



**Master of Science**  
**<<Artificial Intelligence and Visual Computing>>**

**UNIVERSITY OF WEST ATTICA &  
UNIVERSITY OF LIMOGES**

**FACULTY OF ENGINEERING  
DEPARTMENT OF INFORMATICS AND COMPUTER ENGINEERING**

**Master Thesis**

**Real Estate Property Comparison in the Greek Market Using Advanced Image Similarity Methods  
and Web Scraping Techniques**

**Student: Asimina Tzana  
(aivc21015)**

**Supervisor: Anastasios L. Kesidis, Professor**

**Athens, September 2024**

**Μέλη Εξεταστικής Επιτροπής συμπεριλαμβανομένου του εισηγητή**  
**Η διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή**

A/A	ΟΝΟΜΑΤΕΠΩΝΥΜΟ	ΒΑΘΜΙΔΑ/ΙΔΙΟΤΗΤΑ	ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ
1	Αναστάσιος Κεσίδης	Καθηγητής	
2	Πάρις Μαστοροκώστας	Καθηγητής	
3	Παναγιώτα Τσελέντη	ΕΔΙΠ	

## ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Η κάτωθι υπογεγραμμένη Τζάνα Ασημίνα του Ιωάννη, με αριθμό μητρώου ainc21015 φοιτήτρια του Προγράμματος Μεταπτυχιακών Σπουδών Τεχνητή Νοημοσύνη και Οπτική Υπολογιστική του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών και του Τμήματος Μηχανικών Τοπογραφίας και Γεωπληροφορικής της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής σε συνεργασία με το Τμήμα Πληροφορικής της Σχολής Επιστημών και Τεχνολογίας του Πανεπιστημίου της Limoges της Γαλλίας, δηλώνω ότι: «Είμαι συγγραφέας αυτής της μεταπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Η Δηλούσα



## Contents

Contents .....	4
Figures .....	6
Abstract .....	7
Introduction .....	8
Problem Statement .....	9
Structure.....	10
Objective .....	12
Background .....	14
Scope .....	16
Research Questions.....	18
Literature Integration .....	20
Literature Review .....	23
Web Scraping .....	23
The Role of Web Scraping in Data-Driven Industries .....	24
Types of Web Data Extraction .....	25
Detailed Steps in Web Scraping .....	27
Detailed Overview of Technologies Used in Web Scraping.....	29
Data Management Tools for Web Scraping.....	32
Challenges in Web Scraping .....	37
Application of Web Scraping in Real Estate .....	38
Legal and Ethical Considerations:.....	40
Machine Learning in Real Estate .....	41
Application of Machine Learning in Real Estate.....	41
Theoretical and Mathematical Background.....	41
Predictive Modeling in Real Estate.....	44
Image Similarity Methods .....	47
Image similarity on real estate properties .....	47
Key Techniques and Their Mathematical Foundations .....	47
Django for Full-Stack Development .....	54
Origins and Philosophy.....	54
Usage in Full-Stack Development.....	55
Machine Learning Integration:.....	56
Core Features of Django ORM .....	56

Benefits in Real Estate Applications .....	57
Methodology .....	59
Overview of the Web Scraping Pipeline .....	59
Targeted Real Estate Platforms .....	60
Data Integration and Storage .....	60
Technical Architecture .....	61
Challenges and Solutions .....	62
Overview of the Web Scraping Pipeline .....	63
Design Principles and Objectives .....	63
Technical Workflow .....	63
Implementation Details .....	64
Explanation of Data Structure for Each Web Platform .....	65
Data Unification Approach .....	66
Database Management and Data Integration .....	67
Object-Relational Mapping (ORM) .....	68
Implementation of the Django Full Stack Application .....	69
Image similarity model training .....	70
Integration of Image Similarity Algorithm .....	74
Results .....	80
Dataset Description .....	80
Images dataset .....	86
Image similarity model results .....	87
Django app results .....	87
Performance Evaluation of the Image Similarity Detection Program .....	93
Conclusion and Future Work .....	105
References .....	107
Code .....	108

## Figures

Figure 1: The process of web scraping in business .....	24
Figure 2: The process of web scraping .....	28
Figure 3: SQL vs NoSQL databases. ....	32
Figure 4: Challenges in web scraping. ....	37
Figure 5:SIFT( Scale Invariant Feature Transform) .....	48
Figure 6: CNN for image classification. ....	51
Figure 7: Django architecture.....	54
Figure 8: Image similarity training process .....	70
Figure 9: Training steps .....	72
Figure 10: Django app .....	75
Figure 11: Properties details .....	75
Figure 12: Matching property ids found for specific property.....	76
Figure 13: Recommendation system.....	77
Figure 14: Similarity calculation .....	78
Figure 15: Number of Unique IDs per Source .....	80
Figure 16: Top 10 regions by number of listings .....	82
Figure 17: Top 10 regions based on the average price .....	83
Figure 18: Average price per square meter.....	85
Figure 19: UI of the image similarity app.....	93

## **Abstract**

The dynamic and complex nature of the real estate market, especially in regions like Greece with its diverse platforms and non-standardized content, poses significant challenges in data collection and analysis. This thesis presents a comprehensive system that integrates advanced web scraping techniques, machine learning models, and a full-stack Django-based application to significantly enhance the collection, processing, and analysis of real estate data. Central to this system is an innovative image similarity model, designed to improve the detection and comparison of real estate properties based on visual content, thereby enabling a more sophisticated analysis of market dynamics.

At the core of this system is the development of an image similarity model utilizing the ResNet50 architecture, optimized for visual recognition tasks within the real estate domain. The dataset, which includes images collected from Greek real estate platforms, is processed through a pre-trained ResNet50 model, fine-tuned to extract feature embeddings rather than perform direct classification. These images undergo preprocessing, including normalization and resizing to 224x224 pixels, to align with the input requirements of the ResNet50 model. The model then generates a 2048-dimensional feature vector for each image, effectively capturing its visual characteristics. These vectors are stored systematically for efficient retrieval and comparison in image similarity tasks.

The system is fortified with robust data management techniques, such as checkpointing and error handling, ensuring reliable processing of large-scale datasets. By leveraging the pre-trained ResNet50 model, the system achieves high accuracy in image similarity tasks while minimizing computational overhead, offering a scalable and efficient solution for real estate image analysis.

## Introduction

The advent of digital technology has markedly transformed the real estate landscape, particularly in Greece where platforms such as SpitoGatos, XE, Tospitimu, Plot, and Spiti24 have emerged as pivotal resources. These platforms aggregate a wealth of property data, fundamentally altering how properties are marketed, searched, and analyzed. The digital transition has not only democratized access to real estate information but has also streamlined the process of buying and selling properties, making it more transparent and accessible to a broader audience.

As beneficial as this digitization has been, it also presents notable challenges. Chief among these is the issue of duplicated listings across multiple platforms, which can clutter the search experience and obscure market insights. This redundancy complicates effective data management and necessitates sophisticated analytical strategies to ensure that consumers and real estate professionals can extract meaningful insights from the wealth of available data. Moreover, the digital aggregation of property listings raises questions about data accuracy, privacy, and the ethical use of such information.

In addressing these challenges, there is a clear need for innovative solutions that can enhance the efficiency of searching and analyzing real estate data while safeguarding the integrity and privacy of the information. As such, the role of technology in real estate is not only about facilitating access but also about enhancing the quality and reliability of the information provided, ensuring that it serves the best interests of all market participants.

To maximize the benefits of web scraping in the real estate sector, it is crucial to integrate scraped data with advanced analytical tools. Machine learning models can be applied to predict market trends and evaluate investment opportunities based on historical data. Additionally, image recognition technology can be used to enhance listing recommendations by analyzing property photos to identify similar features.

In conclusion, web scraping is a transformative tool in the digital real estate landscape, providing key advantages in data collection and market analysis. However, it must be used responsibly, with careful consideration of legal, ethical, and technical challenges. By effectively integrating web scraping with robust analytical tools, real estate professionals can unlock deeper insights into the market, driving smarter investment and business decisions in the Greek real estate sector.



## **Problem Statement**

A particularly pressing issue is the existence of listings that, while representing the same property, appear on multiple platforms with variations such as different agents or photographs. This phenomenon not only complicates the process of property comparison and market analysis but also impacts the accuracy of data-driven insights in the real estate sector. This thesis focuses on addressing this challenge through the development of a Django-based web platform that not only aggregates real estate listings from major Greek platforms but also employs an innovative image-based algorithm to detect and consolidate duplicate listings. By leveraging web scraping techniques to collect data and applying image recognition technologies, this project aims to streamline the property search process, enhance data quality, and provide deeper insights into the Greek real estate market.

Listings of the same property appearing across different web platforms, represented by various agents or with differing presentation elements (such as photos), create a fragmented view of the market. This fragmentation leads to difficulties in aggregating a coherent dataset that accurately reflects property availability, pricing, and characteristics. The challenge lies in the ability to effectively identify and consolidate these disparate listings into singular, comprehensive entries that accurately represent individual properties in the market analysis.

The development of such an algorithm is poised to significantly enhance the quality and reliability of real estate data, thereby improving market analysis, pricing strategies, and decision-making processes for stakeholders in the real estate sector. It addresses a critical gap in current data management practices, offering a pathway to more accurate and actionable market insights. Furthermore, this research contributes valuable knowledge to the fields of data science and real estate technology, pushing the boundaries of how data can be intelligently processed and utilized.

