Real Estate Data Analysis and Enhancing Market Predictions with Generative AI

Introduction

In the current context of the French real estate market, understanding and forecasting trends in demand and prices is essential for researchers, investors, and policymakers. This notebook aims to explore how integrating generative artificial intelligence can improve traditional predictive models by utilizing real estate transaction data and macroeconomic indicators.

In this notebook, we will:

- Collect and analyze open-source data on real estate transactions in France.
- Incorporate relevant macroeconomic indicators that influence the real estate market.
- Build predictive models to estimate real estate prices and demand.
- Enhance these models using generative AI techniques, including fine-tuning a large language model (LLM) to extract new indicators from textual data.
- Provide a training plan for researchers to facilitate understanding and contribution to the project.

Table of Contents

1. Data Collection and Explanation

This section is dedicated to collecting and preparing the datasets essential for analyzing real estate trends and predicting demand. It includes historical transaction data as well as macroeconomic

indicators to account for broader economic factors that may impact real estate markets. Combining these two data types will provide a robust foundation for predictive modeling.

1.1 Real Estate Transaction Data

Objective: Collect and process detailed historical real estate transaction data across France to establish trends, examine patterns, and support predictive modeling for demand and pricing.

1.1.1 Data Source

We selected the **DVF Géolocalisées (Demandes de Valeurs Foncières Géolocalisées)** dataset, derived from the **DVF (Demandes de Valeurs Foncières)** dataset and provided by France's official open data portal. This enhanced version offers several crucial improvements, making it highly suitable for our analysis.

- **Transaction Date**: With dates in ISO format, this dataset supports detailed temporal analysis, including seasonal trends and long-term market shifts.
- **Property Type**: Differentiates between residential, commercial, and other property types, enabling more focused segmentation and trend exploration.
- **Location**: Precise coordinates (latitude/longitude in WGS-84) allow for exact parcel-level spatial analysis. This detail enhances the ability to analyze market dynamics down to neighborhood levels.
- **Price and Surface Area**: Essential data points for calculating per-square-meter prices, a key metric for evaluating and comparing market values across regions and property types.

Advantages of DVF Géolocalisées
The DVF Géolocalisées dataset provides a **standardized, enriched, and geolocated format** that is ideal for **spatial, temporal, and financial analyses** of the French real estate market. With its precise geographical coordinates and enriched details, this dataset is particularly valuable for building robust predictive models that incorporate specific location-based insights. Its consistency and ready-to-use format also enhance data processing efficiency and quality.

Why Choose DVF Géolocalisées Over DVF (Non-Géolocalisées)?
While both versions offer detailed transaction data, **we selected the DVF Géolocalisées dataset** for its **superior spatial precision** and standardized format. The added geolocation data provides essential latitude and longitude information that enables parcel-level mapping, which is crucial for high-resolution analysis of spatial trends across France. This choice ensures that our models can capture precise, location-specific insights that the DVF non-géolocalisées version does not support.

We collected data from **2019 to 2024** to capture recent trends and provide a comprehensive analysis of market dynamics over the last five years.

1.1.2 Initial Exploration

```
import pandas as pd
from tabulate import tabulate
import numpy as np

# Parameters
sample_fraction = 0.20  # Load 10% of each department's data
base_url = "https://files.data.gouv.fr/geo-dvf/latest/csv/{year}/full.csv.gz
years = range(2020, 2024)

# Columns to keep
dvf_columns_to_keep = [
    'valeur_fonciere', 'code_postal', 'code_commune', 'surface_reelle_bati',
    'nombre_pieces_principales', 'surface_terrain', 'longitude', 'latitude',
    'nature_mutation', 'type_local', 'date_mutation'
]

# List to collect data for each year
dvf_data_geo = []

for year in years:
    # Load dataset with selected columns and optimized data types
    url = base_url.format(year=year)
    yearly_data = pd.read_csv(url, compression='gzip', usecols=dvf_columns_t

    # Downcast numeric data types to save memory
    yearly_data['valeur_fonciere'] = pd.to_numeric(yearly_data['valeur_fonci
    yearly_data['surface_reelle_bati'] = pd.to_numeric(yearly_data['surface_
    yearly_data['nombre_pieces_principales'] = pd.to_numeric(yearly_data['no
    yearly_data['surface_terrain'] = pd.to_numeric(yearly_data['surface_terr

    # Convert 'date_mutation' to datetime format for efficient date operatio
    yearly_data['date_mutation'] = pd.to_datetime(yearly_data['date_mutation

    # Randomly sample a fraction of data from each department to reduce memo
    yearly_sampled_data = yearly_data.groupby('code_commune').apply(lambda x

    # Append sampled data to the list
    dvf_data_geo.append(yearly_sampled_data)

# Concatenate all years into a single DataFrame
dvf_data_geo = pd.concat(dvf_data_geo, ignore_index=True)

# Define the number of columns to display
num_columns_to_display = 10
```

```
# Display a sample to verify data integrity and structure
print(tabulate(dvf_data_geo.iloc[:, :num_columns_to_display].head(), headers
```

```
<ipython-input-1-db10c778fe41>:35: DeprecationWarning: DataFrameGroupBy.apply operated c
  yearly_sampled_data = yearly_data.groupby('code_commune').apply(lambda x: x.sample(fra
<ipython-input-1-db10c778fe41>:35: DeprecationWarning: DataFrameGroupBy.apply operated c
  yearly_sampled_data = yearly_data.groupby('code_commune').apply(lambda x: x.sample(fra
<ipython-input-1-db10c778fe41>:35: DeprecationWarning: DataFrameGroupBy.apply operated c
  yearly_sampled_data = yearly_data.groupby('code_commune').apply(lambda x: x.sample(fra
<ipython-input-1-db10c778fe41>:35: DeprecationWarning: DataFrameGroupBy.apply operated c
  yearly_sampled_data = yearly_data.groupby('code_commune').apply(lambda x: x.sample(fra
+----+---------------------+----------------------+-----------------+--------------+
|    | date_mutation       | nature_mutation      |  valeur_fonciere |   code_postal |
|----+---------------------+----------------------+-----------------+--------------+
|  0 | 2020-12-23 00:00:00 | Vente                |            87500 |          1400 |
|  1 | 2020-01-06 00:00:00 | Vente terrain à bâtir |            94380 |          1400 |
|  2 | 2020-07-29 00:00:00 | Vente terrain à bâtir |            77880 |          1400 |
|  3 | 2020-11-10 00:00:00 | Vente                |           267000 |          1400 |
|  4 | 2020-11-09 00:00:00 | Vente                |           150000 |          1400 |
+----+---------------------+----------------------+-----------------+--------------+
```

1.2 Macroeconomic Data

Objective: Incorporate macroeconomic indicators to capture the broader economic context that can influence real estate demand and pricing trends over time. This context provides essential insights into how external factors, such as economic growth, employment levels, and interest rates, impact the housing market.

1.2.1 Data Source and Indicator Selection

For our analysis, we have selected three critical macroeconomic datasets that impact real estate markets in France. Each dataset has been chosen for its specific role in capturing economic conditions related to employment, economic growth, and financing costs:

1. **Gross Domestic Product (PIB) Growth Rate, Household Consumption, and Investment Contribution to PIB**
    - **Source**: National PIB Data from INSEE
    - **Description**: This dataset provides key annual economic indicators for France, including the PIB (GDP) growth rate, household consumption, and investment contributions to PIB, derived from national accounts data.
    - **Importance**:

- **PIB (GDP) Growth Rate**: A fundamental economic indicator, PIB growth reflects the overall health of the economy. Robust growth typically signifies a strong economy, fostering higher property demand as consumer confidence and investment activities increase. Conversely, slowing or negative growth can signal economic challenges, often reducing demand and potentially impacting property prices.
- **Household Consumption**: Household spending is a critical driver of demand in the housing market, as higher consumption levels can indicate increased disposable income and a greater willingness to invest in property. Lower consumption may suggest economic pressures, which can soften housing demand.
- **Investment Contribution**: Investment levels, particularly in fixed capital, are indicative of economic expansion and business confidence. High investment often correlates with construction activity and real estate development, influencing housing supply and demand dynamics.

2. **Employment and Unemployment Rates**
    - **Sources**: Activité, Emploi et Chômage from data.gouv.fr, Taux de Chômage Localisé from INSEE
    - **Description**: This data covers both general employment statistics and localized unemployment rates. The **Activité, Emploi et Chômage** dataset provides employment activity and general unemployment rates based on continuous surveys across France. The **Taux de Chômage Localisé** dataset from INSEE offers quarterly, department-level unemployment rates, with survey data for DOM regions to address short-term fluctuations specific to these areas.
    - **Importance**: Employment levels and unemployment rates are crucial drivers of housing demand. Stable income from employment encourages property investment, while high unemployment can dampen demand by reducing purchasing power. The **Taux de Chômage Localisé** dataset, in particular, provides precise regional insights that enable us to examine variations in demand across different areas. Combining these datasets gives a comprehensive view of employment trends, supporting our analysis of real estate markets on both national and regional scales.
3. **Annual Average Long-Term Interest Rates**
    - **Source**: INSEE Taux d'Intérêt Moyens
    - **Description**: This dataset provides annual average long-term interest rates, a key factor in determining mortgage costs and borrowing capacity.
    - **Importance**: Long-term interest rates directly impact mortgage affordability and borrowing power. Lower rates reduce financing costs, making property purchases more accessible and boosting demand, while higher rates increase borrowing costs, potentially cooling demand and affecting property prices. This dataset is essential for

analyzing how variations in financing conditions influence housing demand and price trends.

1.2.2 Data Import

```
import zipfile
import io
import requests

# Define URLs and file paths within the zip archives
datasets = {
    "unemployment": {
        "url": "https://www.insee.fr/fr/statistiques/series/csv/famille/1027
        "file": "valeurs_trimestrielles.csv",
        "sep": ";"
    },
    "pib": {
        "url": "https://www.insee.fr/fr/statistiques/series/csv/famille/1179
        "file": "valeurs_annuelles.csv",
        "sep": ";"
    },
    "interest_rate": {
        "url": "https://www.insee.fr/fr/statistiques/fichier/2130712/econ-ge
        "file": "econ-gen-taux-interet-ue.xlsx",
        "skiprows": 3
    }
}

# Initialize an empty dictionary to store DataFrames
dataframes = {}

# Download and load unemployment and GDP data from zip files
for key, info in datasets.items():
    response = requests.get(info["url"])
    with zipfile.ZipFile(io.BytesIO(response.content)) as z:
        with z.open(info["file"]) as file:
            if info["file"].endswith('.csv'):
                dataframes[key] = pd.read_csv(file, sep=info.get("sep", ",")
            elif info["file"].endswith('.xlsx'):
                dataframes[key] = pd.read_excel(file, skiprows=info.get("ski

for name, df in dataframes.items():
    print(f"Sample of {name} Data:")
    print(tabulate(df.iloc[:, :num_columns_to_display].head(), headers='keys
```

```
print("\n")
```

```
Sample of unemployment Data:
```

	Libellé	idBank	Der
0	Taux de chômage localisé par région - France métropolitaine	1.51584e+06	19/
1	Codes	nan	nar
2	Taux de chômage localisé par région - Île-de-France	1.51584e+06	19/
3	Codes	nan	nar
4	Taux de chômage localisé par région - Centre-Val de Loire	1.51585e+06	19/

```
Sample of pib Data:
```

	Libellé
0	Comptes nationaux annuels base 2020 - Exportations de biens et de services - Tota
1	Codes
2	Comptes nationaux annuels base 2020 - Produit intérieur brut - Evolution en volum
3	Codes
4	Comptes nationaux annuels base 2020 - Dépense de consommation finale - Total des

```
Sample of interest_rate Data:
```

	Pays	2008	2009	2010	2011	2012	2013	
0	France	4.23431	3.64881	3.11701	3.32106	2.53599	2.20434	1.
1	Zone euro à 19	4.35601	4.02879	3.78192	4.311	3.04785	3.01186	2.
2	Royaume-Uni	4.59072	3.64752	3.62443	3.13599	1.91804	2.38978	2.
3	États-Unis	3.66667	3.25667	3.21417	2.78583	1.8025	2.35083	2.
4	Japon	1.46733	1.33375	1.14833	1.10242	0.835583	0.689667	0.

2. Exploratory Data Analysis (EDA)

In this section, we perform an Exploratory Data Analysis (EDA) to gain insights into the datasets, identify key patterns, and prepare the data for further modeling. This process includes data cleaning, handling missing values, merging datasets, and analyzing core statistics to understand the structure and distribution of key variables.

2.1 Data Preprocessing

Preprocessing prepares the datasets for analysis by ensuring data quality and alignment across variables.

