase is motivated by an immediate family need, rather than speculative purposes.
- The investor is mindful of real estate price trends in targeted areas but prioritizes a quick solution to meet his family's needs.
- Wants each member of the family to have space and a personal room

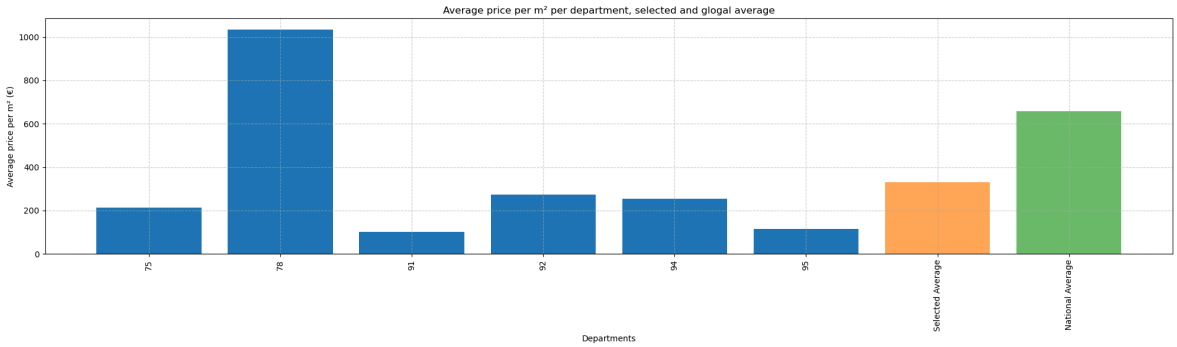
First of all lets apply the following filters :

- We selected the following departement : 75, 78, 91, 92, 94, 95
- We selected only appartements
- We selected all the price from 0 to 350k€
- We selected all the surface from 100 $m^2$
- We choose to use all our function except the one showing the type of property by departement and the one showing the distribution of land values because we only

considered the appartement and the objective here is not to speculate on the price evolution of the property

In [16]:
```
selection_filters_and_analyses(data)
```

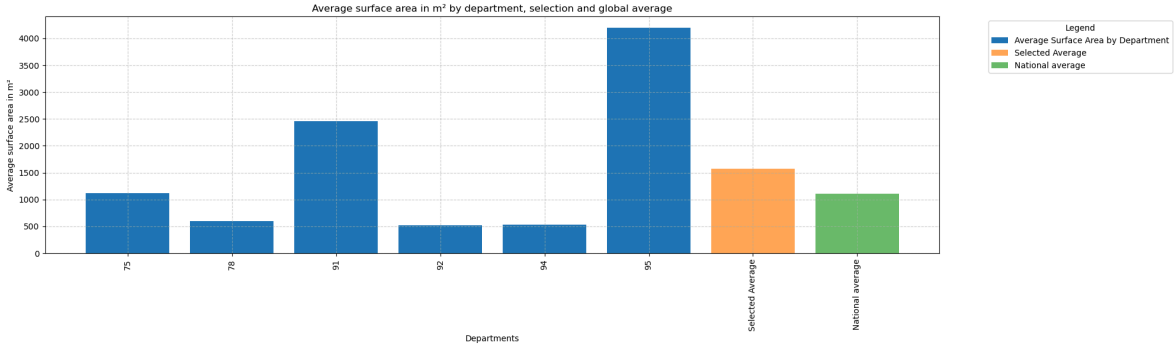
```
SelectMultiple(description='Departments', layout=Layout(width='60%'), options=('0
1', '02', '03', '04', '05', '…
SelectMultiple(description='Types of property', layout=Layout(width='60%'), optio
ns=('House', 'Dependency (Iso…
SelectMultiple(description='Price range', layout=Layout(width='60%'), options=('<
50k €', '50k - 100k €', '100…
SelectMultiple(description='Surface slice', layout=Layout(width='60%'), options=
('0-50 m²', '50-100 m²', '100-…
SelectMultiple(description='Analyse', layout=Layout(width='60%'), options=('Avera
ge price per m²', 'Average su…
Button(button_style='success', description='Apply the filters', style=ButtonStyle
())
```




This first function shows us that for the criteria we used for this investor, the 91 and 95 départements seem to be the best, as this investor's objective is to maximise his surface area with a budget of €350k. Location is not a criterion apart from the choice of department. These two départements seemed to be the cheapest, so we can assume that for the same price he can get a better surface area.