## Structure

This thesis is meticulously organized to present a coherent and thorough examination of the integration of web scraping, machine learning, and full-stack development technologies in analyzing the Greek real estate market. The structure of the thesis is simplified into five main chapters, each designed to build upon the previous to comprehensively address the research questions and objectives:

### Chapter 1: Introduction

This introductory chapter sets the stage for the research by outlining the thesis's scope, objectives, and significance. It defines the research questions that guide the study and explains the rationale behind the chosen methodologies and technologies, particularly focusing on how they apply to the real estate sector in Greece.

### Chapter 2: Literature Review

This chapter provides a critical review of the relevant academic and industry literature, serving as the foundation for the methodologies used in the thesis. It covers:

- **Core Technologies:** An overview of web scraping, machine learning, and Django full-stack development, highlighting key advancements and their relevance to the thesis.
- **Web Scraping:** Detailed exploration of web scraping techniques, including legal and ethical considerations crucial for compliance and effective data gathering.
- **Image Similarity Methods:** Discussion on various image processing algorithms and machine learning techniques, such as CNNs and feature matching, that are essential for analyzing real estate images.
- **Machine Learning in Real Estate:** Analysis of how machine learning is used in real estate for tasks like image recognition and predictive modeling, and its impact on enhancing decision-making processes.
- **Django for Full-Stack Development:** Examination of Django's architecture and its role in developing scalable, data-driven applications within the real estate domain.

### Chapter 3: Methodology and Implementation

Merging methodology with practical implementation, this chapter describes both the theoretical frameworks and their application:

- **Data Collection Techniques:** Strategies and tools developed for scraping real estate data from various platforms.
- **Image Processing and Machine Learning:** Methods used for preprocessing images and developing machine learning models to identify similarities and differences in property images.
- **Technological Stack:** Insight into the use of Django and other tools in crafting the application, focusing on how these technologies were integrated to support data processing and user interaction.
- **Scraping Tools and Database Management:** Implementation details of the scraping tools, database schema design and data normalization.
- **Machine Learning Pipeline and Django Application Architecture:** This section of the chapter has detailed the methodologies and implementation strategies for the machine learning pipeline, the Django application architecture, and the Image Similarity UI. By combining advanced machine learning techniques with a user-friendly interface, the application provides a robust tool for identifying and comparing real estate images based on visual similarity.

#### **Chapter 4: Results and Discussion**

This chapter presents the experimental results of the image similarity algorithm, alongside insights derived from the statistical analysis of the collected data and the evaluation of the Django application. The chapter is structured into four main sections: data analysis, model evaluation, application usability, and a broader discussion of the implications for the Greek real estate market.

#### **Chapter 5: Conclusion and Future Work**

This final chapter provides a summary of the research findings, highlights the key contributions, and discusses the challenges encountered during the study. It also offers suggestions for future research and development.

## Objective

This thesis aims to tackle the fragmentation of real estate listings by developing an innovative algorithm capable of identifying and consolidating listings across multiple platforms that, despite differences in representation, pertain to the same property. The focus is on leveraging sophisticated web scraping techniques to gather data from a variety of online sources and employing advanced matching algorithms that can recognize listings of the same property across different platforms and agents.

This thesis aims to bridge the gap in real estate data aggregation by designing and implementing a sophisticated web scraping framework that extracts real estate listings from Greece's major online platforms. The cornerstone of this project is the development of a novel image-based algorithm capable of identifying and consolidating listings that, despite variations in agent representation or presentation elements, correspond to the same property. This dual approach, combining advanced web scraping techniques with a state-of-the-art matching algorithm, seeks to create a unified, de-duplicated dataset of real estate listings, which will serve as a more accurate and reliable foundation for market analysis and decision-making.

This thesis is dedicated to addressing the issue of fragmented real estate listings by crafting a comprehensive solution that integrates cutting-edge web scraping methodologies with a robust image-based matching algorithm. The objective is twofold: first, to develop a sophisticated web scraping framework capable of systematically harvesting real estate listings from prominent Greek online platforms, and second, to design and implement an innovative algorithm that identifies and merges listings of the same property across these platforms, regardless of variations in agent representation or presentation elements such as photos.

To achieve this, the research will leverage the power of Python for web scraping, utilizing the libraries BeautifulSoup and Selenium and re for efficient data extraction. These tools are chosen for their ability to handle the complexities of modern web structures, enabling the automated collection of vast amounts of data with precision and resilience against common anti-scraping defenses.

Concurrently, this project will utilize Django, a high-level Python web framework, to develop a dynamic web platform. This platform will not only serve as the repository for the aggregated real estate listings but also as the interface through which users can interact with the consolidated data. Django's robustness and scalability make it the ideal choice for managing the backend of the project, facilitating data storage, processing, and presentation through a user-friendly web application.

The innovative algorithm at the heart of this thesis will utilize image recognition technologies to compare and identify listings with matching property images, a critical step in consolidating duplicate entries. By combining image analysis with textual data comparison, the algorithm aims to establish a high degree of accuracy in detecting listings that, despite superficial differences, represent the same property.

This integrated approach, combining sophisticated web scraping techniques with advanced image-based matching and the development of a Django-based web platform, aims to bridge the gap in real estate data aggregation. The ultimate goal is to create a unified, de-duplicated dataset of real estate listings that provides a more accurate and reliable foundation for market analysis and enhances decision-making processes for stakeholders in the real estate sector. Through this research, we aim to push the boundaries of data science and real estate technology, offering new methodologies for intelligently processing and utilizing real estate data.

The primary objective of this thesis is to develop and evaluate a comprehensive system that utilizes advanced web scraping techniques and machine learning models to extract, process, and analyze real estate listings from various Greek real estate platforms. This system aims to leverage the power of image similarity methods to enhance the detection and comparison of real estate properties, thereby facilitating a more sophisticated and accurate analysis of the real estate market in Greece.

## Background

The real estate market is not only a crucial component of the global economy but also a primary indicator of a nation's economic health and stability. This is especially true in Greece, where real estate plays a pivotal role due to its direct ties to both domestic economic activities and the extensive tourist industry. This section outlines the historical context of real estate market analysis, the transition to digital methods, and the specific challenges and opportunities within the Greek market.

## Importance of Real Estate in Greece

In Greece, the real estate sector contributes significantly to the national economy. It not only offers substantial investment opportunities but also serves as a key driver of related industries such as construction, hospitality, and retail. The unique appeal of Greece's landscape—ranging from urban centers like Athens to popular island destinations such as Santorini and Crete—further amplifies the complexity and attractiveness of its real estate market. Additionally, Greece's real estate market has been a focal point for both local and international investors, particularly after the financial crisis of the late 2000s when property values fluctuated markedly.

## Evolution from Traditional to Digital Analysis

Traditionally, real estate market analysis in Greece, as in many parts of the world, has relied heavily on manual data collection methods, including physical surveys and paper-based tracking. These methods are not only labor-intensive but also prone to errors and biases, which can skew market understanding and decision-making. The digital era has ushered in a new wave of data collection and analysis methodologies, characterized by the use of online platforms for real estate listings and transactions. This digital shift offers vast amounts of data that can be harvested and analyzed much more efficiently and accurately than ever before.

## Challenges in the Greek Real Estate Market

Despite the availability of digital tools, the Greek real estate market poses unique challenges:

- **Data Fragmentation:** Unlike markets in some other countries, Greek real estate data is often scattered across various platforms and not standardized, which complicates aggregation and analysis.
- **Economic Volatility:** The economic instability in Greece over the past decade has led to volatile real estate prices and investment patterns, requiring more dynamic and responsive analysis tools.
- **Regulatory Environment:** Greece's regulatory framework for real estate is complex and often changes, impacting the collection and utilization of real estate data.

## Opportunities for Technological Integration

The transition to digital data analysis opens up several opportunities:

- **Advanced Data Analytics:** Leveraging big data technologies and machine learning can transform raw data into insightful analytics, providing a deeper understanding of market trends and buyer behavior.
- **Image Processing Technologies:** Utilizing image recognition and processing technologies to analyze property photos and videos offers a new dimension of property evaluation, allowing for more nuanced comparisons and assessments.
- **Automated and Real-Time Analysis:** Digital tools enable real-time data analysis, which is crucial for adapting to fast-changing market conditions, a common scenario in Greek real estate due to its ties to the fluctuating tourism sector.

Understanding the background and the dynamics of the Greek real estate market is essential for developing effective digital tools for data collection and analysis. This thesis aims to address these needs by integrating advanced web scraping techniques and machine learning models to improve the accuracy, efficiency, and depth of real estate market analysis in Greece, thus providing stakeholders with more reliable and actionable insights.

### **Evolution of Data Analysis in Real Estate**

With advancements in information technology, there has been a substantial shift toward digital data collection and analysis. Real estate platforms now compile vast amounts of data online, including listings with detailed descriptions, photographs, and transaction histories. This shift presents an opportunity to leverage big data analytics to gain a deeper understanding of market dynamics, pricing trends, and consumer preferences.

### **Importance of Web Scraping and Big Data**

Web scraping has emerged as a critical tool in this context, enabling the extraction of large datasets from multiple real estate websites. These data are essential for building comprehensive market models that can predict trends and guide investment decisions. However, the nature of data on real estate platforms poses unique challenges. Listings are often not standardized; they vary in format, detail, and accuracy, requiring sophisticated techniques for data cleansing and preparation.