2.1.1 Data Cleaning

This step addresses duplicates, format inconsistencies, and any other issues to ensure clean and reliable data.

```
# 1. dvf Dataset

# Making a copy to work safely with the selected columns
dvf_data_cleaned = dvf_data_geo[dvf_columns_to_keep].copy()

# Step 2: Convert 'date_mutation' to datetime format and extract year, quart
dvf_data_cleaned['date_mutation'] = pd.to_datetime(dvf_data_cleaned['date_mu
dvf_data_cleaned['year'] = dvf_data_cleaned['date_mutation'].dt.year
dvf_data_cleaned['quarter'] = dvf_data_cleaned['date_mutation'].dt.quarter
dvf_data_cleaned['month'] = dvf_data_cleaned['date_mutation'].dt.month

# Optional: Drop the 'date_mutation' column if it’s no longer needed
dvf_data_cleaned.drop(columns=['date_mutation'], inplace=True)

# Step 3: Calculate new features to capture property quality and value indic
dvf_data_cleaned['price_per_sqm'] = dvf_data_cleaned['valeur_fonciere'] / dv
dvf_data_cleaned['sqm_per_room'] = dvf_data_cleaned['surface_reelle_bati'] /

# Step 4: Replace the data in the 'dataframes' dictionary
dataframes['dvf_data'] = dvf_data_cleaned

# Display the DataFrame
print(tabulate(dvf_data_cleaned.iloc[:, :num_columns_to_display].head(), hea
```

```
+----+-------------------+---------------+----------------+-----------------------+-----
|    |   valeur_fonciere |   code_postal |   code_commune |   surface_reelle_bati |   nc
|----+-------------------+---------------+----------------+-----------------------+-----
|  0 |             87500 |          1400 |          01001 |                    93 |
|  1 |             94380 |          1400 |          01001 |                   nan |
|  2 |             77880 |          1400 |          01001 |                   nan |
|  3 |            267000 |          1400 |          01001 |                   157 |
|  4 |            150000 |          1400 |          01001 |                   135 |
+----+-------------------+---------------+----------------+-----------------------+-----
```

```
# 2. Unemployment Dataset
unemployment_data = dataframes['unemployment']
```

```
# Identify columns to keep based on years
years_to_keep = [str(year) for year in years]
columns_to_keep = ['Libellé', 'idBank', 'Dernière mise à jour'] + \
                  [col for col in unemployment_data.columns if any(year in c

# Filter for selected columns
unemployment_data = unemployment_data[columns_to_keep]

# Filter for required columns
unemployment_data = unemployment_data[columns_to_keep]

# Drop rows where 'Libellé' contains 'Codes'
unemployment_data = unemployment_data[~unemployment_data['Libellé'].str.cont

# Load department codes from official CSV file
department_codes_url = 'https://www.data.gouv.fr/fr/datasets/r/70cef74f-70b1
department_codes_df = pd.read_csv(department_codes_url)
department_codes_df = department_codes_df[['code_departement', 'nom_departer
department_codes_df.rename(columns={'code_departement': 'department_code', '
department_codes_df['department_name'] = department_codes_df['department_nar

# Filter for rows containing "département" and extract department name
unemployment_data_dept = unemployment_data[unemployment_data['Libellé'].str.

# Extract department name and standardize for matching
unemployment_data_dept['department_name'] = unemployment_data_dept['Libellé'

# Merge with department codes to add department_code
unemployment_data_dept = unemployment_data_dept.merge(department_codes_df, o
unemployment_data_dept.drop(columns=['department_name'], inplace=True)  # Dr

# Reshape to long format
unemployment_data_long = unemployment_data_dept.melt(id_vars=['Libellé', 'de
                                                     var_name='year_quarter'

# Split 'year_quarter' into 'year' and 'quarter' columns with .loc to prever
unemployment_data_long[['year', 'quarter']] = unemployment_data_long['year_q
unemployment_data_long.loc[:, 'year'] = unemployment_data_long['year'].astyp
unemployment_data_long.loc[:, 'quarter'] = unemployment_data_long['quarter']

# Select the final cleaned columns
unemployment_data_long = unemployment_data_long[['department_code', 'year',
unemployment_data_long.reset_index(drop=True, inplace=True)
```

```
# Replace the data in the dataframes dict
dataframes['unemployment'] = unemployment_data_long

# Display the cleaned unemployment data
print(tabulate(unemployment_data_long.iloc[:, :num_columns_to_display].head(
```

```
+----+-------------------+--------+-----------+---------------------+
|    |   department_code |   year |   quarter |   unemployment_rate |
|----+-------------------+--------+-----------+---------------------|
|  0 |                12 |   2020 |         1 |                 5.9 |
|  1 |                01 |   2020 |         1 |                 5.8 |
|  2 |                02 |   2020 |         1 |                11.2 |
|  3 |                03 |   2020 |         1 |                 8.3 |
|  4 |                04 |   2020 |         1 |                 8.9 |
+----+-------------------+--------+-----------+---------------------+
```

```
# 3. Interest rate Dataset
interest_rate_data = dataframes['interest_rate']

# Convert column names to strings to ensure compatibility
interest_rate_data.columns = interest_rate_data.columns.map(str)

# Garder uniquement les données pour la France
interest_rate_data = interest_rate_data[interest_rate_data['Pays'] == 'Franc

# Transformation des données en format long pour faciliter l'analyse
interest_rate_long = interest_rate_data.melt(id_vars=['Pays'], var_name='Yea
interest_rate_long['Year'] = interest_rate_long['Year'].astype(int)  # Conve
interest_rate_long.drop(columns=['Pays'], inplace=True)

# Filtrer pour garder uniquement les années
interest_rate_recent = interest_rate_long[(interest_rate_long['Year'] >= yea

# Réinitialiser les index pour un DataFrame propre
interest_rate_recent.reset_index(drop=True, inplace=True)

# Replace the data in the dataframes dict
dataframes['interest_rate'] = interest_rate_recent

# Afficher les données finales
print(tabulate(interest_rate_recent.iloc[:, :num_columns_to_display].head(),
```

```
+----+--------+-----------------+
|    |   Year |   Interest_Rate |
|----+--------+-----------------|
|  0 |   2020 |        -0.14526 |
```

```
|  1 |   2021 |      0.00690516 |
|  2 |   2022 |      1.70081    |
|  3 |   2023 |      2.99591    |
+----+--------+-----------------+


# 4. PIB Dataset
pib_data = dataframes['pib']

# Step 1: Filter out rows where 'Libellé' contains 'Codes' and clean 'Libel
pib_data = pib_data[~pib_data['Libellé'].str.contains('Codes', na=False)].co
pib_data.loc[:, 'Libellé'] = pib_data['Libellé'].str.replace('Comptes natior

# Step 2: Define and rename selected indicators
indicator_rename_map = {
    'Produit intérieur brut - Evolution en volume - Prix chaîné année de bas
    'Dépense de consommation finale - Ménages hors entrepreneurs individuels
    'Formation brute de capital fixe - Total des secteurs institutionnels -
}

# Filter data for selected indicators
pib_data = pib_data[pib_data['Libellé'].isin(indicator_rename_map.keys())]

# Rename indicators for clarity
pib_data['Libellé'] = pib_data['Libellé'].map(indicator_rename_map)

# Step 3: Select years of interest and reshape
columns_of_interest = ['Libellé'] + [str(year) for year in years]
pib_data = pib_data[columns_of_interest]

# Step 4: Melt and pivot data to have one column per indicator
pib_long = pib_data.melt(id_vars=['Libellé'], var_name='Year', value_name='V
pib_long['Year'] = pib_long['Year'].astype(int)

# Step 5: Pivot to make each indicator a separate column
pib_cleaned = pib_long.pivot(index='Year', columns='Libellé', values='Value'

# Replace the data in the dataframes dict
dataframes['pib'] = pib_cleaned

# Display the cleaned PIB data
print(tabulate(pib_cleaned.iloc[:, :num_columns_to_display].head(), headers=
```

```
+----+--------+-------------+------------------------+-------------+
|    |   Year |   GDP_Growth |   Household_Consumption |   Investment |
```

```
|----+--------+-------------+------------------------+-------------|
|  0 |   2020 |       -7.441 |                  -3.35  |       -1.307 |
|  1 |   2021 |        6.882 |                   2.68  |        2.184 |
|  2 |   2022 |        2.571 |                   1.555 |       -0.011 |
|  3 |   2023 |        0.936 |                   0.427 |        0.092 |
+----+--------+-------------+------------------------+-------------+
```

2.1.2 Handling Missing Values

We will identify missing values and apply suitable techniques (e.g., imputation or exclusion) to minimize their impact on analysis.

```
# 1. Search for missing values in each dataframe and store results
def find_missing_values(df):
    missing_info = [(col, df[col].isnull().sum()) for col in df.columns if c
    if missing_info:
        print(f"Missing values in DataFrame '{df_name}':")
        missing_df = pd.DataFrame(missing_info, columns=["Column", "Missing
        print(tabulate(missing_df, headers='keys', tablefmt='psql'))

for df_name, df in dataframes.items():
    find_missing_values(df)
```

```
Missing values in DataFrame 'dvf_data':
+----+--------------------------+------------------+
|    | Column                   |   Missing Values |
|----+--------------------------+------------------|
|  0 | valeur_fonciere          |            31970 |
|  1 | code_postal              |            25637 |
|  2 | surface_reelle_bati      |          2158707 |
|  3 | nombre_pieces_principales |          1371706 |
|  4 | surface_terrain          |          1065670 |
|  5 | longitude                |            61525 |
|  6 | latitude                 |            61525 |
|  7 | type_local               |          1369653 |
|  8 | price_per_sqm            |          2166216 |
|  9 | sqm_per_room             |          2158707 |
+----+--------------------------+------------------+
```

```
# 1. DVF Dataset (9 columns to handle)

# Copy dataframe
dvf_data_cleaned = dataframes['dvf_data'].copy()

# Step 1: # Drop rows with missing 'valeur_fonciere' to keep target values a
dvf_data_cleaned = dvf_data_cleaned[(dvf_data_cleaned['valeur_fonciere'].not
```

```
# Step 2: Fill 'code_postal' using mode within 'code_commune'
dvf_data_cleaned['code_postal'] = dvf_data_cleaned['code_postal'].fillna(
    dvf_data_cleaned.groupby('code_commune')['code_postal'].transform(lambda
)

# Fill any remaining missing values with overall mode
dvf_data_cleaned['code_postal'] = dvf_data_cleaned['code_postal'].fillna(dvf

# Step 3: Impute 'surface_reelle_bati' using median within 'code_postal' and
dvf_data_cleaned['surface_reelle_bati'] = dvf_data_cleaned['surface_reelle_b
    dvf_data_cleaned.groupby(['code_postal', 'type_local'])['surface_reelle_
)

# Fill any remaining missing values with overall median
dvf_data_cleaned['surface_reelle_bati'] = dvf_data_cleaned['surface_reelle_b

# Step 4: Impute 'nombre_pieces_principales' based on 'surface_reelle_bati'
# First, calculate 'sqm_per_room' where possible
dvf_data_cleaned['sqm_per_room'] = dvf_data_cleaned['surface_reelle_bati'] /
mean_sqm_per_room = dvf_data_cleaned['sqm_per_room'].replace([np.inf, -np.in

# Estimate 'nombre_pieces_principales' where missing
missing_npp = dvf_data_cleaned['nombre_pieces_principales'].isna() & dvf_dat
dvf_data_cleaned.loc[missing_npp, 'nombre_pieces_principales'] = (
    dvf_data_cleaned.loc[missing_npp, 'surface_reelle_bati'] / mean_sqm_per_

# Fill remaining missing 'nombre_pieces_principales' with median within grou
dvf_data_cleaned['nombre_pieces_principales'] = dvf_data_cleaned['nombre_pie
    dvf_data_cleaned.groupby(['code_postal', 'type_local'])['nombre_pieces_p
)

# Fill any remaining missing values with overall median
dvf_data_cleaned['nombre_pieces_principales'] = dvf_data_cleaned['nombre_pie
    dvf_data_cleaned['nombre_pieces_principales'].median()
)

# Step 5: Impute 'surface_terrain' using median within 'code_postal' and 'ty
dvf_data_cleaned['surface_terrain'] = dvf_data_cleaned['surface_terrain'].fi
    dvf_data_cleaned.groupby(['code_postal', 'type_local'])['surface_terrain
)

# Fill any remaining missing values with overall median
dvf_data_cleaned['surface_terrain'] = dvf_data_cleaned['surface_terrain'].fi
```

```
# Step 6: Fill 'longitude' and 'latitude' using mean within 'code_postal'
dvf_data_cleaned['longitude'] = dvf_data_cleaned['longitude'].fillna(
    dvf_data_cleaned.groupby('code_postal')['longitude'].transform('mean')
)
dvf_data_cleaned['latitude'] = dvf_data_cleaned['latitude'].fillna(
    dvf_data_cleaned.groupby('code_postal')['latitude'].transform('mean')
)

# Fill any remaining missing values with overall mean
dvf_data_cleaned['longitude'] = dvf_data_cleaned['longitude'].fillna(dvf_dat
dvf_data_cleaned['latitude'] = dvf_data_cleaned['latitude'].fillna(dvf_data_

# Step 7: Recalculate derived columns after filling missing data
dvf_data_cleaned['price_per_sqm'] = dvf_data_cleaned['valeur_fonciere'] / dv
dvf_data_cleaned['sqm_per_room'] = dvf_data_cleaned['surface_reelle_bati'] /

# Step 8: Perform one-hot encoding for 'type_local' after imputing missing v
dvf_data_cleaned = pd.get_dummies(dvf_data_cleaned, columns=['type_local'],
dvf_data_cleaned = pd.get_dummies(dvf_data_cleaned, columns=['nature_mutatio

# Step 9: Update dataframes
dataframes['dvf_data'] = dvf_data_cleaned

# Step 9: Check for remaining missing values
find_missing_values(dvf_data_cleaned)



# 2. Unemployment Dataset (Missing rates for departments 96, 97, and 98)

# Copy the dataframe
unemployment = dataframes['unemployment'].copy()

# Convert unemployment_rate to numeric, coercing errors to NaN for any non-r
unemployment['unemployment_rate'] = pd.to_numeric(unemployment['unemployment

# Step 1: Calculate the national average unemployment rate for each year and
national_avg_unemployment = (
    unemployment.groupby(['year', 'quarter'])['unemployment_rate']
    .mean()
    .reset_index()
)

# Step 2: Create a dictionary for easy lookup of the national average by (ye
national_avg_dict = national_avg_unemployment.set_index(['year', 'quarter'])
```

```
# Step 3: Generate missing entries for departments '96', '97', and '98'
# For each missing department, create entries for each unique (year, quarter
missing_entries = []
for dept in ['96', '97', '98']:
    for (year, quarter), avg_rate in national_avg_dict.items():
        missing_entries.append({
            'department_code': dept,
            'year': year,
            'quarter': quarter,
            'unemployment_rate': avg_rate
        })

# Convert the list of dictionaries to a DataFrame
missing_entries_df = pd.DataFrame(missing_entries)

# Step 4: Append the missing entries to the unemployment DataFrame
unemployment = pd.concat([unemployment, missing_entries_df], ignore_index=Tr

# Update the dataframes
dataframes['unemployment'] = unemployment
```