In [17]:
```
selection_filters_and_analyses(data)
```

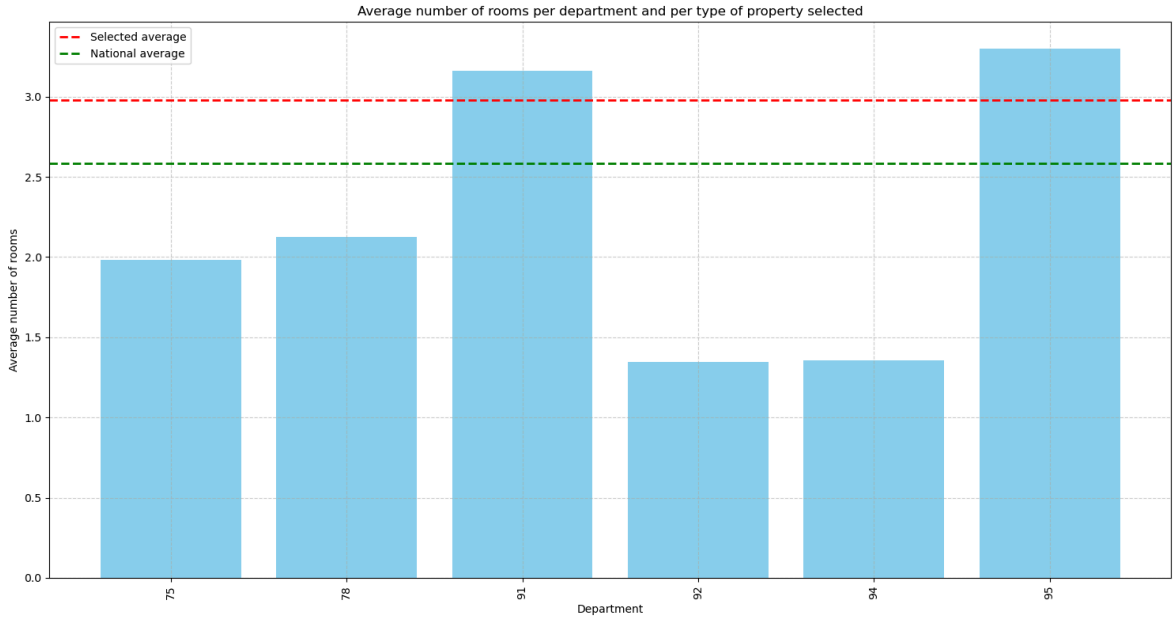
```
SelectMultiple(description='Departments', layout=Layout(width='60%'), options=('0
1', '02', '03', '04', '05', '…
SelectMultiple(description='Types of property', layout=Layout(width='60%'), optio
ns=('House', 'Dependency (Iso…
SelectMultiple(description='Price range', layout=Layout(width='60%'), options=('<
50k €', '50k - 100k €', '100…
SelectMultiple(description='Surface slice', layout=Layout(width='60%'), options=
('0-50 m²', '50-100 m²', '100-…
SelectMultiple(description='Analyse', layout=Layout(width='60%'), options=('Avera
ge price per m²', 'Average su…
Button(button_style='success', description='Apply the filters', style=ButtonStyle
())
```




As with the previous analysis, our objective here is to study the average surface area of flats in the 6 départements covered by our research. The previous analysis showed that the price per square metre was much lower in 91 and 95 than in the other départements. This first choice of department is also confirmed here in the context of this new indicator, as 91 and 95 are the two departments with the largest average flat surface area compared with the other departments in our study. For the time being, we are confident in our choice of 91 and 95 as departments for this investor.