### **Role of Machine Learning and Image Processing**

Moreover, the visual component of real estate listings — images — has largely been underutilized in quantitative analyses despite containing valuable information about property features and quality. Recent developments in machine learning, particularly in image recognition and processing, have opened new avenues for incorporating visual data into real estate analysis. By analyzing images, it is possible to identify patterns and features that are not mentioned in textual descriptions, such as the property's condition, style, and more subtle attributes that could influence its value.

### **Integration into Full-Stack Applications**

The integration of these technologies into full-stack applications, particularly using frameworks like Django, further enhances the utility and accessibility of real estate data. Django's capabilities enable the development of robust, scalable applications that can handle large volumes of data while providing a user-friendly interface. This allows users, whether they are market analysts, real estate agents, or prospective buyers, to interact directly with the data and gain insights that were previously difficult to access.

### **Contribution to the Field**

This thesis seeks to build on these advancements by creating a comprehensive system that not only collects and processes real estate data but also provides analytical tools that harness the power of both textual and visual data. By doing so, it aims to enhance the accuracy, depth, and speed of real estate market analysis, providing stakeholders with better tools to make informed decisions. This approach is particularly novel in the context of the Greek real estate market, where such technological integration has not yet been fully explored or implemented.

## Scope

This project is strategically designed to address two primary challenges in the analysis of the Greek real estate market through the use of advanced data science techniques, specifically web scraping and machine learning.

### Development of a Web Scraping System

The initial aspect of this project focuses on the development of a sophisticated web scraping system tailored to navigate and extract data from a variety of Greek real estate platforms. These platforms exhibit significant variation in their structure and content presentation, which necessitates a flexible scraping approach capable of handling different data formats and layouts. The information to be collected includes:

- **Textual Data:** This includes key details such as prices, locations, sizes, types of properties, and additional features that are critical for comprehensive market analysis.
- **Visual Data:** Images of listings are crucial as they provide insights into the aesthetic and functional aspects of the properties, which are often decisive factors in real estate valuation and customer interest.

This scraping system will employ adaptive algorithms to ensure efficient data extraction while maintaining compliance with ethical web scraping guidelines and avoiding disruptions to the functionality of the target websites.

### Application of Machine Learning Techniques

The second core component of this project involves the application of machine learning algorithms to analyze the collected images to identify similarities and discrepancies. This analysis aims to:

- **Detect Duplicate Listings:** Identifying duplicate or near-duplicate listings across different platforms is vital for ensuring the accuracy of market analysis by eliminating redundancies.
- **Recognize Closely Related Properties:** By analyzing image similarities, the system can cluster properties that, while not identical, share significant features, thus providing potential buyers with alternative options and aiding in price comparison.

### Geographical Focus on Greece

Focusing specifically on the Greek real estate market introduces unique challenges and opportunities:

**Diverse Property Types:** Greece's real estate landscape is characterized by a wide array of property types, including urban apartments in Athens, traditional houses in village settings, and luxurious villas in coastal regions. This diversity requires a versatile analysis approach.

**Regional Market Variations:** The real estate market dynamics can vary significantly between different regions in Greece, influenced by factors such as tourism, local economic conditions, and demographic trends. This geographic variability enriches the dataset and provides a fertile ground for demonstrating the effectiveness of machine learning in adapting to and elucidating complex market dynamics.

### Contribution to Real Estate Analytics



The comprehensive approach outlined in this project not only aims to enhance the understanding and analysis of the Greek real estate market but also seeks to contribute to the broader field of real estate analytics by demonstrating how integrating technological solutions can address specific market challenges. The methodologies developed could serve as a blueprint for similar analyses in other geographic contexts, potentially driving further innovations in the field of real estate data science.

## Research Questions

The real estate market is burgeoning with data, yet the challenge of efficiently harnessing this information remains significant, particularly in markets like Greece where platform heterogeneity and data inconsistency prevail. This thesis aims to bridge this gap through the development of a comprehensive analytical system that leverages advanced web scraping techniques, machine learning models, and a robust full-stack application. The research is driven by several critical questions that explore the potential of technology to transform real estate data gathering, processing, and analysis. These questions probe the effectiveness of web scraping for collecting diverse data, the ability of machine learning to detect image similarities and duplicate listings, and the integration capabilities of Django to deliver a user-centric platform for real estate professionals and decision-makers. The answers to these questions will provide a foundation for understanding how technological integration can enhance market analysis and decision-making processes in the real estate sector.

### The research will focus on several key questions:

- How can web scraping be effectively utilized to gather comprehensive real estate data from Greek platforms, which often feature diverse and non-standardized content?
- What methodologies can be developed to identify listings across multiple platforms that represent the same property but are presented by different agents or with different photos?
- What machine learning techniques are most effective for identifying similarities in real estate images to detect duplicates or closely related properties across different platforms?
- Can a Django-based full-stack application integrate these technologies to provide a robust, user-friendly platform for real estate data visualization and decision-making?
- What are the implications of consolidating fragmented listings for market analysis and decision-making processes in the real estate sector?

## Addressing the Research Questions: Objectives and Methodology

### Data Collection

**Advanced Scraping Techniques:** Implement cutting-edge web scraping technologies to comprehensively extract data from multiple Greek real estate platforms, ensuring a rich dataset that reflects the current market landscape.

**Structured Data Extraction:** Focus on capturing essential structured data such as price, location, size, and property type, which are critical for basic filtering and sorting functionalities.

**Unstructured Data Extraction:** Gather rich unstructured data including images and descriptive text to enable deeper, more nuanced analysis and to facilitate advanced features like image similarity analysis.

### Data Processing and Management

**Data Cleaning Protocols:** Establish and maintain high standards of data quality through rigorous cleaning protocols that address missing values, errors, and inconsistencies.

**Data Storage Solutions:** Implement robust database management systems that ensure data integrity, security, and fast, reliable access, which is vital for real-time applications.

### **Image Processing and Machine Learning**

**Preprocessing Techniques:** Utilize sophisticated image preprocessing methods to standardize and enhance image data before analysis, improving the reliability of machine learning outcomes.

**Model Development:** Develop and train advanced machine learning models to detect similarities and differences in property images, which will help in identifying duplicate listings and extracting meaningful features from images.

**Image Similarity Methods:** Integrate state-of-the-art image similarity algorithms to significantly boost the system's capability to analyze and categorize image data effectively.

### **Full-Stack Application Development**

**Django-Based Integration:** Construct a Django-based web application that seamlessly integrates backend processing with a user-friendly frontend, ensuring that users can easily navigate and interact with the system.

**Real-Time Data Interaction and Dynamic Visualization:** Provide dynamic tools that allow users to interact with real-time data and visualize results in meaningful ways, enhancing user engagement and decision-making processes.

### **Integration of Image Similarity Features**

**Interactive Image Uploads and Comparisons:** Enable functionality for users to upload property images and use built-in tools to compare them against the database, aiding in identifying similar properties or duplicates.

**Recommendations:** Develop sophisticated recommendation algorithms that leverage analyzed image data to suggest properties based on user preferences and past interactions.

### **Analytics and Insights**

**Market Trends Analysis:** Conclude the study with an extensive analysis of the collected data, focusing on market trends, price dynamics, and property characteristics across Greece. This analysis aims to unearth actionable insights and highlight investment opportunities.

By systematically addressing these research questions and objectives, this thesis will contribute valuable knowledge and tools to the field of real estate market analysis, offering practical solutions that can adapt to other markets and scales.

## **Literature Integration**

Recent academic studies and industry reports have increasingly utilized web scraping techniques and machine learning algorithms to analyze various aspects of the real estate market. This literature overview examines key works that highlight the effectiveness and adaptability of web scraping in exploring market trends, pricing strategies, and consumer behavior analytics in diverse data environments:

### **1. Exploring the Rental Market Dynamics of the Guadalajara Metropolitan Area**

This study focuses on creating a custom dataset using web scraping techniques to understand factors influencing rental pricing and property listings in the Guadalajara metropolitan area. By collecting data from real estate websites, the researchers were able to analyze market dynamics and rental trends. The study employed a Python web scraping script to navigate through various real estate websites, extracting data on rental prices, property locations, and other relevant attributes. This detailed analysis helps provide insights into the factors affecting rental markets in the region.

### **2. Media Framing in the Digital Age: Interplay of Real Estate and Welfare Narratives in South Korean News Articles**

This research leverages web scraping to collect and analyze online journalistic articles, studying the interplay between real estate and welfare narratives in South Korean media. The researchers utilized web scraping to gather a large dataset of news articles from various online sources, which were then analyzed to understand how real estate issues are framed in the context of welfare policies. This approach highlights the utility of web scraping in media studies and the analysis of digital content.

### **3. Forecasting Housing Price Using GRU, LSTM, and Bi-LSTM for California**

Utilizing web scraping to gather data from major real estate sales platforms, this study applies advanced machine learning models such as GRU (Gated Recurrent Unit), LSTM (Long Short-Term Memory), and Bi-LSTM (Bi-directional LSTM) to forecast housing prices in California. The data collected included historical sales prices, property details, and temporal factors. The application of these sophisticated models demonstrates the capability of web scraping to support data-driven forecasting methods, providing accurate predictions for housing market trends.