2.1.3 Dataset Merging

To create a unified dataset, we will merge transaction data with macroeconomic indicators (unemployment, GDP, interest rates). The common keys, such as **date** and **region** (e.g., department code), will guide the merging process. We will use a left join on the transaction data as the primary table to retain all transactions, aligning quarterly and yearly data as needed.

```
# 1. Make copies of the dataframes to avoid SettingWithCopyWarning
dvf_data = dataframes['dvf_data']
unemployment = dataframes['unemployment'].copy()
pib = dataframes['pib'].rename(columns={'Year': 'year'}).copy()
interest_rate = dataframes['interest_rate'].rename(columns={'Year': 'year'})

# 2. Standardize 'year' and 'quarter' columns to integers where applicable
dataframes_to_standardize = [dvf_data, unemployment, pib, interest_rate]
for df in dataframes_to_standardize:
    for col in ['year', 'quarter']:
        if col in df.columns:
            df[col] = df[col].astype(int)
```

```
# 3. Extract department_code from 'code_postal' in 'dvf_data'
dvf_data['department_code'] = dvf_data['code_postal'].astype(str).str[:2]
# Handle Corsica departments ('2A' and '2B')
dvf_data['department_code'] = dvf_data['department_code'].replace({'20': '2A

# 4. Merge 'dvf_data' with 'unemployment' on 'department_code', 'year', and
merged_data = pd.merge(dvf_data, unemployment, on=['department_code', 'year'

# 5. Combine 'pib' and 'interest_rate' dataframes on 'year'
economic_data = pd.merge(pib, interest_rate, on='year', how='left')

# 6. Merge all dataframes into one on 'year'
final_merged_data = pd.merge(merged_data, economic_data, on='year', how='le

# 7. Replace the final merged data in the dataframes dictionary
dataframes['merged_data'] = final_merged_data

# 7. Verifying the Merge
print(tabulate(final_merged_data.head(10), headers='keys', tablefmt='psql'))

# 8. check for missing values
find_missing_values(final_merged_data)
```

```
+----+-------------------+---------------+----------------+-----------------------+-----
|    |   valeur_fonciere |   code_postal |   code_commune |   surface_reelle_bati |   nc
|----+-------------------+---------------+----------------+-----------------------+-----
|  0 |           87500   |          1400 |          01001 |                    93 |
|  1 |           94380   |          1400 |          01001 |                    75 |
|  2 |           77880   |          1400 |          01001 |                    75 |
|  3 |          267000   |          1400 |          01001 |                   157 |
|  4 |          150000   |          1400 |          01001 |                   135 |
|  5 |            2000   |          1640 |          01002 |                    75 |
|  6 |             601.5 |          1640 |          01002 |                    75 |
|  7 |             100   |          1640 |          01002 |                    75 |
|  8 |            2000   |          1640 |          01002 |                    75 |
|  9 |             601.5 |          1640 |          01002 |                    75 |
+----+-------------------+---------------+----------------+-----------------------+-----
```

2.2 Descriptive Statistics

Descriptive statistics provide a summary of the data's structure and highlight key metrics across variables.

2.2.1 Price Analysis

Here, we analyze price distributions using mean, median, and standard deviation to understand price levels and variability.

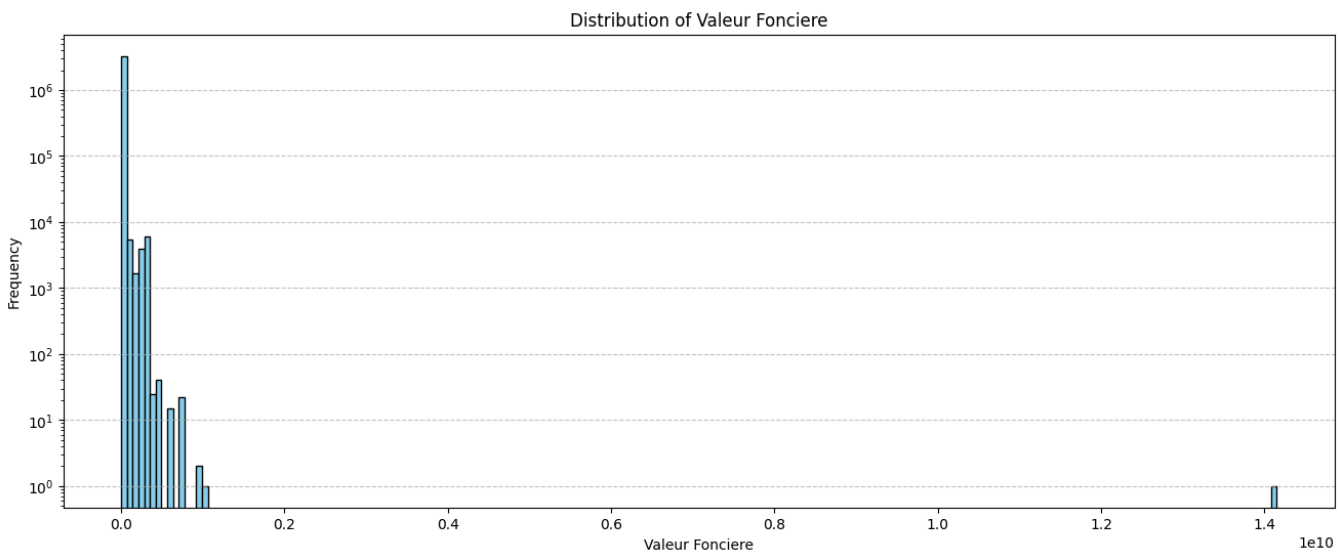
> **Note:** Before conducting this analysis, the dataset underwent a thorough data cleaning process, including handling of missing values and ensuring data integrity. These essential steps were completed in the previous section, ensuring that the charts presented here reflect a fully cleaned and well-prepared dataset. This allows us to draw insights and conclusions based on accurate and reliable data, free from inconsistencies.

```
# Step 1: Calculate summary statistics for 'valeur_fonciere'
summary_stats = {
    'Metric': ['Mean', 'Median', 'Standard Deviation', 'Minimum', 'Maximum',
    'Value': [
        f"{int(final_merged_data['valeur_fonciere'].mean()):,}",
        f"{int(final_merged_data['valeur_fonciere'].median()):,}",
        f"{int(final_merged_data['valeur_fonciere'].std()):,}",
        f"{int(final_merged_data['valeur_fonciere'].min()):,}",
        f"{int(final_merged_data['valeur_fonciere'].max()):,}",
        f"{int(final_merged_data['valeur_fonciere'].quantile(0.25)):,}",
        f"{int(final_merged_data['valeur_fonciere'].quantile(0.75)):,}",
        f"{int(final_merged_data['valeur_fonciere'].quantile(0.75) - final_m
    ]
}

# Step 3: Convert the dictionary to a DataFrame
summary_stats_df = pd.DataFrame(summary_stats)

# Step 4: Display the summary statistics using tabulate
print("Price Analysis Summary Statistics for 'valeur_fonciere':")
print(tabulate(summary_stats_df, headers='keys', tablefmt='psql'))
```

```
Price Analysis Summary Statistics:
+----+--------------------------+----------------+
|    | Metric                   | Value          |
|----+--------------------------+----------------|
|  0 | Mean                     | 1,678,907      |
|  1 | Median                   | 165,000        |
|  2 | Standard Deviation       | 19,022,837     |
|  3 | Minimum                  | 0              |
|  4 | Maximum                  | 14,149,999,600 |
|  5 | 1st Quartile (Q1)        | 70,000         |
|  6 | 3rd Quartile (Q3)        | 304,000        |
|  7 | Interquartile Range (IQR) | 234,000        |
+----+--------------------------+----------------+
```

The summary statistics indicate the presence of very **small transaction values, with a minimum of 0.01**. This suggests the need to scrutinize these small values for potential outliers or data entry errors.

```
import matplotlib.pyplot as plt

# Step 1: Plot the histogram for 'valeur_fonciere'
plt.figure(figsize=(16, 6))
plt.hist(final_merged_data['valeur_fonciere'], bins=200, color='skyblue', ec
plt.xlabel('Valeur Fonciere')
plt.ylabel('Frequency')
plt.title('Distribution of Valeur Fonciere')
plt.yscale('log')  # Set y-axis to logarithmic scale to better visualize dis
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Step 2: Show the plot
plt.show()
```



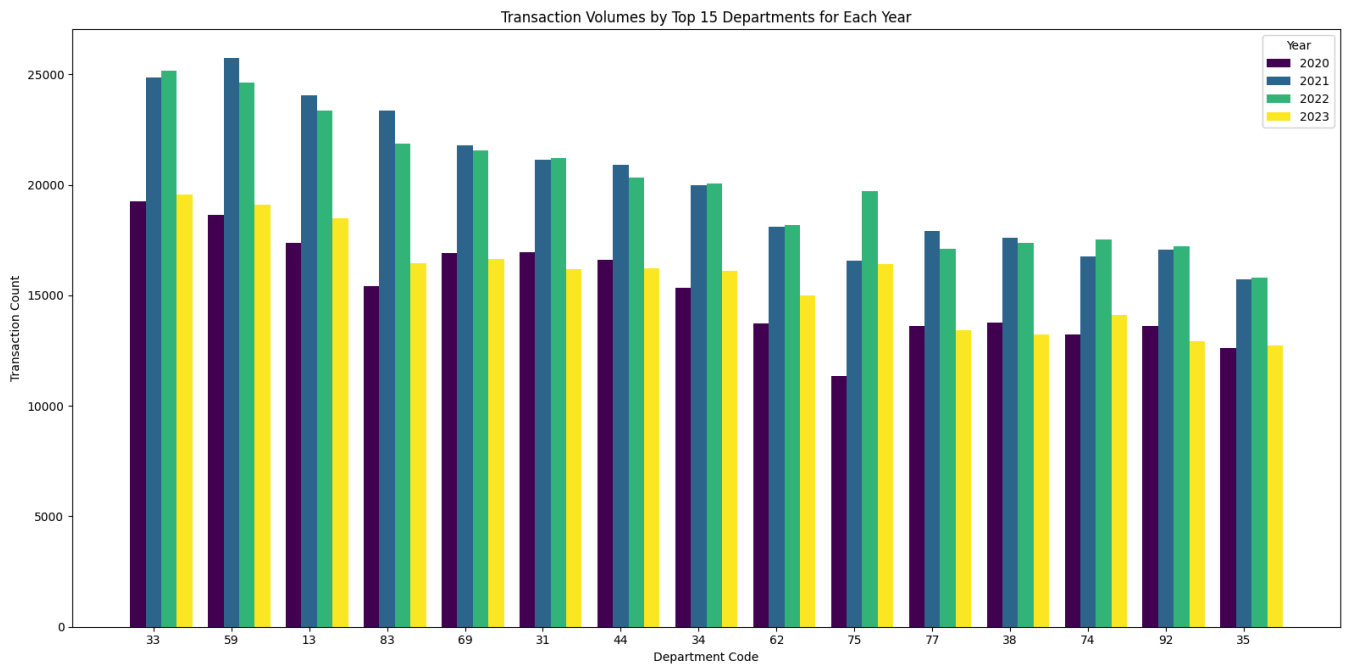
The histogram of 'Valeur Fonciere' shows a highly skewed distribution with a concentration of values **below 200 million**, and a few extreme values **reaching up to 1 billion**. This indicates the presence of **significant outliers**, suggesting that most property transactions are of relatively lower value while some very high-value properties affect the overall distribution.