In [19]:
```
selection_filters_and_analyses(data)
```

```
SelectMultiple(description='Departments', layout=Layout(width='60%'), options=('0
1', '02', '03', '04', '05', '…
SelectMultiple(description='Types of property', layout=Layout(width='60%'), optio
ns=('House', 'Dependency (Iso…
SelectMultiple(description='Price range', layout=Layout(width='60%'), options=('<
50k €', '50k - 100k €', '100…
SelectMultiple(description='Surface slice', layout=Layout(width='60%'), options=
('0-50 m²', '50-100 m²', '100-…
SelectMultiple(description='Analyse', layout=Layout(width='60%'), options=('Avera
ge price per m²', 'Average su…
Button(button_style='success', description='Apply the filters', style=ButtonStyle
())
```

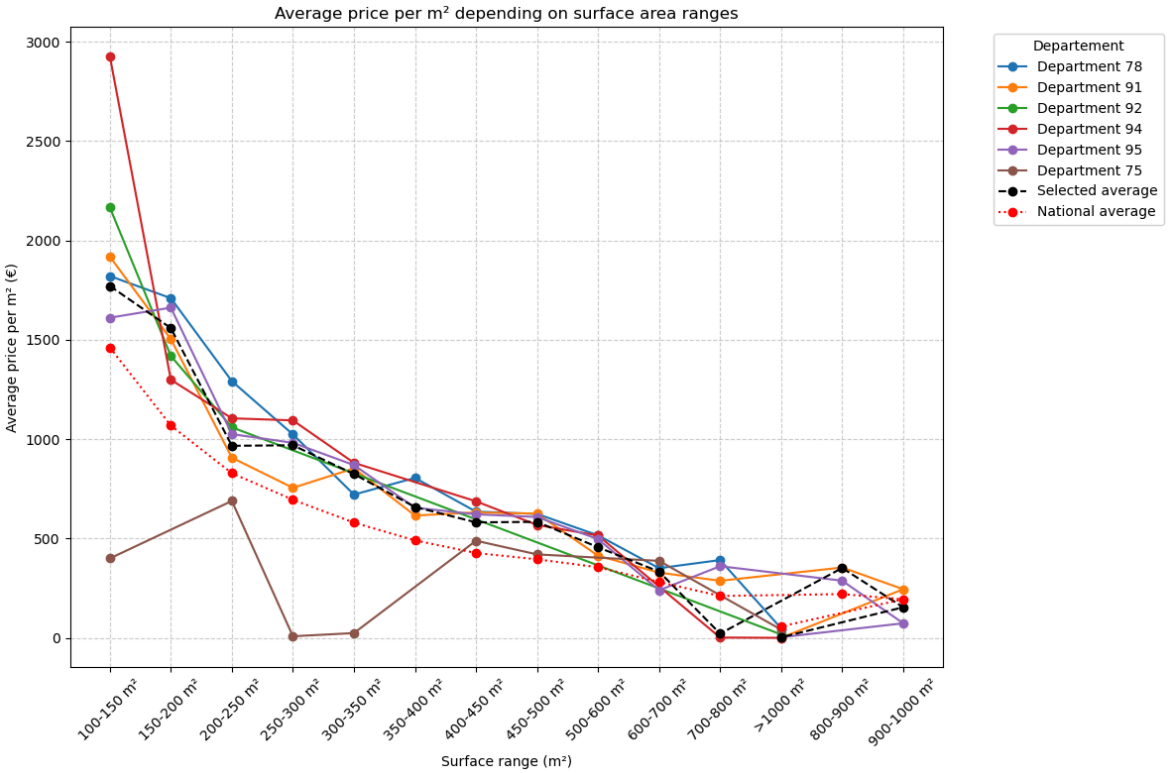



This indicator shows us the average number of main rooms per département according to the search criteria we entered earlier. Here we have the case of a father looking for a home for his wife and two children. As we explained, surface area is important to this buyer, who also wants each member of the family to have space and a personal room.

This indicator shows us that 91 and 95 are the two départements with the highest number of main rooms compared with all the other départements in our study, with more than 3 rooms on average. This average could allow us to validate 91 and 95 as our choice of département.