### **4. Modelo Random Forest Aplicado a Precificação de Imóveis à Venda em Aracaju, SE**

In this work, web scraping is used to collect data on property prices in Aracaju, Brazil. The data gathered includes various attributes such as location, size, and price of properties. The collected data is then analyzed using the Random Forest model to develop pricing strategies. This study showcases the application of web scraping in developing robust real estate pricing models, highlighting its effectiveness in extracting valuable insights from large datasets.

### **5. Enhancing Real Estate Market Insights through Machine Learning: Predicting Property Prices with Advanced Data Analytics**

This research utilizes web scraping to gather data for predicting property prices in Mumbai. Advanced data analytics and machine learning techniques, including various regression and classification models, are employed to analyze the collected data. The study demonstrates how web scraping can be used to compile comprehensive datasets, which are then used to enhance market insights and predict property prices accurately.

## **6. Applying Machine Learning Models for Forecasting House Prices – A Case of the Metropolitan City of Karachi**

This study collects web-based real estate data to develop a dataset for forecasting house prices using machine learning models. The data collection process involves web scraping from various real estate websites to gather information on property listings, prices, and other relevant features. The study then applies different machine learning models to forecast house prices, showcasing the potential of web scraping in generating actionable real estate market insights.

## **7. Scrimmo: A Real-Time Web Scraper Monitoring the Belgian Real Estate Market**

The Scrimmo project presents a comprehensive exploration of web scraping for automated data collection and analysis in the Belgian real estate market. The real-time web scraper, named SCRIMMO, is tailored to collect data from websites containing real estate listings. This study underscores the real-time monitoring capabilities enabled by web scraping, providing continuous updates and insights into the real estate market.

## **8. ML-based Telegram bot for real estate price prediction**

The paper presents an innovative approach to real estate price prediction by developing a Telegram bot that uses machine learning algorithms. This bot is designed to provide users with real-time predictions of real estate prices, leveraging the accessibility and user-friendliness of Telegram.

## **9. Predictive analytics using Big Data for the real estate market during the COVID-19 pandemic**

This study investigates which apartment attributes most significantly influence price changes during the pandemic. Using web scraping, 18,992 property listings were collected from Vilnius. Fifteen machine learning models were tested to forecast price revisions, with SHAP values used for interpretability. The study found that the real estate market was resilient, with less dramatic price drops than anticipated. The time-on-the-market (TOM) variable was the most dominant predictor of price revisions, showing an inverse U-shaped behavior. **Link:** [here](#)

## **10. A Model for the Estimation of Land Prices in Colombo District using Web Scraped Data**

The real estate market in Sri Lanka, especially in Colombo, has seen a boom, with increasing prices and demand. Accurate land price estimation is crucial for investors and the general public. This study addresses the lack of publicly available structured data on land prices by scraping data from online advertisements and combining it with other publicly available data to build a comprehensive dataset for machine learning model development.

## **11. Web scraping or web crawling: State of art, techniques, approaches and application**

This paper discusses the use of web scraping for collecting data and the application of machine learning for sentiment recognition and market analysis. The paper explores the concept of web scraping, its significance in the modern age of Business Intelligence, and its applications in various fields such as data science, artificial intelligence, and cybersecurity. It highlights the technologies and techniques involved in web scraping, including spidering and pattern matching, and discusses the ethical and legal issues associated with it.

## **12. Web Scraping Methods Used in Predicting Real Estate Prices**

This paper discusses various web scraping methods employed to collect data from the real estate market and their application in predicting real estate prices. The study emphasizes the importance of accurate and up-to-date data in making reliable predictions and explores different approaches to web scraping, data processing, and machine learning techniques.

These studies collectively demonstrate the broad applicability and utility of web scraping techniques in real estate analysis. By enabling the collection and analysis of large volumes of data from various online sources, web scraping facilitates a deeper understanding of market trends, pricing strategies, and consumer behavior. This adaptability makes web scraping an invaluable tool in modern real estate research and industry practice.

## Literature Review

### Web Scraping

Web scraping is an indispensable digital technique that automates the extraction of data from websites. It transforms the vast and chaotic world of web data into structured, actionable information, making it a cornerstone technology for many data-driven industries. From real estate to retail, finance to healthcare, web scraping enables businesses and researchers to capture and utilize web data in real-time, thus driving decision-making, competitive analysis, and market research.

### Significance of Web Scraping in Modern Industries

In the current digital age, where data is as valuable as currency, the ability to access and analyze data quickly and accurately is crucial. Web scraping provides a strategic advantage by automating the collection of this data, thus bypassing traditional methods of data gathering such as manual entry or single-source data feeds which can be slow and error-prone.

Web scraping tools simulate the behavior of a web browser to retrieve the content of web pages, which includes text, images, and other multimedia elements. However, unlike human interaction with a browser, web scraping automates and repeats these interactions programmatically, allowing for the collection of large amounts of data in a fraction of the time.

### How Web Scraping Fuels Industries

- **Real Estate:** In real estate, web scraping is used to gather data on property listings, historical prices, neighborhood demographics, and market trends. This information is crucial for investors, real estate companies, and potential home buyers to make informed decisions.
- **Finance:** Financial analysts use web scraping to track stock market movements, corporate financials, and economic indicators in real-time, providing a solid basis for investment decisions.
- **Retail:** Retail companies leverage web scraping to monitor competitor pricing, product availability, consumer reviews, and market trends, helping them to optimize pricing, manage inventory, and enhance customer experience.
- **Academic Research:** Researchers and academics utilize web scraping for gathering vast datasets from multiple sources online, enabling comprehensive studies across various fields such as social sciences, technology, and humanities.

Web scraping is a vital digital technique used to extract data from websites. This process allows for the automated collection of information from the Internet, which is especially beneficial in fields that require frequent updates and access to a vast amount of data. In the real estate sector, web scraping provides a crucial edge by collecting extensive data on property listings, market prices, and trends efficiently and rapidly.

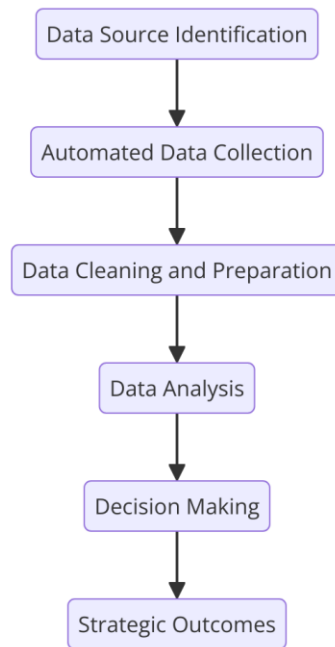


Figure 1: The process of web scraping in business

### The Role of Web Scraping in Data-Driven Industries

In today's data-driven world, timely and accurate information is paramount. Web scraping serves as a bridge between raw data available online and structured data ready for analysis. By automating the data collection process, web scraping not only saves time and resources but also enhances the reliability of the data by reducing human errors.

Web scraping is a powerful technology employed to collect data automatically from websites. This data is often crucial for competitive analysis, market research, and real-time decision-making. Given its importance across various industries, including real estate, finance, retail, and more, understanding the intricacies of web scraping is vital. Web scraping, fundamentally, is about extracting data from websites. It's a practice that bridges the gap between static web content and dynamic data analysis applications. Here's a deeper theoretical exploration of the concept.

### Applications Across Industries

The versatility of web scraping extends across various industries, each leveraging it for distinct purposes:

- **Real Estate:** Aggregating property listings from multiple real estate websites for price comparison, trend analysis, and market research.
- **Finance:** Collecting financial data from stock market sites, financial news portals, and company websites for investment analysis and decision-making.
- **Retail:** Monitoring competitor prices, tracking consumer reviews, and analyzing market trends to optimize pricing strategies and inventory management.



- **Academic Research:** Gathering data sets from various online sources to support empirical research, sentiment analysis, and other scholarly activities.

## Definition and Core Concepts

**Web Scraping:** At its core, web scraping is the process of using automated tools to extract information from web pages. This information is typically presented in HTML format, which web scraping tools analyze to extract data such as text, links, and other content.

**HTML and the DOM:** Understanding web scraping requires a basic understanding of HTML (HyperText Markup Language) and the DOM (Document Object Model). HTML structures the content on web pages, while the DOM is an object-oriented representation of the web page, which can be modified with programming languages like JavaScript.

## Types of Web Data Extraction

**Static Content Extraction:** This involves scraping data from web pages that do not require user interaction to display content. Tools like **Requests** and **BeautifulSoup** are typically used here to parse the static HTML of the page to retrieve the content.

**Dynamic Content Extraction:** For web pages that load content dynamically with JavaScript, tools like Selenium or Puppeteer are used. These tools can interact with web pages just like a user would, allowing them to retrieve data that only loads as a result of user interactions or that appears after initial page loads.

## Benefits of Web Scraping

Web scraping offers a multitude of advantages that make it an indispensable tool for businesses and researchers alike. Here are some key benefits:

- **Efficiency:** Web scraping automates the data collection process, significantly reducing the time and resources required compared to manual methods. This allows for rapid acquisition of large datasets, which can be crucial for timely decision-making and analysis.
- **Accuracy:** By minimizing human intervention, web scraping reduces the likelihood of errors that can occur during manual data collection. Automated scripts can consistently extract data with high precision, ensuring the reliability of the collected information.
- **Scalability:** Web scraping can handle vast amounts of data across numerous web pages. It is capable of scaling up the data collection process to accommodate large volumes of data, making it feasible to gather information that would be impractical to collect manually.
- **Cost-Effective:** Automating data collection processes through web scraping can lead to substantial cost savings. It reduces the need for large teams of data collectors, thereby lowering labor costs and increasing operational efficiency.
- **Real-Time Data Access:** Web scraping can be set up to run at regular intervals, providing access to the most current data available. This is particularly valuable for industries that rely on up-to-date information, such as finance, e-commerce, and news.
- **Competitive Advantage:** By providing access to comprehensive and up-to-date data, web scraping can offer businesses a competitive edge. Companies can track competitor prices, monitor market trends, and gather insights that inform strategic decisions.