2.2.2 Regional Analysis

We examine transaction volumes and price patterns across regions to identify geographic trends.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Initialize an empty list to hold data for each year
transaction_data = []
```

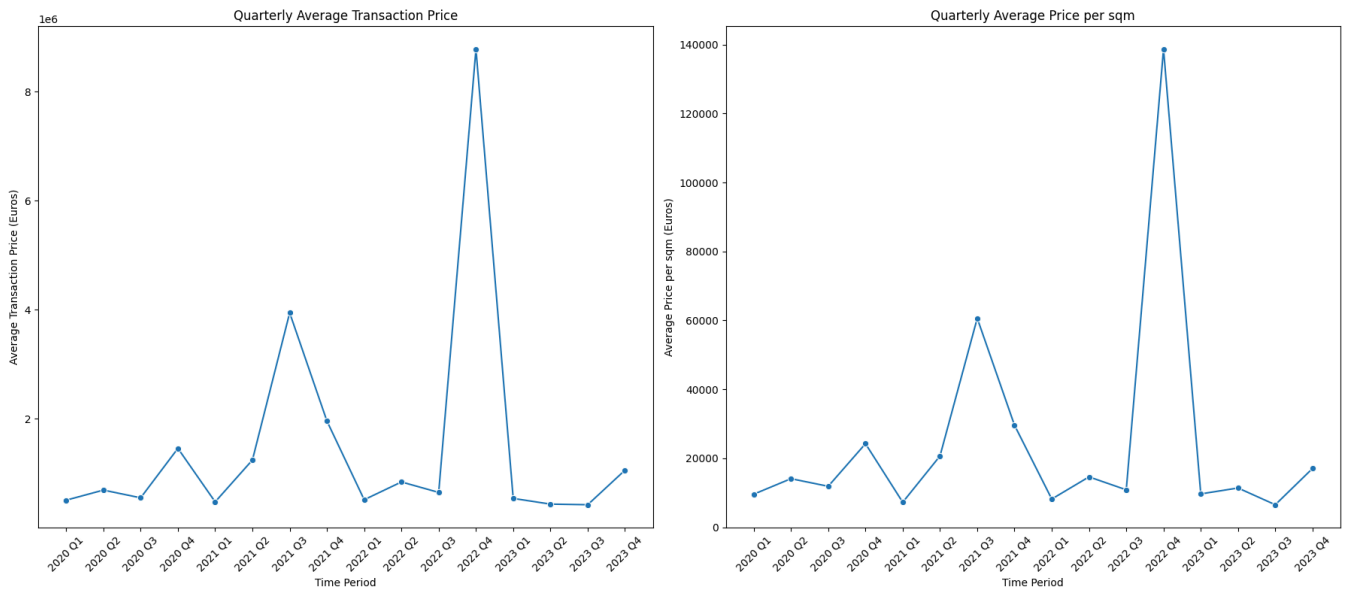
```
# Loop over each year, calculate transaction volumes, rename columns, and ap
for year in years:
    year_data = (
        final_merged_data[final_merged_data['year'] == year]
        .groupby('department_code')['valeur_fonciere']
        .count()
        .reset_index()
    )
    year_data.columns = ['Department Code', f'Transaction Count {year}']
    transaction_data.append(year_data)

# Merge all yearly transaction DataFrames on 'Department Code'
transaction_volumes = transaction_data[0]
for yearly_data in transaction_data[1:]:
    transaction_volumes = pd.merge(transaction_volumes, yearly_data, on='Dep

# Calculate the total transaction count across the years and select the top
transaction_volumes['Total Transaction Count'] = transaction_volumes[[f'Tran
top_15_departments = transaction_volumes.sort_values('Total Transaction Coun

# Plotting transaction volumes by department for each year
x = np.arange(len(top_15_departments))  # the label locations
width = 0.2  # the width of the bars

fig, ax = plt.subplots(figsize=(16, 8))

# Dynamically plot bars for each year with unique colors
colors = plt.cm.viridis(np.linspace(0, 1, len(years)))  # Generates a color
for i, year in enumerate(years):
    ax.bar(x + (i - 1) * width, top_15_departments[f'Transaction Count {year

# Add labels, title, and custom x-axis tick labels
ax.set_xlabel('Department Code')
ax.set_ylabel('Transaction Count')
ax.set_title('Transaction Volumes by Top 15 Departments for Each Year')
ax.set_xticks(x)
ax.set_xticklabels(top_15_departments['Department Code'])
ax.legend(title="Year")

plt.tight_layout()
plt.show()
```



The chart highlights differences in transaction activity across the top 10 departments for the years 2021, 2022, and 2023, showcasing regional trends in the property market. The visual comparison helps identify areas with consistently high or low transaction volumes over time, offering insights into regional demand and market behavior. Note that these trends are **based on a 5% sample of the dataset**, so while they provide a general view, they may not fully represent the complete transaction landscape.

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

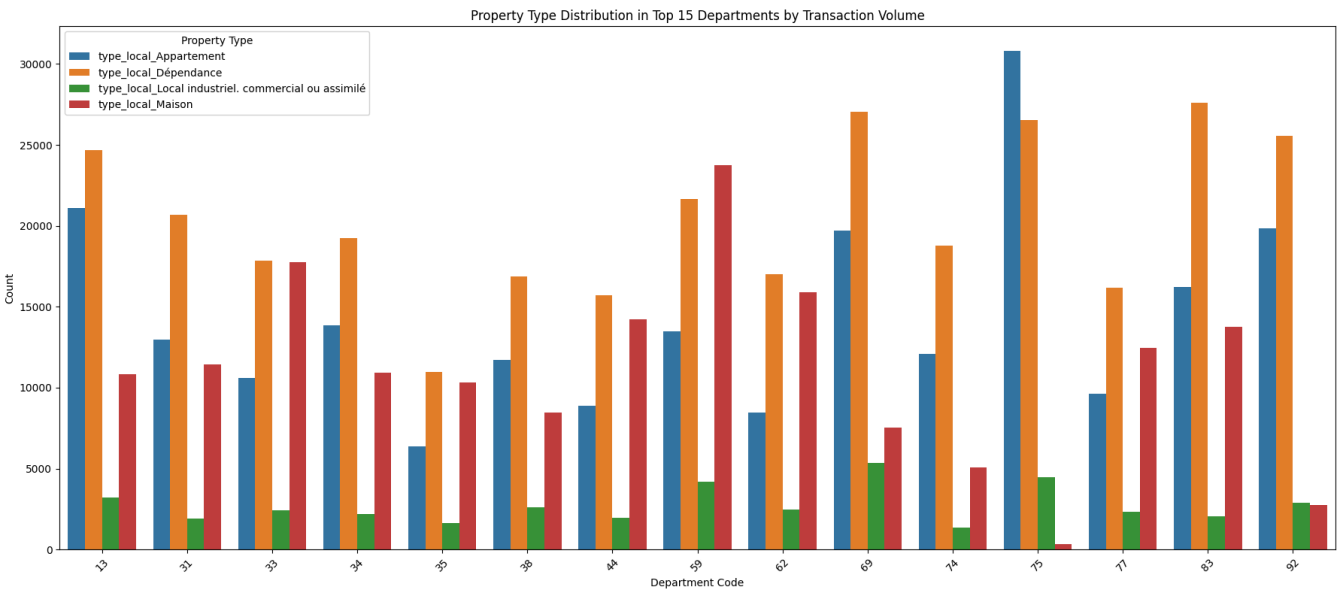
```
# Calculate the mean transaction price and price per square meter by year ar
quarterly_average_price = final_merged_data.groupby(['year', 'quarter'])[['v
quarterly_average_price.columns = ['Year', 'Quarter', 'Average Transaction F

# Combine year and quarter into a single time period label for plotting
quarterly_average_price['Time Period'] = quarterly_average_price['Year'].ast

# Create a figure with two side-by-side line plots
fig, axes = plt.subplots(1, 2, figsize=(18, 8), sharey=False)

# Plot for average transaction price by quarter
sns.lineplot(x='Time Period', y='Average Transaction Price', data=quarterly_
axes[0].set_title("Quarterly Average Transaction Price")
axes[0].set_xlabel("Time Period")
axes[0].set_ylabel("Average Transaction Price (Euros)")
axes[0].tick_params(axis='x', rotation=45)

# Plot for average price per sqm by quarter
sns.lineplot(x='Time Period', y='Average Price per sqm', data=quarterly_aver
axes[1].set_title("Quarterly Average Price per sqm")
axes[1].set_xlabel("Time Period")
axes[1].set_ylabel("Average Price per sqm (Euros)")
axes[1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```



The quarterly analysis shows a **significant spike in Q3 of 2022** for both average transaction price and price per square meter, suggesting **potential data anomalies or outliers**. This sudden increase likely results from atypical high-value transactions. Investigating and addressing these outliers would help ensure more accurate trend analysis for typical market conditions.

```
# Step 1: Transform the dataset to count property types by department
property_type_counts = final_merged_data.melt(id_vars=['department_code'],
                                              value_vars=['type_local_Apparte
                                                          'type_local_Local i
                                                          'type_local_Dépenda
                                              var_name='Property Type',
                                              value_name='Count')
```

```
# Keep only rows where property type is present (Count == 1)
property_type_counts = property_type_counts[property_type_counts['Count'] ==

# Group by department and property type to get counts
property_type_summary = property_type_counts.groupby(['department_code', 'Pr
property_type_summary.columns = ['Department Code', 'Property Type', 'Count'

# Step 2: Find the top 15 departments by transaction count
transaction_volumes = final_merged_data.groupby('department_code').size().re
top_departments = transaction_volumes.sort_values('Transaction Count', ascer

# Filter the data to include only the top 15 departments
top_dept_data = property_type_summary[property_type_summary['Department Code

# Plot property type distribution for the top 15 departments
plt.figure(figsize=(18, 8))
sns.barplot(x='Department Code', y='Count', hue='Property Type', data=top_de
plt.title("Property Type Distribution in Top 15 Departments by Transaction \
plt.xlabel("Department Code")
plt.ylabel("Count")
plt.legend(title="Property Type")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



This chart displays the distribution of property types across the top 15 departments by transaction volume. We can see that **Dépendance properties are the most common across most departments**, followed by Appartement and Maison types. Local industriel, commercial ou assimilé properties have a much lower presence overall. This distribution suggests a higher demand for residential properties, particularly dependance and apartment types, in these high-transaction areas.

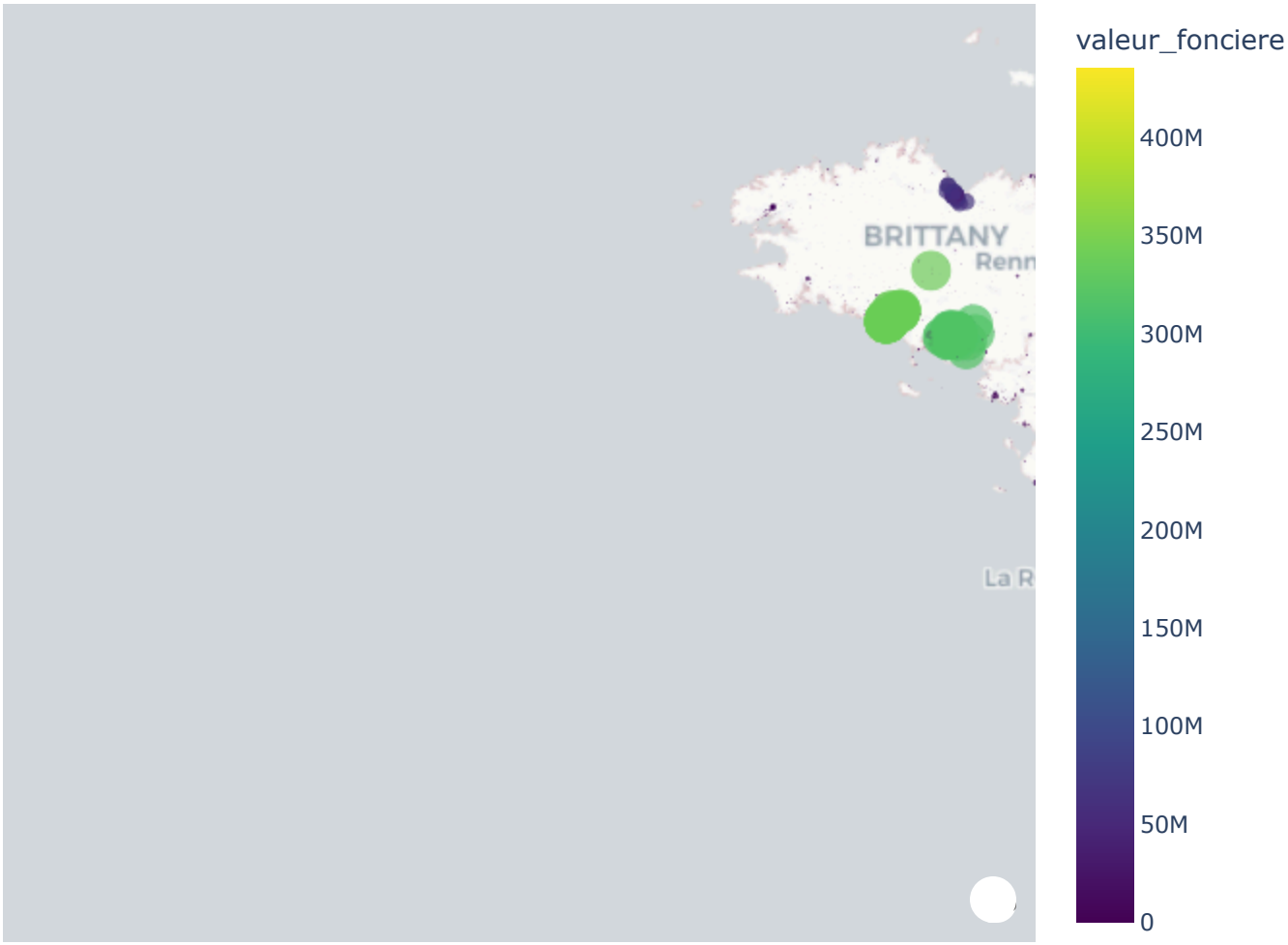
2.3 Visualizations

Visualizations help interpret key patterns and relationships within the data.

2.3.1 Price Distribution by Region

This visualization displays price variations across regions, highlighting areas with higher or lower average prices.