In [22]:
```
selection_filters_and_analyses(data)
```

```
SelectMultiple(description='Departments', layout=Layout(width='60%'), options=('0
1', '02', '03', '04', '05', '…
SelectMultiple(description='Types of property', layout=Layout(width='60%'), optio
ns=('House', 'Dependency (Iso…
SelectMultiple(description='Price range', layout=Layout(width='60%'), options=('<
50k €', '50k - 100k €', '100…
SelectMultiple(description='Surface slice', layout=Layout(width='60%'), options=
('0-50 m²', '50-100 m²', '100-…
SelectMultiple(description='Analyse', layout=Layout(width='60%'), options=('Avera
ge price per m²', 'Average su…
Button(button_style='success', description='Apply the filters', style=ButtonStyle
())
```

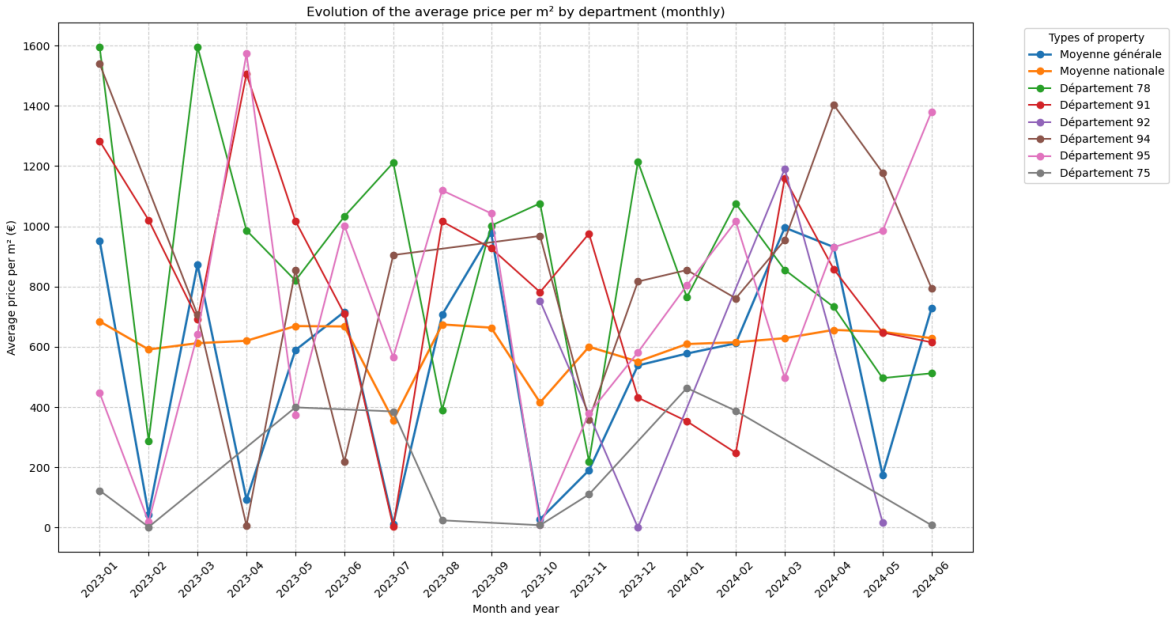



Here, the aim of this indicator was to observe changes in price per $m^2$ as a function of the surface area of the property concerned. This would have allowed us to see whether there was a surface area for which the price per $m^2$ was lower than for others. Here, we can make the same assumption for all the départements covered by our study: the larger the surface area of the property, the lower the price per $m^2$. This is beneficial in the case of our buyer who wants a large surface area, as it means that the price per $m^2$ will be lower if he or she buys a property with a large surface area.

In [24]:
```
selection_filters_and_analyses(data)
```

```
SelectMultiple(description='Departments', layout=Layout(width='60%'), options=('0
1', '02', '03', '04', '05', '…
SelectMultiple(description='Types of property', layout=Layout(width='60%'), optio
ns=('House', 'Dependency (Iso…
```

```
SelectMultiple(description='Price range', layout=Layout(width='60%'), options=('<
50k €', '50k - 100k €', '100…
SelectMultiple(description='Surface slice', layout=Layout(width='60%'), options=
('0-50 m²', '50-100 m²', '100-…
SelectMultiple(description='Analyse', layout=Layout(width='60%'), options=('Avera
ge price per m²', 'Average su…
Button(button_style='success', description='Apply the filters', style=ButtonStyle
())
```




Even if the investor's objective is not to invest in real estate with a view to profitability, it may be worth observing whether one département seems to be more attractive than another. This could be a plus to facilitate the investor's choice if he hesitates between several départements. In our case, our previous indicators have shown that 91 and 95 are the two departments we need to focus on. Our investor wants to buy a property now. As it happens, 95 is currently experiencing a significant increase in price per $m^2$ over the last few months, while 91 is in a slightly declining phase, with a downward trend in price per $m^2$ over the last few months. So if you had to choose between these two departments, 91 might be the one to choose.