- **Customizability:** Web scraping scripts can be tailored to collect specific data points relevant to a particular analysis or business need. This flexibility ensures that the data gathered is highly relevant and aligned with specific objectives.
- **Broad Data Collection:** Web scraping can aggregate data from multiple sources, providing a holistic view of the subject matter. This capability is particularly useful for market research, where insights from various platforms are needed to form a complete picture.
- **Enhanced Data Analysis:** The structured data obtained through web scraping can be directly fed into analytical tools and models, facilitating advanced data analysis, machine learning, and predictive analytics. This enables deeper insights and more informed decision-making.
- **Unbiased Data Collection:** Automated data scraping ensures that data is collected in a consistent manner, free from the biases that can occur with manual collection. This helps maintain the objectivity and integrity of the data.
- **Availability of Historical Data:** Web scraping can also be used to archive data over time, building a repository of historical data that can be valuable for trend analysis and forecasting.

## Detailed Steps in Web Scraping

The web scraping process can be broken down into more detailed steps:

**Identification of Target URL:** The first step involves identifying the URL or series of URLs from which data needs to be extracted. This can often involve dynamic construction of URLs especially if the data spans multiple pages or categories.

**Sending Requests:** Using tools like Requests in Python, a web scraper sends a request to the server hosting the website. This is akin to what happens when you manually click a link or type a web address; the server then responds with the data, typically in HTML format.

**Data Parsing:** Once the HTML content of the webpage is retrieved, parsing libraries such as BeautifulSoup are used to parse the HTML. This step transforms the raw HTML content into a structured format that is easier to navigate and extract data from. This involves isolating the parts of the HTML document that contain the relevant data.

**Data Extraction:** After parsing the document, the next step involves extracting the actual data. This can include anything from product details on e-commerce sites to property listings on real estate platforms. The data must be meticulously extracted to ensure accuracy and completeness.

**Data Storage:** The extracted data is then formatted and stored in a database, file, or a spreadsheet. This structured data is now ready for analysis or integration into data-driven applications.

**Automation and Scheduling:** For ongoing data collection, web scraping processes are often automated and scheduled to run at specific intervals. This ensures that the collected data is up-to-date and relevant.

**Data Cleaning:** Often, the extracted data may contain inconsistencies, errors, or duplicates. Data cleaning processes are applied to ensure the quality of the data before it's used for any analysis or business application.

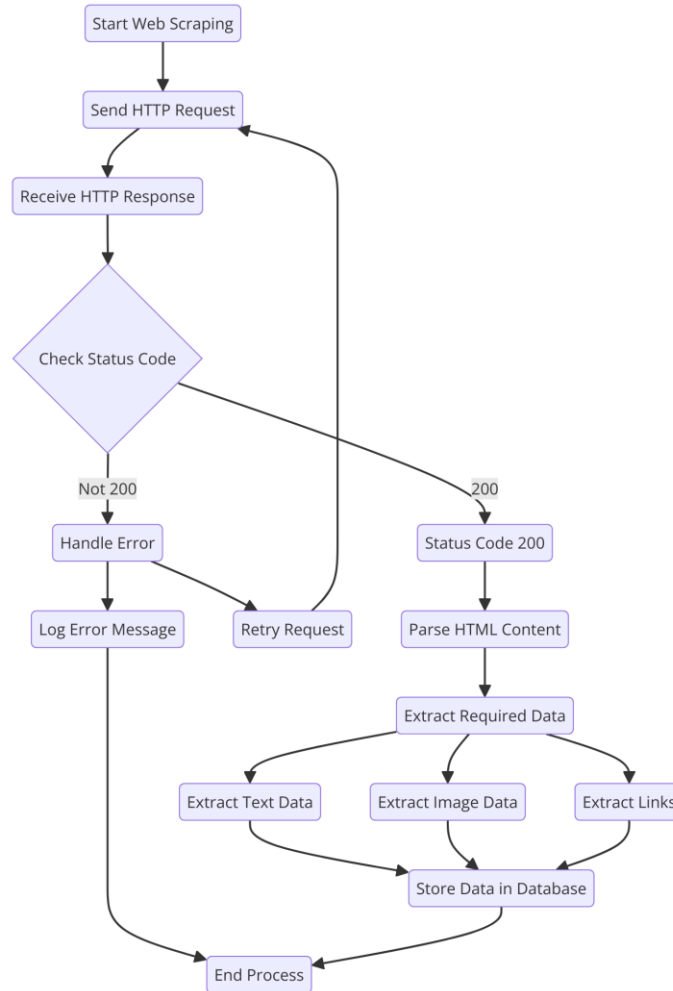


Figure 2: The process of web scraping

## Technologies and Tools

- **HTTP Libraries:** Tools like Requests for Python are used for sending HTTP requests to servers.
- **HTML/XML Parsers:** Libraries such as BeautifulSoup, lxml, and HTMLParser are used to parse and extract data from HTML/XML documents.
- **Browser Automation:** Tools like Selenium or Puppeteer are used for websites that require interaction or render content dynamically using JavaScript.
- **Data Management Tools:** Once data is scraped, it is often managed using databases like MySQL, PostgreSQL, or MongoDB, or even in simpler formats like CSV or JSON.

## Detailed Overview of Technologies Used in Web Scraping

In the realm of web scraping, a variety of technologies and tools are employed to effectively gather data from the web. This overview details the primary technologies used, highlighting their functions, use cases, and how they integrate into the data collection and management process.

### HTTP Libraries

HTTP libraries are fundamental components in the toolkit of web scraping, acting as the backbone for data exchange on the web. They handle the protocol level interactions necessary for sending and receiving HTTP requests and responses. In the context of web scraping, these libraries are utilized to programmatically request the HTML or XML data from web servers, mimicking the behavior that occurs when a user visits a website through a browser.

### Understanding HTTP Libraries

HTTP (Hypertext Transfer Protocol) is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that users can easily access. HTTP libraries manage the sending of request messages, including method requests (GET, POST, DELETE, PUT), and handle the responses from servers.

**GET Requests:** Used to request data from a specified resource. In web scraping, GET requests are commonly used to retrieve the HTML content of a webpage.

**POST Requests:** Used to send data to a server to create/update a resource. This is often used in web scraping when interacting with forms or log-in pages.

### Features of HTTP Libraries

**Handling Sessions:** Advanced HTTP libraries like Requests manage sessions, making it easier to persist parameters across requests. For instance, a session might involve logging into a website and maintaining that login state across multiple requests.

**Cookies:** HTTP libraries handle cookies automatically, allowing scripts to interact with websites that require cookie-based authentication.

**Redirection:** Automatically handles HTTP redirections, which is when a requested URL points to another URL. This feature is crucial for maintaining the flow of data retrieval without manual intervention.

**Timeouts:** Supports setting timeouts to ensure that requests do not hang indefinitely, which can improve the reliability and efficiency of web scraping scripts.

**Headers Customization:** Allows custom headers to be attached to requests, which can be used to simulate different types of browsers or custom API needs.

**SSL Verification:** Capable of handling secure connections, ensuring data is encrypted over the network, and optionally allowing for the verification of SSL certificates.

### Role of HTTP Libraries in Web Scraping

In web scraping, HTTP libraries are not just tools for sending and receiving data; they are the intermediaries that translate a scraper's needs into HTTP protocol commands, manage data transmission, and handle network

protocols. This includes constructing requests, managing persistent connections, and parsing responses from the web servers.

**Efficiency:** They optimize the process of sending requests and parsing responses, which can be crucial when dealing with high-volume data scraping.

**Flexibility:** HTTP libraries can be extended with middleware or plugins to handle custom scenarios encountered in web scraping, such as rate limiting or domain-specific data parsing rules.

**Requests for Python:** One of the most popular HTTP libraries, Requests<sup>1</sup> is renowned for its simplicity and ease of use. It allows for sending HTTP/1.1 requests extremely easily, without the need to manually add query strings to your URLs, or to form-encode your POST data.

The Requests library in Python, specifically, is lauded for its user-friendly interface that abstracts away most of the complexities involved with making HTTP requests. It makes HTTP methods directly accessible through simple functions, which greatly simplifies the process of coding web scraping scripts, especially for beginners.

## HTML/XML Parsers

HTML/XML parsers are indispensable tools in web scraping, designed to interpret and extract data from HTML and XML documents. These tools are crucial for turning unstructured web data into a structured format that can be manipulated and analyzed programmatically. Below is an overview of the most commonly used HTML/XML parsers in Python, which are essential for efficient data extraction in web scraping projects.

### Overview of Popular HTML/XML Parsers

**BeautifulSoup:** This Python library is designed for quick turnaround projects like screen-scraping. What makes BeautifulSoup a popular choice is its ability to parse anything you give it and build a parse tree from it. It automatically converts incoming documents to Unicode and outgoing documents to UTF-8. It's flexible and forgiving; it can handle different markup types and build a parse tree that makes intuitive sense.