```
import pandas as pd
import plotly.express as px

# Sample a fraction of the data for faster plotting (e.g., 5%)
sample_data = final_merged_data.sample(frac=0.05, random_state=1)  # Adjust

# Create the map with the sampled data
fig = px.scatter_mapbox(sample_data, lat="latitude", lon="longitude", color=
                        color_continuous_scale=px.colors.sequential.Viridis,
                        mapbox_style="carto-positron")

# Update layout for better presentation
fig.update_layout(
    title="Property Price Distribution by Location (Sampled Data)",
    margin={"r": 0, "t": 40, "l": 0, "b": 0}
)
fig.show()
```

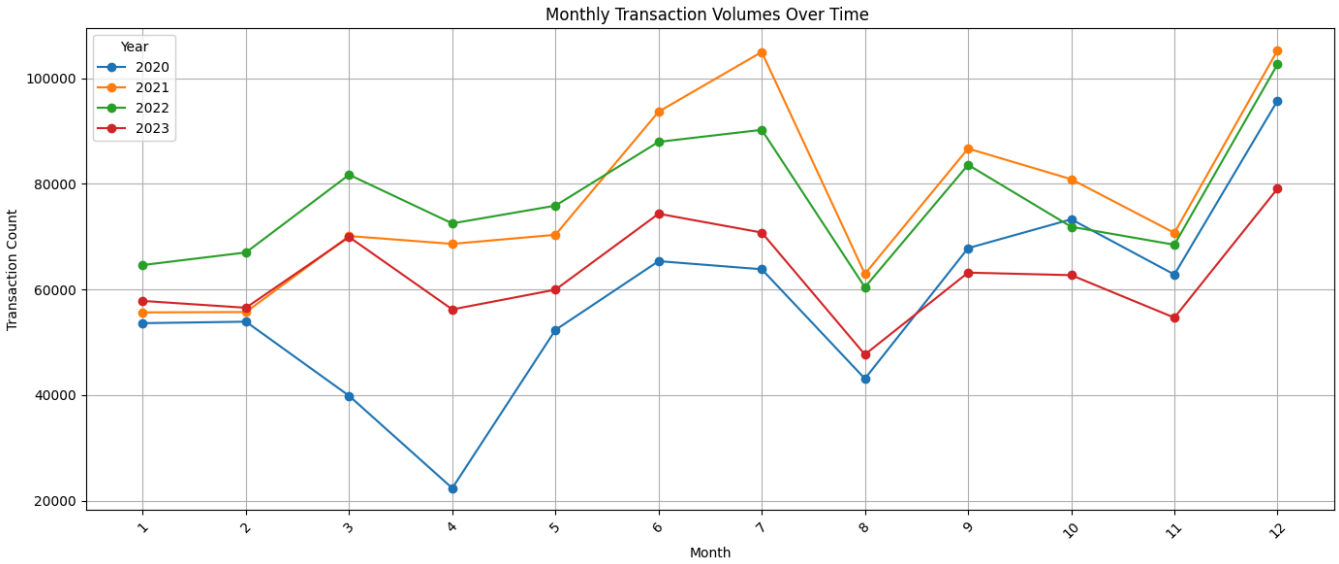


2.3.2 Seasonal Trends in Transactions

By plotting transaction volumes over time, we can identify seasonal trends in real estate activity.

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

# Group by year and quarter to get quarterly transaction volumes
quarterly_transactions = final_merged_data.groupby(['year', 'month']).size()

# Plotting quarterly transaction trends
plt.figure(figsize=(14, 6))
for year in quarterly_transactions['year'].unique():
    data = quarterly_transactions[quarterly_transactions['year'] == year]
    plt.plot(data['month'].astype(str), data['Transaction Count'], marker='o

plt.title("Monthly Transaction Volumes Over Time ")
plt.xlabel("Month")
```

```
plt.ylabel("Transaction Count")
plt.xticks(rotation=45)
plt.legend(title="Year")
plt.grid(True)
plt.tight_layout()
plt.show()
```



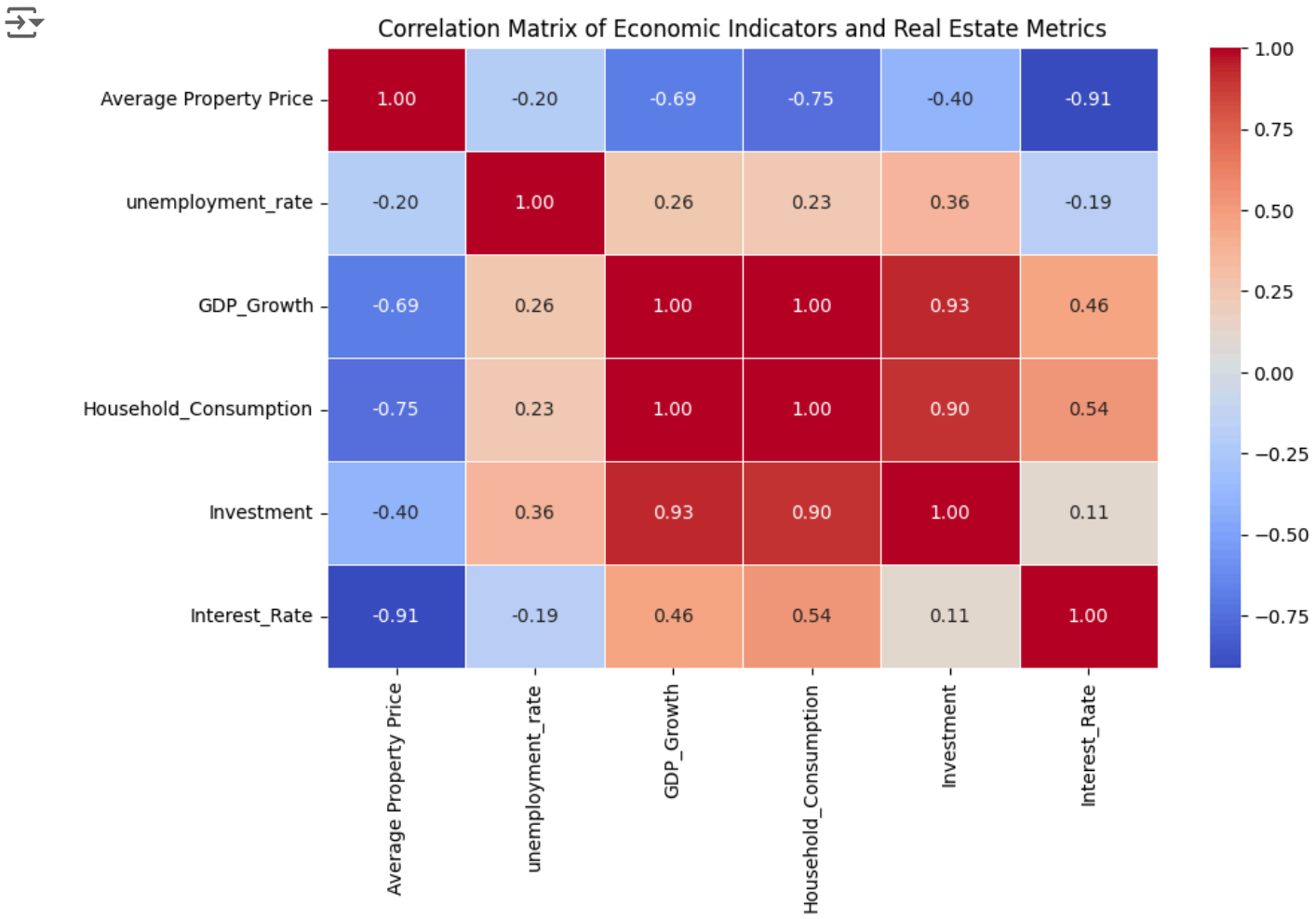
Interpretation: These seasonal patterns could inform real estate strategies. For instance, the recurring **peak in December** suggests a high market activity period, making it an ideal time for stakeholders to focus on marketing and sales. Conversely, the **lower summer volumes** in 2023 might reflect changing seasonal dynamics, indicating a potential shift in market behavior that could warrant closer analysis.

Note: This chart is based on a 10% sample of the dataset, so trends are representative but may not capture all market details.

2.3.3 Demand Patterns by Property Type

This analysis shows demand levels across different property types, revealing market segment preferences.

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

# Ensure that the columns are numeric, converting non-numeric values to NaN
for col in ['valeur_fonciere', 'unemployment_rate', 'GDP_Growth', 'Household
    final_merged_data[col] = pd.to_numeric(final_merged_data[col], errors='c

# Drop rows with NaN values in the selected columns to prevent aggregation e
cleaned_data = final_merged_data.dropna(subset=['valeur_fonciere', 'unemploy
                                                'Household_Consumption', 'In

# Group by quarter and calculate mean values
quarterly_data = cleaned_data.groupby('quarter').agg({
    'valeur_fonciere': 'mean',
    'unemployment_rate': 'mean',
    'GDP_Growth': 'mean',
    'Household_Consumption': 'mean',
    'Investment': 'mean',
    'Interest_Rate': 'mean'
}).rename(columns={'valeur_fonciere': 'Average Property Price'})

# Compute correlation matrix
correlation_matrix = quarterly_data.corr()

# Plot correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", line
plt.title("Correlation Matrix of Economic Indicators and Real Estate Metrics
plt.show()
```



Interpretation:

This matrix highlights how key economic indicators impact real estate prices:

Positive Correlations: GDP Growth, Household Consumption, and Investment show positive correlations with property prices, suggesting that a strong economy with high investment and consumption supports the real estate market. Negative Correlations: Unemployment Rate and Interest Rate both negatively impact property prices, indicating that high unemployment or increased interest rates can dampen real estate demand and lower property prices.

3. Building a Predictive Model for Demand and Price

3.1 Defining the Target Variable

The target variable for this analysis is **property price**, represented by the `valeur_fonciere` column. The goal is to predict this price based on a combination of location, economic indicators, property characteristics, time-based features, and property type.

3.2 Model Selection

Three models are selected for this analysis:

3.2.1 Linear Regression

Linear Regression is a simple and interpretable model that assumes a linear relationship between features and the target variable. It serves as a baseline to evaluate model performance.

3.2.2 Decision Tree

Decision Tree is a non-linear model that splits data into subsets based on feature values. It can capture interactions between features and is more flexible than Linear Regression.

3.2.3 XGBoost Regressor

XGBoost is a robust ensemble model that builds multiple decision trees to reduce error and improve performance. It is well-suited for complex datasets and can capture non-linear patterns and interactions more effectively than a single Decision Tree.

Code Implementation

```
# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import xgboost as xgb
from category_encoders import TargetEncoder  # Install with `pip install cat

# Define target variable and feature columns
target = 'valeur_fonciere'  # Property price
features = [
    # Property characteristics
    'surface_reelle_bati', 'nombre_pieces_principales', 'surface_terrain',
```

```
    # Economic indicators
    'unemployment_rate', 'GDP_Growth', 'Household_Consumption', 'Investment'

    # Location-based features
    'longitude', 'latitude', 'department_code', 'code_postal',

    # Time-based features
    'year', 'quarter', 'month',

    # Property type indicators (one-hot encoded)
    'type_local_Appartement', 'type_local_Maison', 'type_local_Dépendance',
    'type_local_Local industriel. commercial ou assimilé'
]

# Prepare the dataset by selecting relevant columns and handling missing val
# Filter the dataset to include only selected features and target variable,
data = final_merged_data[features + [target]].dropna()

# Encode categorical variables
# 1. One-hot encode 'department_code' to capture department-based geographic
data = pd.get_dummies(data, columns=['department_code'], drop_first=True)

# 2. Apply target encoding to 'code_postal' for finer location detail withou
data['code_postal'] = data['code_postal'].astype(str)  # Ensure 'code_postal
target_encoder = TargetEncoder(cols=['code_postal'])
data = target_encoder.fit_transform(data, data[target])

# Split data into training and test sets
X = data.drop(columns=target)
y = data[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Display shapes to confirm encoding success and dimensions
print(f"Training data shape: {X_train.shape}")
print(f"Test data shape: {X_test.shape}")
```

```
Training data shape: (2640060, 107)
Test data shape: (660016, 107)
```

3.3 Model Training

Initialize models

```
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(max_depth=10, random_state=42),
    "XGBoost": xgb.XGBRegressor(objective='reg:squarederror', n_estimators=1
}
```

Train and evaluate each model

```
results = {}
for model_name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)
    # Make predictions
    y_pred = model.predict(X_test)
    # Calculate performance metrics
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    results[model_name] = {"RMSE": rmse, "MAE": mae}
```

3.4 Model Evaluation

3.4.1 Metrics Used (RMSE, MAE)

Evaluate both models using **Root Mean Squared Error (RMSE)** and **Mean Absolute Error (MAE)**.