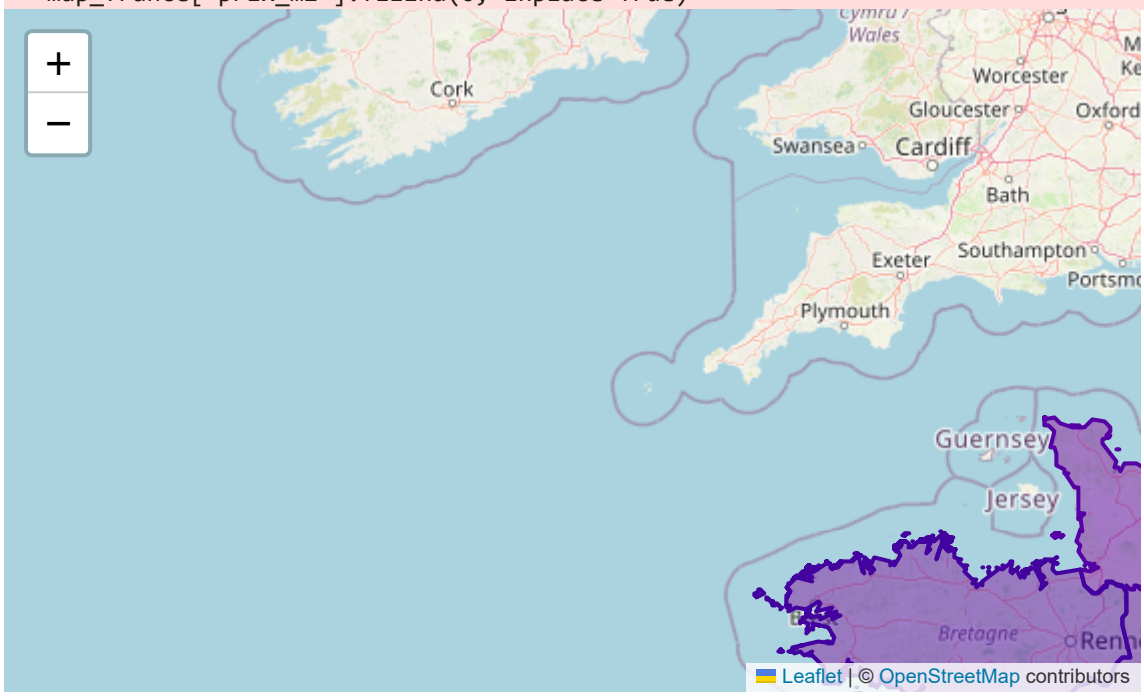
In [23]:
```
selection_filters_and_analyses(data)
```

```
SelectMultiple(description='Departments', layout=Layout(width='60%'), options=('0
1', '02', '03', '04', '05', '…
SelectMultiple(description='Types of property', layout=Layout(width='60%'), optio
ns=('House', 'Dependency (Iso…
SelectMultiple(description='Price range', layout=Layout(width='60%'), options=('<
50k €', '50k - 100k €', '100…
SelectMultiple(description='Surface slice', layout=Layout(width='60%'), options=
('0-50 m²', '50-100 m²', '100-…
SelectMultiple(description='Analyse', layout=Layout(width='60%'), options=('Avera
ge price per m²', 'Average su…
Button(button_style='success', description='Apply the filters', style=ButtonStyle
())
```

```
C:\Users\maube\AppData\Local\Temp\ipykernel_10476\97304001.py:76: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  map_france["prix_m2"].fillna(0, inplace=True)
```




This map shows the large number of properties matching our customer's search criteria in recent years. It shows that this type of search exists on the market and that it is possible for him to find an apartment for himself and his family in the requested criteria. What's more, this map shows that the 91 and 95 départements have the most properties matching our customer's search criteria. A future enhancement could be to also display schools, supermarkets and other amenities on this map, enabling our investor's search to be narrowed down according to one or more locations.

## Conclusion

To conclude, in the case of this investor looking for an apartment with a substantial surface area, in Paris or the Paris region. We have selected two departments that particularly stand out: 95 and 91. If he had to choose just one, it could be 91, because looking at the price per $m^2$ in this department, our buyer is likely to make a capital gain on resale.