**lxml:** Known for its efficiency and speed, lxml is a comprehensive library that handles both XML and HTML data very well. It provides the feature completeness of these libraries with the simplicity of a Pythonic API, which makes it a powerful tool for complex web scraping tasks that require quick execution and handling a large volume of data.

**HTMLParser:** This is a built-in Python library that offers a straightforward method of parsing HTML documents. It is especially suited for projects where simplicity and speed are necessary, and external dependencies need to be minimized. It provides basic functionalities that are enough for many scraping tasks, particularly when scraping simpler HTML pages.

### Functionality of HTML/XML Parsers

These parsers provide several functionalities that are essential for web scraping:

---

<sup>1</sup> <https://pypi.org/project/requests/>

**Tree Traversal:** After parsing a document, these libraries provide numerous ways to navigate, search, and modify the parse tree. This is particularly useful for extracting specific data from deep within a complex structure of HTML or XML.

**Data Extraction:** They allow for precise data extraction using tags, attributes, or text content, making it easier to pull out exact pieces of information from a web page.

**Error Tolerance:** HTML/XML parsers are typically built to be tolerant of malformed markup and are capable of making sense of the intended structure even when not strictly compliant with HTML/XML standards.

### Common Use Cases

**Web Data Extraction:** From pulling out data such as headlines, links, and text blocks to extracting and transforming data from online catalogs, parsers can handle a wide range of web scraping needs.

**Data Cleansing:** After initial data extraction, these tools can also be used to clean and organize raw web data, preparing it for analysis or storage.

**Legacy System Integration:** For older websites or systems that do not offer modern API access, HTML/XML parsers can be used to automate data extraction, enabling integration with newer systems.

### Role in Web Scraping

HTML/XML parsers are more than just tools for pulling data out of web documents—they transform the web scraping process by adding structure to the data being extracted and by offering methods to handle web data at scale. While BeautifulSoup offers ease of use and flexibility, lxml brings performance and extensive functionality, and HTMLParser offers a lightweight, dependency-free option. Depending on the specific needs of a web scraping project, such as the complexity of the documents or the speed required, the choice of parser can significantly affect both the performance and ease of scraping.

In summary, HTML/XML parsers are critical for any web scraping operation as they provide the necessary functions to extract and manipulate web data efficiently, making them a cornerstone of any data extraction or web automation project.

### Browser Automation

**Selenium:** Selenium automates browsers. Primarily it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can also be automated as well.

**Puppeteer:** Puppeteer is a Node library which provides a high-level API to control headless Chrome or Chromium over the DevTools Protocol. It is often used for automated testing of web applications, scraping web pages, and generating pre-rendered content from SPAs (Single Page Applications).

**Functionality:** These tools simulate user interactions with a web browser, allowing for the scraping of dynamic content generated by JavaScript scripts and Ajax calls.

**Common Use Cases:** Scraping sites that load their data with JavaScript, performing tasks that require login and navigation, and automating form submissions.

## Data Management Tools for Web Scraping

Once data is successfully scraped from the web, the next critical step involves managing that data efficiently. Effective data management ensures that the data is not only stored securely but is also readily accessible for analysis and integration into various applications. This chapter will explore the range of data management tools and formats that are pivotal in the handling of data post-scraping, their functionalities, common use cases, and how they integrate into the broader web scraping workflow.

### Categorization and Explanation of Database Systems Used in Web Scraping

Databases play a pivotal role in the storage, management, and retrieval of data collected through web scraping. Based on their structure and data handling capabilities, they are broadly categorized into two types: SQL Databases and NoSQL Databases. Each type serves different needs depending on the nature of the data and the application requirements.

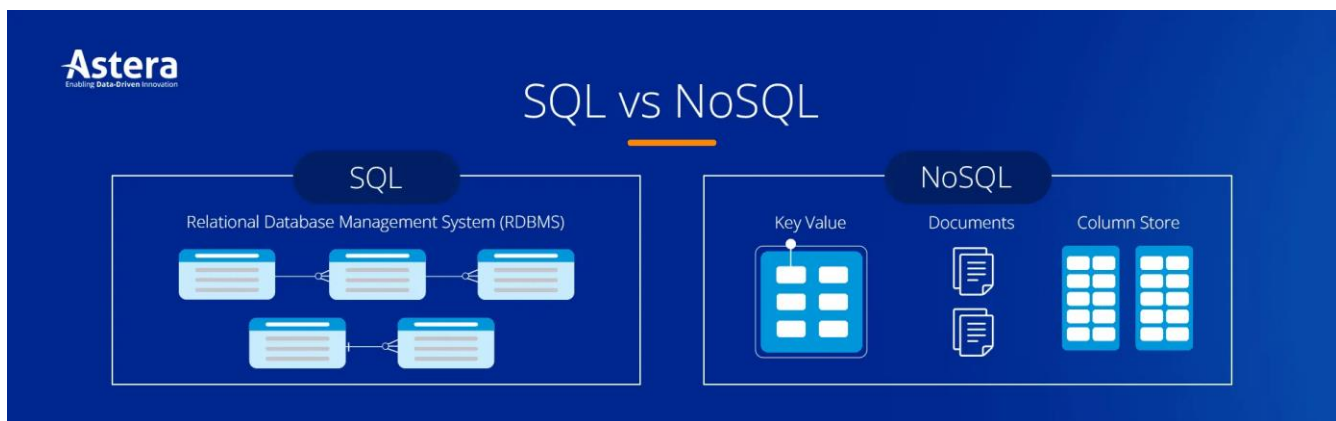


Figure 3: SQL vs NoSQL databases.

## SQL Databases

SQL databases, also known as relational databases, use a structured query language (SQL) for defining and manipulating data. This model is based on data stored in tables and the relationships among those tables. SQL databases are particularly effective for scenarios where integrity and data consistency are crucial.

### 1. MySQL

**Description:** MySQL is one of the most popular relational database management systems. Known for its high performance, reliability, and ease of use, MySQL is used in a wide range of applications, from small to large-scale web applications.

**Functionality:** Provides robust data security, supports complex queries, efficient data management, and has broad compatibility with major hosting providers.

**Use Cases:** Often used for web applications requiring a reliable database solution without the necessity for extensive scaling in terms of concurrent writes and reads.

### 2. PostgreSQL



**Description:** PostgreSQL is a powerful, open-source object-relational database system. It extends the SQL language with advanced features that enable the storage and scaling of complicated data workloads.

**Functionality:** Supports advanced data types and performance optimization features like indexing, full-text search, and concurrency without read locks. It is highly extensible, allowing users to define their own data types, build out custom functions, and even write code from different programming languages without recompiling the database.

**Use Cases:** Suitable for applications that require frequent read and write operations, complex queries, and high concurrency, such as dynamic web applications and enterprise-level systems.

## NoSQL Databases

NoSQL databases are designed to provide high operational speed and flexibility with regard to the data models. They are particularly useful for handling large volumes of structured, semi-structured, and unstructured data. NoSQL databases do not use a standard SQL query language and often provide more scalable performance.

### 1. MongoDB

**Description:** MongoDB is a document-oriented NoSQL database that stores data in JSON-like documents with dynamic schemas (in the BSON format), making the integration of data in certain types of applications easier and faster.

**Functionality:** Offers high flexibility with its schema-less architecture, making it suitable for storing unstructured and semi-structured data. It supports ad-hoc queries, indexing, and real-time aggregation, providing a rich set of features to handle diverse data types effectively.

**Use Cases:** Ideal for applications that require rapid development, frequent application iterations, and where data structures can change over time. Commonly used in big data and real-time web applications.

### Summary

Choosing between SQL and NoSQL databases typically depends on the specific requirements of the application and the nature of the data being handled:

- SQL Databases are preferred when dealing with complex queries and relationships where transactional integrity is important.
- NoSQL Databases are chosen for their flexibility with data schemas and efficiency in handling large volumes and varieties of data, making them suitable for big data applications, real-time analytics, and handling rapidly changing designs.

## Functionality of Data Management Tools

These tools and formats are crucial for the following reasons:

- **Storage:** They provide systems to store large volumes of data securely.
- **Management:** Offer features to update, delete, and manage data efficiently.
- **Retrieval:** Enable quick and flexible retrieval of data, which is essential for analysis and reporting.

## Common Use Cases

**Long-term Data Analysis:** Databases are ideal for projects where data needs to be stored long-term for complex analysis and historical data tracking.

**Data Sharing:** CSV and JSON formats are frequently used for data interchange between different programs or frameworks, making them suitable for projects where data needs to be exported and used in different environments.

**Real-time Applications:** NoSQL databases like MongoDB are used in applications that require real-time data access and updates.

## Integration and Workflow

In a comprehensive web scraping setup, the workflow typically involves several stages:

**Data Acquisition:** Data is retrieved from the web using HTTP libraries like Requests.

**Data Parsing and Extraction:** Libraries like BeautifulSoup or lxml are used to parse and extract useful information from the HTML/XML data.

**Dynamic Content Handling:** For dynamic content, tools such as Selenium or Puppeteer are employed to interact with the webpage.

**Data Cleaning:** Before storing, data often needs to be cleaned and transformed to ensure quality.

**Data Storage:** Finally, the cleaned data is stored using suitable data management tools. For instance, structured data may be stored in SQL databases, while JSON from dynamic content scraping might be stored directly in MongoDB.