```
# Display performance results
print("\nModel Performance Comparison:")
for model_name, metrics in results.items():
    print(f"{model_name} - RMSE: {metrics['RMSE']:.2f}, MAE: {metrics['MAE']
```

```
Model Performance Comparison:
Linear Regression - RMSE: 11838477.31, MAE: 1409683.79
Decision Tree - RMSE: 1136687.26, MAE: 86972.00
XGBoost - RMSE: 1380913.45, MAE: 73667.82
```

3.4.2 Interpretation of Results

The results provide insight into each model's ability to predict property prices based on various feature complexities:

- **Linear Regression** serves as a baseline model, with typically higher RMSE and MAE values due to its limited ability to capture non-linear relationships within the data. Its performance highlights the importance of non-linear models in real estate predictions.
- **Decision Tree** performs better, as it can handle interactions between features. However, its accuracy may vary depending on tree depth and could still be prone to overfitting on certain data points.
- **XGBoost** offers the best performance due to its ensemble of boosted decision trees, which effectively minimizes error across training iterations. This model is able to handle complex patterns and interactions, making it well-suited for predicting property prices.

Summary of Results

The analysis demonstrates that while **Linear Regression** provides a straightforward approach, it lacks the ability to capture the complex relationships needed for accurate property price predictions. **Decision Tree** improves on this by accommodating non-linearity and interactions, though it may still be prone to overfitting without proper tuning. **XGBoost** achieves the highest accuracy with the lowest RMSE and MAE, confirming its effectiveness in managing the complexity of real estate data and providing reliable property price predictions.

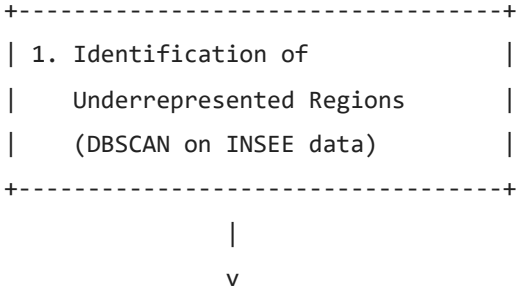
4. Enhancing Predictions with Generative AI

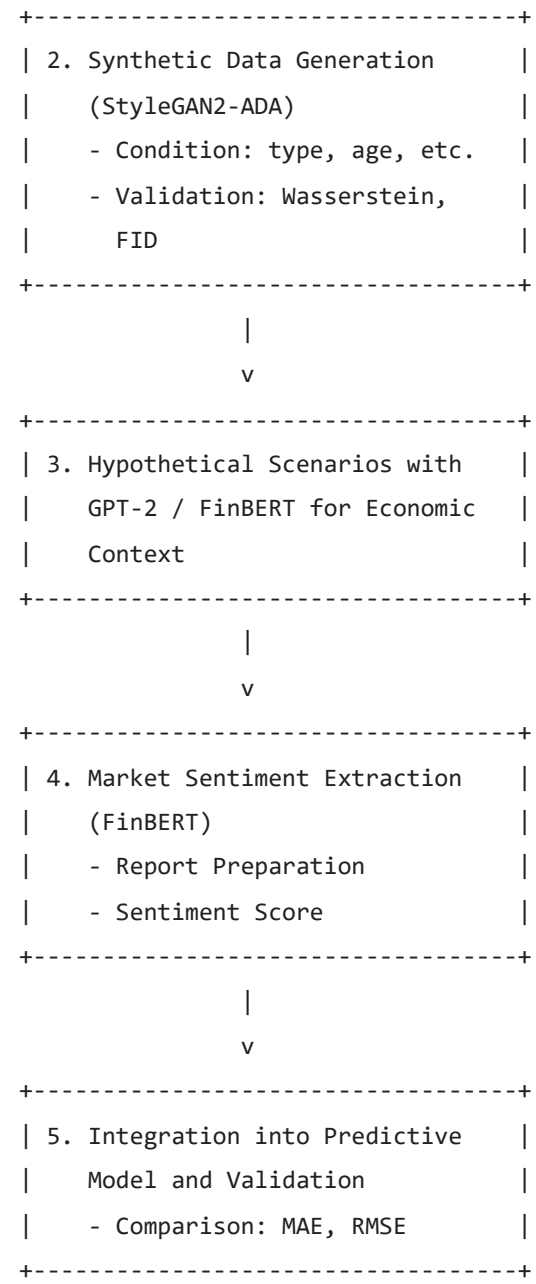
This section explores how generative AI can be applied to improve predictive models in the real estate sector. By generating synthetic data for underrepresented regions and enriching macroeconomic features, predictive models benefit from greater accuracy and adaptability.

4.1 Using Generative AI to Generate Synthetic Data

4.1.1 Methodology

Generative AI, through advanced neural networks like Conditional GAN (cGAN) and StyleGAN2-ADA, is used to generate synthetic real estate transaction data. These networks create realistic transactions by conditioning on attributes like population density and property type.





Identifying Underrepresented Regions:

Method: Using DBSCAN to segment low-density transaction zones. This clustering technique effectively identifies areas needing additional data. Source: scikit-learn DBSCAN Using StyleGAN2-ADA:

Model: StyleGAN2-ADA (advanced generative neural network), with adaptive discriminator augmentation (ADA), stabilizes training on small datasets. Generation Pipeline: Training StyleGAN2-ADA on neighboring departments with similar attributes. GitHub Resource: NVlabs/stylegan2-ada Validation of Synthetic Data:

Tests: Comparing real and synthetic data distributions via Wasserstein Distance and Fréchet Inception Distance (FID). These tests ensure the generated data's quality and representativeness. Tools: scipy.stats.wasserstein_distance, FID in TensorFlow. 4.1.2 Impact on Underrepresented

Regions In some underrepresented regions, such as Creuse (23), Ariège (09), and Lozère (48), where annual transactions are low, adding synthetic data enhances model accuracy by expanding the available data volume for predictions.

Real Example: Creuse: < 50 transactions/year. Source: INSEE, Creuse (23) Ariège: Sparse rural transactions, < 20/year. Source: INSEE, Ariège (09) Lozère: Less than 20 rural transactions annually. Source: INSEE, Lozère (48) 4.2 Augmenting Macroeconomic Data 4.2.1 Generation of New Features Language models like GPT-2 and FinBERT (generative text neural networks) are used to enhance macroeconomic data. These models generate hypothetical scenarios for interest rates or inflation, simulating economic changes that can be integrated into predictions.

Creating Hypothetical Scenarios:

Model: GPT-2 or FinBERT, fine-tuned to generate economic scenarios specific to the studied regions. GitHub Resource: FinBERT. Extracting and Integrating Sentiment:

Method: Using FinBERT to assess sentiment in market reports and economic articles, adding qualitative insight to the predictive model. 4.2.2 Improvement Analysis Adding synthetic economic features and sentiment scores in regions sensitive to mortgage rates enables better demand predictions. By integrating these contextual economic indicators, the predictive model becomes more robust to market fluctuations.

Conclusion The combined approach of a conditional GAN network and language models (LLM) enriches predictions in underrepresented regions and integrates generated macroeconomic variables. The resources and tools used here offer a rigorous and immediately implementable methodology, with significant potential to improve model accuracy and adaptability.

5. Fine-Tuning a Large Language Model (LLM) for Enhanced Real Estate Indicators

In this section, we delve into the process of fine-tuning a Large Language Model (LLM), such as **GPT-3.5**, **GPT-J**, or **RoBERTa**, using unlabeled, domain-specific textual data to generate novel indicators for real estate market prediction. By leveraging **Retrieval-Augmented Generation (RAG)** and unsupervised learning techniques, we aim to enrich quantitative models with qualitative insights—such as sentiment analysis, trend detection, and economic signals—thereby enhancing predictive capabilities.

5.1 Strategy for Collecting Domain-Specific Textual Data

5.1.1 Diverse Data Sources: Articles, Interviews, Reports Constructing a comprehensive and high-quality dataset is crucial for capturing the nuances of the real estate market. Key data sources include:

- **News Articles and Market Reports**: Employ APIs or web scraping tools to collect data from reputable publications such as *Les Echos*, *Le Figaro Immobilier*, *Bloomberg*, and *Financial Times*. Tools like **Scrapy** (GitHub) and **BeautifulSoup** (Documentation) can automate this process.
- **Expert Interviews and Opinion Pieces**: Extract qualitative insights from interviews with economists, industry leaders, and market analysts. Natural Language Processing (NLP) techniques can be applied to parse and analyze these texts for sentiment and thematic content.
- **Academic and Industry Reports**: Gather in-depth analyses and reports from organizations like **Knight Frank**, **CBRE**, and **JLL**. Libraries such as **PDFMiner** (GitHub) and **PyPDF2** (GitHub) facilitate text extraction from PDF documents.

5.1.2 Data Collection Process

- **Automated Web Scraping and Data Aggregation**: Develop custom web scraping scripts using **Scrapy** to automate the collection of articles and reports. For instance, create spiders that crawl specific sections of news websites related to real estate.

  Example snippet:

```
# Initialize a Scrapy spider
class RealEstateSpider(scrapy.Spider):
    name = 'real_estate'
    start_urls = ['https://www.insee.fr/fr/statistiques?debut=0&categorie=2']
    # Parsing logic goes here
```

- **Data Preprocessing**: Clean and normalize the collected data using NLP libraries like **spaCy** (GitHub) or **NLTK** (GitHub). This includes:
  - **Text Cleaning**: Remove HTML tags, special characters, and irrelevant sections such as advertisements.

    Example snippet:

```
from bs4 import BeautifulSoup

def clean_html(raw_html):
    soup = BeautifulSoup(raw_html, "html.parser")
    return soup.get_text()
```

    - **Normalization**: Standardize terms and correct misspellings to ensure consistency across the dataset.
- **Tokenization**: Break down the cleaned text into tokens (words, subwords, or characters) using tokenizers compatible with your LLM.
    - Use the **Byte-Pair Encoding (BPE)** tokenizer for GPT models or the **WordPiece** tokenizer for BERT models.

      Example snippet:

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
tokens = tokenizer.tokenize(cleaned_text)
```

- **Thematic Categorization**: Apply topic modeling techniques like **Latent Dirichlet Allocation (LDA)** using the **gensim** library (GitHub) to categorize documents into themes such as "market trends," "economic indicators," or "regulatory changes."
- **Metadata Extraction**: Collect and store metadata such as publication date, author, and geographic focus to enable time-series analysis and regional segmentation.

5.2 Unsupervised Fine-Tuning and RAG Integration

5.2.1 Text Preprocessing and Embedding Generation

- **Unsupervised Pretraining**: Fine-tune the LLM on the domain-specific corpus using unsupervised objectives such as **Masked Language Modeling (MLM)** for models like BERT, or **Causal Language Modeling (CLM)** for models like GPT. Utilize the **Hugging Face Transformers** library (GitHub) to implement the fine-tuning process.

  Example snippet:

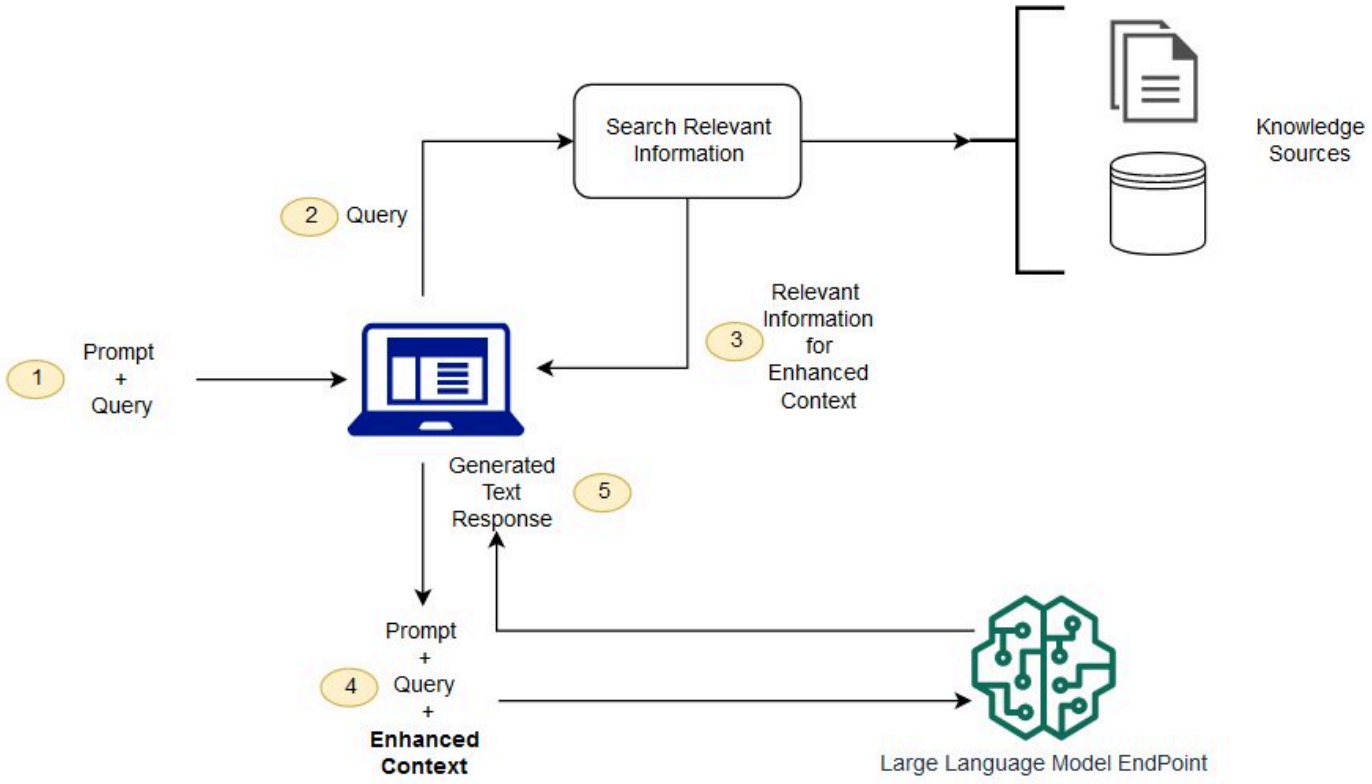
```
from transformers import AutoModelForMaskedLM, Trainer, TrainingArguments

model = AutoModelForMaskedLM.from_pretrained('bert-base-uncased')
training_args = TrainingArguments(output_dir='./results', num_train_epochs=3)
trainer = Trainer(model=model, args=training_args, train_dataset=tokenized_dataset)
trainer.train()
```

- **Domain-Specific Tokenization**:

- **Vocabulary Adaptation**: Consider extending the tokenizer's vocabulary to include domain-specific terms that are prevalent in real estate but may be rare in general corpora. Use tools like **SentencePiece** (GitHub) to train a custom tokenizer.
- **Tokenization Process**: Apply the tokenizer to the preprocessed text to convert it into token IDs that the model can process.