This integration not only facilitates the smooth transition of data through different stages of processing but also ensures that the data remains consistent, secure, and accessible throughout the lifecycle of the web scraping project. Effective data management is crucial for leveraging the full potential of scraped data, allowing businesses and analysts to generate actionable insights and drive decision-making.

## Integration and Workflow

In a typical web scraping setup, an HTTP library like Requests is used to retrieve web pages which are then passed to a parsing library such as BeautifulSoup. The extracted data might be interactively processed with Selenium or Puppeteer if dynamic content is involved. Finally, the data is cleaned and stored using appropriate data management tools.

## **Data Formats for Web Scraping**

Data formats are essential for storing, transferring, and processing data extracted through web scraping. The choice of format can significantly influence the ease of data manipulation and integration into various applications. While CSV and JSON are commonly used, there are several other formats that cater to specific needs and complexities of data projects.

### **Commonly Used Data Formats**

#### **1. CSV (Comma-Separated Values)**

Description: CSV is a simple file format used to store tabular data, such as spreadsheets or databases. Each line of the file corresponds to a data record, with commas separating each field within the record.

Functionality: CSV files are straightforward and supported by most data processing applications, making them a universal choice for data exchange.

Use Cases: Ideal for storing data with a simple, flat structure without nested fields. Commonly used in data import/export scenarios, lightweight analytics, and situations where readability is important.

#### **2. JSON (JavaScript Object Notation)**

Description: JSON is a text-based format with "objects" structured as key/value pairs and arrays. It is highly readable and commonly used in web development.

Functionality: JSON's structure mirrors many programming languages' native data structures, which simplifies data interchange in web technologies.

Use Cases: Perfect for data with nested structures and for use in web applications, APIs, and configurations where interoperability between platforms and languages is needed.

### **Additional Data Formats**

#### **3. XML (eXtensible Markup Language)**

Description: XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

Functionality: Like HTML, XML uses tags to define elements but is designed to be self-descriptive. XML files are highly structured and can support complex data structures with nested and repeated elements.

Use Cases: Commonly used in web services (SOAP), rich internet applications, and situations where data needs extensive structure and metadata support.

#### **4. YAML (YAML Ain't Markup Language)**

Description: YAML is a human-readable data serialization format that supports all the primary data types used in popular programming languages.

Functionality: It uses a more readable format than JSON and is particularly good for configuration files. YAML allows complex data structures to be represented hierarchically.

Use Cases: Ideal for configuration files, data storage, and inter-process messaging where human readability and data complexity handling are paramount.

## **5. Parquet**

Description: Parquet is a columnar storage file format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model, or programming language.

Functionality: It is optimized for use with complex nested data structures and performs well for data compression and encoding schemes, which allows efficient data storage and retrieval.

Use Cases: Best suited for heavy read-write operations and big data processing tasks typically found in data science and machine learning projects.

## **6. Protocol Buffers**

Description: Developed by Google, Protocol Buffers are a language-neutral, platform-neutral, extensible way of serializing structured data for use in communications protocols, data storage, and more.

Functionality: Known for their efficiency and smaller size, they are an alternative to JSON or XML and require a schema to be defined using which the data is encoded and decoded.

Use Cases: Ideal for developing programs that communicate with servers and for data storage where quick and compact data transmission is needed.



### Challenges of Web Scraping

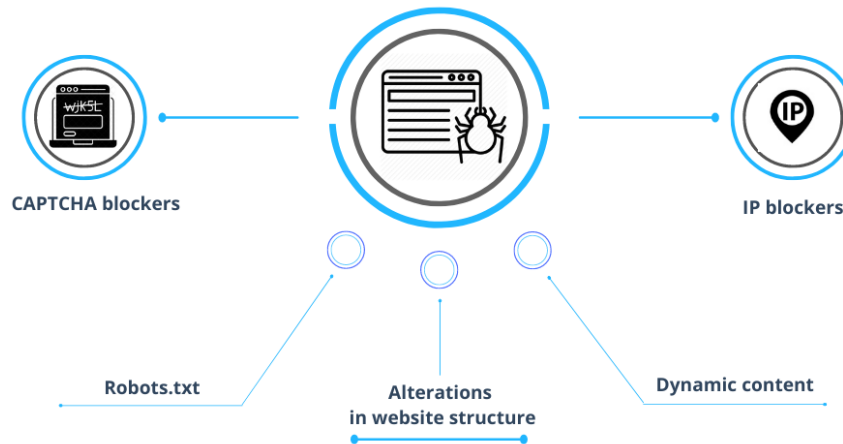


Figure 4: Challenges in web scraping.

While web scraping is a powerful tool, it comes with its challenges. Websites often change their layout and coding structures, which can break the scraping setup. Furthermore, ethical and legal considerations must be taken into account to ensure compliance with data privacy laws and website terms of service. Scrapers must be designed to respect robots.txt files and avoid overwhelming website servers with high-frequency requests. Some more challenges are the following:

**Data Quality:** Extracted data can sometimes be messy or incomplete, requiring significant cleaning and processing to become useful.

**Complex Data Structures:** Web pages often have nested and complex data structures that can be challenging to navigate and parse effectively.

**Handling JavaScript:** Many modern websites use JavaScript to load data dynamically. Tools that can execute JavaScript and handle AJAX calls are necessary for these cases.

**CAPTCHA Blockers:** CAPTCHAs are mechanisms designed to determine whether the user is human and are commonly used on websites to prevent automated data scraping. They pose a significant hurdle as they require solving tasks that are typically easy for humans but difficult for automated scripts.

**IP Blockers:** Many websites monitor the IP addresses from which they receive requests. If a single IP address makes too many requests within a short period, it can be blocked from accessing the site. This is done to prevent overload on the server and to thwart scraping efforts.

**Alterations in Website Structure:** Websites frequently update their layout and underlying HTML structure, which can disrupt web scraping scripts. Scrapers that were functional one day can stop working the next if the targeted elements of the webpage change or are removed.

**Robots.txt:** This is a file used by websites to communicate with web crawlers and other web robots. The file tells the robot which areas of the website should not be processed or scanned. Respecting "robots.txt" is crucial for ethical web scraping practices.

**Dynamic Content:** Web pages that load content dynamically with JavaScript present another layer of complexity. Traditional scraping tools that simply download the HTML of a page will not capture content loaded via JavaScript, necessitating the use of more advanced tools like Selenium or Puppeteer, which can simulate a real user's interaction with the browser.

**Legal and Ethical Considerations:** It is crucial to ensure that web scraping activities comply with legal regulations and respect the data privacy and usage policies specified by websites.

### **Integrating Challenges into the Context of Web Scraping**

These challenges are important considerations in the design and execution of web scraping tasks:

**Technical Adaptations:** To effectively manage CAPTCHA, dynamic content, and changes in website structure, sophisticated scraping setups using headless browsers or integrating AI to solve CAPTCHAs may be required.

**Ethical Web Scraping:** Respecting "robots.txt" and ensuring that scraping activities do not harm the website's normal operation or violate privacy laws is essential for maintaining compliance and protecting the scraper's legal standing.

**Mitigation Strategies for IP Blocking:** Using proxy servers or a network of rotating IPs can help mask the scraping activity, reducing the risk of being blocked by the website's security measures.

Addressing these challenges requires a balanced approach that considers both the technical hurdles and the ethical implications of web scraping. By understanding and navigating these challenges, developers can design more resilient, effective, and compliant web scraping solutions.

### **Application of Web Scraping in Real Estate**

Web scraping has become a transformative tool in the real estate industry, revolutionizing how data is collected, analyzed, and utilized for market insights and decision-making. This section of the thesis explores the various applications of web scraping in real estate, focusing on the types of data collected and their impact on the industry.

The real estate market thrives on information. The accuracy, accessibility, and timeliness of this information can significantly affect the dynamics of the market. Web scraping facilitates the extraction of vast amounts of data from multiple sources, such as property listing sites, auction sites, and real estate marketplaces. This process not only enhances the breadth and depth of data available but also increases its reliability and timeliness. The key data points collected include:

## Data Collection in Real Estate

The real estate market thrives on information. The accuracy, accessibility, and timeliness of this information can significantly affect the dynamics of the market. Web scraping facilitates the extraction of vast amounts of data from multiple sources, such as property listing sites, auction sites, and real estate marketplaces. This process not only enhances the breadth and depth of data available but also increases its reliability and timeliness. The key data points collected include:

**Property Prices:** Price data is fundamental in real estate as it directly influences investment decisions and market analyses. Web scraping automates the collection of updated property prices across different regions, enabling stakeholders to track market trends, assess property valuations, and perform comparative market analyses.

**Property Descriptions and Features:** Detailed descriptions and features provide insights into the usability and appeal of properties. Scraping these details helps in compiling comprehensive profiles of properties, including size, number of rooms, amenities, and unique attributes. This data is crucial for buyers, real estate agents, and analysts to understand property characteristics that influence purchasing decisions.

**Geographical Information:** Location is a critical factor in real estate, often dictating property value more than any other feature. Geographical data scraped from online sources is used for geospatial analysis, which helps in understanding property value distribution, identifying regional market trends, and planning for urban development.

**Images:** Images play a pivotal role in real estate listings as they provide a visual representation of the property, significantly impacting buyer interest and perceptions. Scraping images allows for the analysis of property conditions, architectural styles, and presentation quality. Advanced image analysis can further aid in automated valuation models (AVMs) where property values are estimated based on visual cues.