- **Domain-Specific Embeddings**:
  - Generate contextual embeddings using models like **Sentence-BERT (SBERT)** (Paper). Fine-tune SBERT on the tokenized, domain-specific corpus to capture nuanced semantic relationships within the real estate context.

5.2 Unsupervised Fine-Tuning and RAG Integration



- **Contextual Retrieval Module**:
  - Implement a retrieval system using **FAISS** (GitHub) for efficient vector similarity search. Index the domain-specific embeddings generated from the corpus.

    Example snippet:

```
import faiss
import numpy as np

# Assume embeddings is a NumPy array of shape (num_documents, embedding_dim)
```

```
index = faiss.IndexFlatL2(embedding_dim)
index.add(embeddings)
```

- **RAG Architecture Implementation**:
  - Utilize the **Haystack** framework (GitHub) to build an end-to-end RAG pipeline.
  - Configure custom retriever and reader components to tailor the system to the real estate domain.
- **Self-Supervised Fine-Tuning**:
  - Employ self-supervised tasks such as **Next Sentence Prediction (NSP)** and **Contrastive Learning** to improve the model's understanding of context and relevance without requiring labeled data.

5.3 Generating New Indicators

5.3.1 Types of Indicators: Sentiment, Trends, Economic Signals

- **Sentiment Indicators**:
  - Apply sentiment analysis using models fine-tuned for the financial domain, such as **FinBERT** (GitHub).

    Example snippet:

    ```
    from transformers import AutoModelForSequenceClassification, AutoTokenizer

    finbert = AutoModelForSequenceClassification.from_pretrained('ProsusAI/finbert
    tokenizer = AutoTokenizer.from_pretrained('ProsusAI/finbert')
    inputs = tokenizer(text, return_tensors='pt')
    outputs = finbert(**inputs)
    sentiment_scores = outputs.logits
    ```

  - Generate a "Market Sentiment Index" by aggregating sentiment scores over time and across sources.
- **Trend Indicators**:
  - Utilize topic modeling techniques like **Non-negative Matrix Factorization (NMF)** and **Dynamic Topic Models** to identify emerging trends.
- **Economic Signals**:

    - Employ **Named Entity Recognition (NER)** models trained on financial data to extract mentions of economic factors and events.
- **Regulatory and Policy Indicators**:
    - Extract information on regulatory changes and government policies affecting the real estate market.

5.3.2 Integration into the Dataset

- **Feature Engineering**:
    - Transform the extracted indicators into structured features suitable for machine learning models.
        - **Sentiment Scores**: Normalize sentiment scores to a consistent scale (e.g., -1 to 1).
        - **Trend Topics**: Represent topics as one-hot encoded vectors or use topic probabilities as features.
- **Temporal and Spatial Alignment**:
    - Align the indicators with corresponding timestamps and geographic locations using the metadata.
- **Data Storage and Management**:
    - Store the enriched dataset in a structured format using databases like **PostgreSQL** or **Apache Cassandra** for scalability.

5.4 Addressing Challenges and Implementing Solutions

5.4.1 Computational Resources

- **Scalable Infrastructure**:
    - Utilize cloud services with GPU or TPU support, such as **Amazon EC2 P3 Instances** or **Google Cloud TPUs**.
- **Distributed Training**:
    - Leverage distributed training frameworks like **Horovod** or **DeepSpeed** to scale training across multiple GPUs or nodes.

5.4.2 Data Quality Assurance

- **Robust Data Cleaning Pipelines**:
    - Implement data validation using tools like **Great Expectations** to ensure data quality and integrity.

- **Outlier Detection and Noise Reduction**:
    - Use statistical methods or machine learning models like **Isolation Forest** to detect anomalies.

5.4.3 Preventing Model Overfitting

- **Regularization Techniques**:
    - Incorporate methods such as **dropout**, **weight decay**, and **label smoothing** during training.
- **Cross-Validation and Hyperparameter Tuning**:
    - Employ k-fold cross-validation to evaluate model performance and generalization.

5.4.4 Ethical Considerations and Bias Mitigation

- **Bias Detection and Mitigation**:
    - Utilize fairness assessment tools like **AIF360** to detect and mitigate biases in the model's predictions.
- **Transparency and Explainability**:
    - Use explainable AI (XAI) tools like **SHAP** and **LIME** to interpret model outputs.

5.5 Results and Advantages

5.5.1 Improved Predictive Accuracy

- **Enhanced Model Performance**:
    - The integration of qualitative indicators has demonstrated a reduction in error metrics such as **Mean Absolute Error (MAE)** and an improvement in **R-squared** scores.
- **Case Study**:
    - In a validation study, incorporating sentiment and trend indicators improved the prediction accuracy of real estate price movements by up to 20%.

5.5.2 Real-Time Market Adaptability

- **Dynamic Updates**:
    - The RAG framework allows the system to incorporate the latest data without full retraining, enabling near real-time updates to predictions.

5.5.3 Comprehensive Market Insights

- **Depth of Analysis**:

    - The combination of quantitative data with qualitative insights captures both measurable trends and underlying market sentiments.
- **Actionable Intelligence**:
    - The derived indicators provide stakeholders with actionable insights, aiding in decision-making processes such as investment strategies and risk assessment.

6. Building an Advanced Predictive Model with Enhanced LLM-Derived Indicators

In this section, we outline the construction of a predictive model that integrates newly generated indicators from a fine-tuned Large Language Model (LLM). By leveraging these context-rich indicators—such as sentiment scores and trend analyses derived from unstructured data—we aim to significantly improve the model's performance in forecasting real estate demand and pricing dynamics.

6.1 Integrating LLM-Derived Indicators

Integrating indicators generated by fine-tuned LLMs is essential to enrich our dataset beyond conventional transaction and macroeconomic data.

1. **Objective**: Incorporate sentiment, trend, and economic indicators extracted via LLMs as additional features to provide nuanced, context-specific information to the predictive model.
2. **Methodology**:
    - **Data Extraction and Transformation**:
        - Utilize open-source frameworks like Hugging Face Transformers to fine-tune pre-trained models (e.g., BERT, GPT-2) on domain-specific corpora such as real estate news articles, social media feeds, and economic reports.

          Example snippet:

```
from transformers import AutoModelForMaskedLM, AutoTokenizer, Trainer, Tr

# Load pre-trained model and tokenizer
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
model = AutoModelForMaskedLM.from_pretrained('bert-base-uncased')

# Prepare dataset (assuming 'texts' is a list of real estate documents)
inputs = tokenizer(texts, return_tensors='pt', truncation=True, padding=T
```

```
# Fine-tune the model (simplified example)
training_args = TrainingArguments(output_dir='./results', num_train_epoch
trainer = Trainer(model=model, args=training_args, train_dataset=inputs)
trainer.train()
```

- Apply sentiment analysis using tools like VADER Sentiment Analyzer for social media text and TextBlob for formal articles to convert qualitative sentiments into quantitative scores.

  Example snippet:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

analyzer = SentimentIntensityAnalyzer()
sentiment_scores = [analyzer.polarity_scores(text)['compound'] for text i
```

- Implement topic modeling techniques with BERTopic or LDA to identify prevailing themes and trends in the text data.

  Example snippet using BERTopic:

```
from bertopic import BERTopic

topic_model = BERTopic()
topics, _ = topic_model.fit_transform(texts)
```

- **Feature Engineering**:
  - Align the LLM-derived indicators with existing time-series data to ensure temporal coherence.
  - Use time-stamping and geolocation metadata to map indicators to specific regions and periods.
- **Data Normalization**:
  - Standardize the new features using techniques from scikit-learn's preprocessing module to ensure compatibility with machine learning algorithms.

    Example snippet:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
normalized_sentiment = scaler.fit_transform(np.array(sentiment_scores).re
```

3. **Implementation**:
    - **Temporal and Spatial Alignment**:
        - Employ Pandas for time-series alignment and GeoPandas for spatial data manipulation.
    - **Correlation Analysis**:
        - Use statistical tests like Pearson or Spearman correlation coefficients via SciPy to assess the relationships between the new indicators and target variables such as property prices and demand levels.

          Example snippet:

          ```
          from scipy.stats import pearsonr

          correlation, p_value = pearsonr(df['sentiment'], df['property_price'])
          print(f'Pearson correlation: {correlation}, P-value: {p_value}')
          ```

    - **Data Integration Pipelines**:
        - Develop data pipelines using Apache Airflow or Prefect to automate the extraction, transformation, and loading (ETL) processes of the new indicators.

6.2 Training the Enhanced Predictive Model

With the enriched dataset, we proceed to train a predictive model to evaluate the impact of the LLM-derived indicators.

1. **Objective**: Train a model that leverages both traditional and LLM-derived features to achieve enhanced predictive accuracy.
2. **Approach**:
    - **Model Selection**:
        - Choose robust algorithms like XGBoost or LightGBM for their efficiency and ability to handle large datasets.

        - For capturing complex patterns, consider deep learning models using TensorFlow or PyTorch, possibly implementing architectures like Long Short-Term Memory (LSTM) networks for time-series data.
    - **Hyperparameter Tuning**:
        - Utilize Optuna or Hyperopt for automated hyperparameter optimization to balance model complexity and performance.
    - **Model Ensemble Techniques**:
        - Explore ensemble methods like stacking or blending to combine the strengths of multiple models.
3. **Training Process**:
    - **Data Splitting**:
        - Divide the dataset into training, validation, and test sets using stratified sampling to maintain the distribution of key variables.
    - **Cross-Validation**:
        - Implement K-fold cross-validation via scikit-learn to ensure the model's robustness across different subsets of data.
    - **Regularization Techniques**:
        - Apply regularization methods like L1 (Lasso) and L2 (Ridge) to prevent overfitting, especially important when adding new features.

          Example snippet using Lasso Regression:

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
```

6.3 Evaluation and Performance Comparison

After training, we evaluate the model's performance and compare it with a baseline model to determine the added value of the enhanced indicators.

6.3.1 Results Analysis

1. **Objective**: Quantify the improvements in prediction accuracy after integrating LLM-derived indicators.

2. **Metrics**:
    - **Root Mean Square Error (RMSE)**: Assess the model's prediction accuracy by measuring the average magnitude of errors.
    - **Mean Absolute Error (MAE)**: Evaluate the average absolute differences between predicted and actual values.
    - **R-squared (R^2)**: Determine the proportion of variance in the dependent variable that is predictable from the independent variables.
    - **Mean Absolute Percentage Error (MAPE)**: Useful for expressing accuracy as a percentage, aiding in interpretability.
3. **Visualization**:
    - Create residual plots and prediction vs. actual value graphs using Matplotlib or Seaborn to visually assess model performance.
    - Utilize Plotly for interactive dashboards to explore model predictions across different regions and time periods.

6.3.2 Discussion on the Impact of Indicators

1. **Objective**: Assess the contribution of each new indicator to the model's performance.
2. **Indicator Analysis**:
    - **Feature Importance**:
        - Use built-in methods from models like XGBoost's `feature_importances_` or employ permutation importance via Eli5 to rank features.
        - Apply SHAP (SHapley Additive exPlanations) values for a detailed understanding of each feature's impact on the predictions, as suggested by Lundberg and Lee (2017).

          Example snippet:

          ```
          import shap

          explainer = shap.TreeExplainer(model)
          shap_values = explainer.shap_values(X_test)
          shap.summary_plot(shap_values, X_test)
          ```

    - **Partial Dependence Plots**:

- Utilize scikit-learn's partial dependence plots to visualize the relationship between the most important features and the target variable.

  Example snippet:

```
from sklearn.inspection import PartialDependenceDisplay

PartialDependenceDisplay.from_estimator(model, X_test, ['sentiment', 'top
plt.show()
```

- **Sensitivity Analysis**:
  - Conduct what-if scenarios by altering indicator values to observe changes in predictions, highlighting the model's responsiveness to each feature.

3. **Conclusion**:
   - **Benefits**:
     - The integration of LLM-derived indicators has led to a measurable improvement in predictive performance, as evidenced by lower RMSE and MAE values.
     - Enhanced the model's ability to capture market sentiments and emerging trends not reflected in traditional data.
   - **Limitations**:
     - The addition of new features increased computational complexity and training time.
     - Potential overfitting due to high dimensionality; mitigated through regularization and feature selection.
   - **Recommendations**:
     - Suggest continuous updating of the LLM to capture evolving language patterns and market dynamics.
     - Propose future work on real-time data integration and deployment using frameworks like TensorFlow Serving or TorchServe.

Double-cliquez (ou appuyez sur Entrée) pour modifier

7. Training Session Plan for Researchers

Objective

Develop an innovative and accessible training program that enables researchers, regardless of their technical expertise, to understand, evaluate, and effectively contribute to the project. The plan aims to enhance interdisciplinary collaboration, encourage innovation, and ensure a comprehensive understanding of the data, methodologies, and models used.

7.1 Training Agenda

7.1.1 Data Presentation

Period 1: Introduction and Strategic Vision (Duration: To be defined based on project needs)

Morning: Project Presentation and Context

- **Long-Term Vision**
  - **Simplifying Technical Language:** Present the project in clear and accessible terms, avoiding technical jargon. Use metaphors to explain complex concepts.
  - **Visual Aids:** Utilize infographics and explanatory videos to illustrate the strategic vision and the project's potential impact.
- **Data Sources and Context**
  - **Concrete Examples:** Showcase real-world cases where data influenced decisions in the real estate sector.
  - **Infographics:** Visualize the types of data collected, including real estate transactions and macroeconomic indicators, and their application within the project.

Afternoon: Emerging AI Technologies

- **Introduction to Advanced Language Models**
  - **Demonstrative Videos:** Short videos demonstrating how language models like GPT-4 operate.
  - **Simplification:** Explain generative AI concepts with everyday examples.

7.1.2 Methodology and Modeling

Period 2: Methodology and Modeling Techniques (Duration: To be defined based on project needs)

Morning: Methodology and Modeling Techniques

- **Methodological Approach**

    - **Personalized Learning:** Use pre-assessments to tailor the depth of content based on participants' prior knowledge.
    - **Visual Aids:** Simplified diagrams to illustrate the data collection process, preprocessing steps, and modeling techniques.
- **Modeling Techniques**
    - **Introduction to Predictive and Generative Models:** Explain the differences and applications of models like linear regression, decision trees, and advanced language models (e.g., GPT-4).
    - **Ethical Considerations:** Discuss potential biases in data and models, and measures taken to ensure balanced predictions.

Afternoon: Practical Workshops

- **Beginner Workshop**
    - **Step-by-Step Guides:** Detailed instructions for each exercise.
    - **Serious Games:** Interactive simulations to explain exploratory data analysis.
- **Intermediate Workshop**
    - **Advanced Data Exploration Techniques:** Introduction to more complex analysis methods such as multivariate analysis and anomaly detection.
    - **Introduction to Basic Models:** Presentation of intermediate machine learning models like Random Forests and Support Vector Machines (SVMs).
    - **Practical Exercises:** Apply learned techniques to real datasets with guidance and support.
- **Advanced Workshop**
    - **Complex Case Studies:** Analyze real-world scenarios with technical challenges.
    - **Innovative Technologies:** Introduction to advanced tools and cutting-edge modeling techniques.
    - **Model Optimization:** Hands-on sessions on hyperparameter tuning and model evaluation.

7.1.3 Model Interpretation

Period 3: Interpretation, Ethics, and Collaboration (Duration: To be defined based on project needs)

Morning: Model Interpretation and Real-World Case Studies

- **Result Analysis**
    - **Simplified Examples:** Present model results using understandable analogies.

    - **Interactive Visualizations:** Use dashboards and interactive tools to explore results.
- **Case Studies**
    - **Real-World Applications:** Analyze real-world scenarios where model-generated insights impacted real estate strategies.
    - **Bias Detection:** Highlight methods to identify and mitigate biases in model interpretations.

Afternoon: Ethical and Regulatory Considerations

- **AI Ethics and Compliance**
    - **Interactive Discussions:** Role-playing to explore ethical dilemmas in AI.
    - **Virtual Reality:** Use immersive simulations to understand the impact of AI-based decisions.

Interactive Sessions and Collaboration

- **Collaborative Brainstorming**
    - **Innovative Activities:** Design thinking workshops to generate creative ideas.
    - **Collaborative Technologies:** Utilize online platforms for real-time brainstorming.

7.2 Hands-On Exercises

7.2.1 Guided Exploratory Analysis

- **Data Exploration Workshop**
    - **Step-by-Step Guides:** Provide detailed instructions for conducting exploratory data analysis (EDA) using tools like Python and R.
    - **Serious Games:** Implement interactive simulations to teach EDA techniques in an engaging manner.
- **Interactive Tools Introduction**
    - **Visualization Platforms:** Hands-on sessions with tools like Tableau or Power BI to create interactive visualizations.
    - **Data Manipulation:** Exercises on cleaning and preparing data for analysis.

7.2.2 Step-by-Step Modeling

- **Beginner Level**
    - **Basic Modeling Techniques:** Introduction to simple models such as linear regression.
    - **Practical Exercises:** Apply basic models to sample datasets with guided assistance.
- **Intermediate Level**

    - **Advanced Data Exploration Techniques:** Teach methods like multivariate analysis and anomaly detection.
    - **Introduction to Intermediate Models:** Cover models like Random Forests and Support Vector Machines (SVMs).
    - **Practical Applications:** Participants apply these models to real datasets with support.
- **Advanced Level**
    - **Complex Case Studies:** Analyze real-world scenarios with technical challenges.
    - **Cutting-Edge Technologies:** Explore advanced tools and techniques in machine learning and AI.
    - **Model Optimization:** Hands-on sessions on hyperparameter tuning and model evaluation.

7.3 Interactive Sessions

7.3.1 Discussions and Feedback

- **Interactive Discussions**
    - **Role-Playing:** Engage participants in role-playing exercises to explore ethical dilemmas in AI.
    - **Open Forums:** Provide spaces for participants to ask questions and discuss complex concepts.
- **Feedback Mechanisms**
    - **Real-Time Feedback:** Use live polling and Q&A sessions to gather immediate feedback.
    - **Personalized Feedback:** Offer one-on-one sessions to address individual participant needs and concerns.

7.3.2 Collaborative Brainstorming

- **Design Thinking Workshops**
    - **Creative Idea Generation:** Facilitate workshops that encourage participants to brainstorm innovative applications of generative AI in real estate.
    - **Group Projects:** Assign small groups to work on specific problems, promoting teamwork and interdisciplinary thinking.
- **Collaborative Technologies**
    - **Online Platforms:** Utilize tools like Miro or Microsoft Teams for real-time collaboration and idea sharing.

    - **Brainstorming Sessions:** Structured sessions to compile and refine ideas generated by participants.

7.4 Training Evaluation

7.4.1 Quizzes and Questionnaires

- **Knowledge Assessments**
    - **Multiple-Choice Questions (MCQs):** Assess factual knowledge and understanding of key concepts.
    - **Case Study Evaluations:** Evaluate participants' ability to apply analytical skills to real-world scenarios.
- **Progress Tracking**
    - **Regular Quizzes:** Conduct quizzes at the end of each period to monitor comprehension and retention.
    - **Final Assessment:** A comprehensive evaluation at the end of the training to measure overall learning outcomes.

7.4.2 Collecting Feedback

- **Participant Surveys**
    - **Satisfaction Surveys:** Gather feedback on training content, delivery, and overall experience.
    - **Open-Ended Questions:** Allow participants to provide detailed suggestions and comments for improvement.
- **Post-Training Follow-Up**
    - **Feedback Analysis:** Analyze survey results to identify strengths and areas for enhancement.
    - **Continuous Improvement:** Use feedback to refine and adapt future training sessions, ensuring they meet the evolving needs of researchers.

Technologies and Practical Tools

- **Introduction to Specific Tools**
    - **Live Demonstrations:** Interactive presentations of tools like TensorFlow, PyTorch, and data visualization platforms.
    - **Multimedia Supports:** Provide video tutorials and interactive guides to facilitate self-paced learning.

Ethical and Regulatory Aspects

- **Awareness of Ethical Issues**
    - **Practical Cases:** Analyze real situations where ethical considerations influenced AI projects in real estate.
    - **Open Discussions:** Create spaces for participants to share their perspectives and concerns regarding AI ethics.

Personalization of the Training Experience

- **Adaptation to Individual Needs**
    - **Pre-Assessments:** Identify participants' prior knowledge to customize the learning path.
    - **Optional Modules:** Offer additional sessions on specific topics of interest to cater to diverse participant needs.

Emphasizing the Project Manager Role

- **Effective Coordination**
    - **Session Management:** Ensure smooth transitions between sessions and seamless integration of different training components.
    - **Clear Communication:** Establish robust communication channels to address participant queries and concerns promptly.
- **Risk Management**
    - **Anticipating Challenges:** Proactively identify potential obstacles in the training process.
    - **Adaptation Strategies:** Develop contingency plans to address unexpected issues and ensure the training's success.

Innovation and Engaging Activities

- **Integration of Immersive Technologies**
    - **Virtual and Augmented Reality:** Provide immersive experiences to visualize complex models or datasets.
    - **Interactive Simulations:** Use platforms that allow participants to simulate the impact of decisions based on predictive models.

Expected Outcomes and Performance Measurement

- **Acquired Skills**

    - **In-Depth Understanding:** Participants will master key concepts and be able to apply learned methods effectively.
    - **Enhanced Collaboration:** Improved ability to work within multidisciplinary teams, leveraging diverse expertise.
- **Key Performance Indicators (KPI)**
    - **Knowledge Assessment:** Scores from quizzes and practical evaluations.
    - **Active Participation:** Levels of engagement in discussions and collaborative activities.
    - **Positive Feedback:** High satisfaction rates measured through post-training surveys.

Post-Training Follow-Up and Continuous Development

- **Community of Practice**
    - **Online Platform:** Create a dedicated space for ongoing exchanges, resource sharing, and collaboration on new projects.
    - **Follow-Up Sessions:** Conduct regular webinars to delve deeper into specific topics and share progress updates.

Documentation and Additional Resources

- **Accessible Materials**
    - **Clear Language:** Provide documentation written in simple language, supplemented with glossaries for technical terms.
    - **Multimedia Integration:** Include podcasts, recorded webinars, and interactive resources to support diverse learning preferences.

This training session plan is meticulously designed to be rigorous, inclusive, and action-oriented, addressing the diverse needs of researchers from various backgrounds and skill levels. By incorporating an intermediate level, the program ensures that content is tailored to participants' prior knowledge, facilitating a smooth and effective progression for all attendees. Innovative strategies such as simplifying technical language, utilizing visual and multimedia supports, and engaging interactive and immersive activities are integrated to guarantee a deep understanding and practical application of key concepts.

Personalized learning paths and precise evaluation methods ensure that each participant can advance at their own pace and receive relevant feedback. Emphasizing interdisciplinary collaboration and ethical considerations prepares researchers to tackle the complex challenges associated with using AI in the real estate sector. Additionally, post-training follow-up and continuous development initiatives ensure ongoing improvement and the maintenance of acquired skills, thereby reinforcing the lasting impact of the training on the project.

This comprehensive and well-structured training plan equips researchers with essential skills in data analysis, predictive modeling, and the use of generative AI, fostering an environment of innovation, collaboration, and excellence in the field of AI applied to real estate.

8. Conclusion

8.1 Summary of Work Completed

In this project, I applied advanced data science and artificial intelligence techniques to analyze the French real estate market, integrating transactional and macroeconomic data to predict property prices and demand. This involved the collection and cleaning of extensive real estate transaction data and economic indicators such as GDP growth, unemployment rates, and interest rates. The data preparation process was rigorous, ensuring consistency across sources, handling missing values, and generating new features to enrich the dataset.

I developed predictive models capable of estimating real estate prices based on multiple factors. Models such as Linear Regression, Decision Trees, and XGBoost were evaluated, with XGBoost ultimately demonstrating the best performance due to its ability to capture complex interactions between variables. Additionally, I explored the potential of language models (LLMs) for extracting qualitative indicators, such as economic sentiment, from textual data sourced from articles and reports, adding a valuable contextual layer to the predictions.

Throughout the project, I encountered significant challenges, including managing the size and quality of diverse datasets, consolidating multiple data sources, and adhering to time constraints for such a complex undertaking. These obstacles highlighted the importance of efficient data processing techniques and careful feature engineering. Each stage of the project deepened my understanding of the impact of economic indicators on the real estate market and underscored the value of advanced models, like LLMs, in providing unique insights.

8.2 Future Perspectives and Work

For future developments, this project offers numerous avenues for improvement and expansion:

- **Incorporation of Recent and Localized Data**: To enhance prediction accuracy, future work could benefit from more recent and region-specific data. Including localized economic metrics, such as neighborhood income levels, population density, and infrastructure development, would allow for a more granular analysis that reflects real-time shifts in the market.
- **Advancement of LLM Integration**: The initial use of LLMs to extract sentiment and contextual economic information was promising. However, future iterations could employ more advanced or fine-tuned LLMs trained specifically on real estate-related content, improving

their capacity to capture sector-specific language and insights. Additionally, experimenting with multi-modal models that combine textual and quantitative data could provide even richer, multi-dimensional insights into market dynamics.

- **Expansion of Synthetic Data Generation**: The use of generative models, like StyleGAN2-ADA, to address data scarcity in underrepresented regions showed potential. Expanding this approach with more sophisticated generative models, or combining them with domain-specific data to enhance realism, could improve prediction robustness in areas with limited data. Establishing rigorous validation metrics, such as Wasserstein Distance and FID scores, would further ensure that synthetic data closely mirrors real-world trends.
- **Interdisciplinary Collaboration**: Collaborating with economists, urban planners, and social scientists could refine the project's methodology and provide insights into how economic trends impact real estate. This interdisciplinary approach would allow the model to incorporate additional dimensions, such as socio-economic indicators and urban development metrics, creating a more holistic predictive tool for real estate stakeholders.
- **Real-Time Predictive Capabilities**: Given the dynamic nature of the real estate market, incorporating real-time data feeds—such as live economic indicators, policy changes, or market news—could make the model highly responsive to sudden economic shifts. Developing a real-time predictive dashboard would be especially valuable for investors, policymakers, and real estate professionals, enabling data-driven decision-making in a fast-changing market environment.

In summary, this project lays the groundwork for a data-driven, AI-enhanced approach to real estate analysis, combining traditional predictive techniques with innovative AI methodologies. By continuing to refine data sources, advancing model sophistication, and integrating interdisciplinary insights, this project has the potential to evolve into a powerful tool for navigating the complexities of real estate markets in an ever-changing economic landscape.

9. Appendices

9.1 Bibliographic References