## Impact on the Real Estate Industry

The integration of web scraping in real estate has led to several transformative impacts:

**Enhanced Market Transparency:** With more data readily available, the market becomes more transparent, allowing buyers and sellers to make more informed decisions.

**Operational Efficiency:** Automation of data collection reduces the manpower and time required, increasing operational efficiency for real estate businesses.

**Advanced Analytics and Predictive Modeling:** The availability of large, detailed datasets enables more sophisticated analyses, including predictive modeling of property prices and market demand forecasting.

**Improved Customer Engagement:** Real estate platforms can provide richer, more detailed listings that enhance customer engagement and satisfaction.

In conclusion, web scraping is not just a technical tool but a strategic asset in the real estate sector. It empowers stakeholders with data-driven insights that drive better decision-making and fosters a more dynamic and competitive market environment. As technology evolves, the scope and accuracy of web scraping will continue to expand, further enhancing its value to the real estate industry. This thesis section underscores the critical role of web scraping in shaping the future of real estate analytics and market strategy.

**Legal and Ethical Considerations:**

The legality of web scraping varies by country and can depend on several factors, such as the nature of the data collected and the way it is used. This subsection explores key legal frameworks such as the Computer Fraud and Abuse Act (CFAA) in the U.S. and similar regulations in the EU, particularly under the GDPR. Ethical considerations are also discussed, emphasizing respect for website terms of service, the avoidance of server overloads, and the anonymization of collected data to protect privacy.



## Machine Learning in Real Estate

### Application of Machine Learning in Real Estate

Machine learning (ML) has become an indispensable technology in the real estate sector, primarily due to its ability to process and analyze vast datasets quickly and accurately. ML has emerged as a transformative force in the real estate sector, offering unprecedented insights and capabilities that were previously unattainable with traditional analytical methods. By leveraging ML, industry professionals can now automate complex processes, enhance property listings, and conduct predictive market analyses with greater precision and efficiency. This section explores the pivotal applications of machine learning in real estate, focusing on image recognition and predictive modeling, and examines their profound impact on the industry.

#### 1. Image Recognition

Image recognition has fundamentally transformed the real estate industry by enhancing the way properties are visualized, analyzed, and evaluated on digital platforms. Leveraging sophisticated machine learning models, especially Convolutional Neural Networks (CNNs), image recognition technologies enable the automatic extraction and categorization of visual information from property photos. This capability not only streamlines the listing process but also significantly enriches the data quality presented to potential buyers.

**Feature Detection:** ML algorithms can identify and classify a wide range of features from property images, such as room types, landscaping attributes, and the presence of luxury amenities like swimming pools or elaborate interiors. This automated classification helps in enriching property listings with detailed, accurate descriptors that improve searchability and match potential buyers' preferences.

**Condition Assessment:** Beyond basic feature detection, image recognition can assess the condition of a property by analyzing signs of wear, the age of installations, and overall maintenance. This capability enables more accurate valuations and helps buyers visualize potential upkeep costs.

#### Theoretical and Mathematical Background

##### Convolutional Neural Networks (CNNs):

CNNs are a class of deep neural networks that are particularly effective for analyzing visual imagery. They are structured to recognize patterns from pixel data of images, learning hierarchical representations.

**Architecture:** A typical CNN architecture consists of an input layer, multiple convolutional layers, pooling layers, fully connected layers, and an output layer. The convolutional layers apply various filters to the input to create feature maps that capture essential visual features like edges, textures, and more complex patterns in deeper layers.

##### Mathematical Operation:

- **Convolution Operation:** In the convolutional layers, the neuron's response at a certain position is calculated as a dot product of the neuron's weights with the input volume within a receptive field. Mathematically, it is represented as:

$$f(x, y) = \sum_m \sum_n I(x + m, y + n) \cdot K(m, n)$$

where  $f(x, y)$  is the output at position  $(x, y)$ ,  $I$  is the input image, and  $K$  is the kernel/filter matrix.

**Pooling:** This operation reduces the dimensionality of each feature map but retains the most important information. Common pooling methods include max pooling and average pooling.

## Key Algorithms and Their Applications in Real Estate Image Analysis

### 1. Feature Detection with Convolutional Neural Networks (CNNs)

#### Objective:

To automate the detection and classification of various architectural and design features from property images, such as room types and specific amenities.

#### Theoretical and Mathematical Background:

- Convolutional Neural Networks (CNNs) are a specialized kind of neural network for processing data that has a grid-like topology, such as images. CNNs employ a mathematical operation called convolution in at least one of their layers.
- Convolutional Layer: The core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the filter and input, producing a 2D activation map of that filter. As a result, the network learns filters that activate when they see specific types of features at some spatial position in the input.

$$f_{i,j} = \text{Activation} \left( \sum_m \sum_n I_{i+m,j+n} \cdot K_{m,n} + b \right)$$

Where:

I is the input image,

K is the kernel/filter,

b is the bias,

$f_{i,j}$  is the feature map.

**ReLU Layer:** This layer applies the non-linear function  $\max(0,x)$  element-wise. It introduces non-linearity to the system, allowing the model to learn more complex functions.

**Pooling Layer:** This layer performs a downsampling operation along the spatial dimensions (width, height), resulting in spatial invariance to input distortions.

#### Process:

Trained CNN models scan images and utilize learned filters to identify various features such as shapes, colors, and patterns. Features detected include room types, presence of amenities like swimming pools, and interior design styles. These features are then classified into predefined categories (e.g., bedrooms, bathrooms).

### 2. Condition Assessment using Image Analysis

#### Objective:

To evaluate the physical condition of a property by analyzing visual cues in images that indicate wear, age, or maintenance level.

### **Theoretical and Mathematical Background:**

Advanced CNNs or other image processing algorithms are employed to analyze textural and color features in property images. These algorithms might use additional layers or techniques like:

- **Edge Detection:** Using filters that identify edges and contours, crucial for spotting cracks or wear.
- **Texture Analysis:** Leveraging textural patterns identified by CNNs can help in assessing the condition of surfaces and installations.

### **Process:**

CNNs analyze images to detect signs of aging or wear such as cracks, damp spots, or peeling paint. Textural and color changes are quantified, and a condition score is computed, which aids in accurate property valuation.

### **Practical Implementation**

**Training Data:** A substantial dataset of real estate images labeled with both features and conditions is essential. This dataset must represent a wide variety of property types and conditions to train effective models.

**Model Training:** CNNs are trained using backpropagation and gradient descent algorithms. These models iteratively adjust their weights based on the error rate of outputs compared to the training labels, minimizing the prediction error.

**Integration into Platforms:** Once trained, these models are integrated into real estate platforms. They automatically process new property images uploaded by sellers, providing instant insights into the features and condition of the property.

The application of CNNs in real estate image analysis represents a significant advancement in automating property evaluation. By accurately identifying features and assessing conditions through learned models, real estate platforms can offer more detailed and accurate information, enhancing both seller and buyer experiences. The mathematical grounding of these models ensures that they can learn to recognize and generalize from the training data provided, making them robust tools in the digital transformation of real estate assessments.

### **Challenges and Considerations**

**Data Variability:** Real estate images can vary greatly in terms of angle, lighting, and quality, which can affect the accuracy of feature detection and condition assessment.

**Model Generalization:** Models trained on data from specific regions or types of properties might not perform well on others unless adequately generalized.

**Ethical Considerations:** Ensuring that the automated assessments are transparent and do not inadvertently bias against certain property types or locations.

Image recognition via machine learning, particularly through the use of CNNs, represents a significant advancement in the way real estate properties are presented and analyzed. By automating the extraction of detailed, accurate visual information, these technologies not only enhance the appeal and transparency of real estate listings but also provide valuable insights that aid in the decision-making process for buyers and sellers alike.

## Predictive Modeling in Real Estate

Predictive modeling in real estate leverages statistical and machine learning techniques to analyze historical data and predict future outcomes. These models are built upon foundational principles from statistics, probability, and computer science. Predictive modeling uses historical data and machine learning algorithms to forecast future trends in the real estate market. These models are crucial for both macro and micro-level decision-making processes in real estate.

**Price Prediction:** By training on variables such as historical prices, location demographics, market conditions, and property features, ML models can predict future property prices with significant accuracy. This is invaluable for investors looking to buy at the right price and sell for the best possible return.

**Market Demand Forecasting:** Machine learning models analyze trends and patterns to forecast future market behaviors. This includes predicting up-and-coming neighborhoods, changes in consumer demand, and potential market disruptions. Such insights help real estate professionals and policymakers in strategic planning and resource allocation.

### 1. Regression Analysis

Regression models are fundamental in predictive modeling for price prediction. These models establish a relationship between a dependent variable (property prices) and one or more independent variables (such as location, demographics, and property features).

- **Linear Regression:** The simplest form of regression, useful for predicting property prices based on linear relationships. The model assumes a straight-line relationship between the dependent and independent variables.

$$Y = \beta_0 + \beta_1 \cdot X_1 + \beta_2 \cdot X_2 + \dots + \beta_n \cdot X_n + \varepsilon$$

Where:

- Y is the property price.
- $\beta_0$  is the y-intercept.
- $\beta_1, \beta_2, \dots, \beta_n$  are the coefficients which represent the effect of each variable.
- $X_1, X_2, \dots, X_n$  are the independent variables.
- $\varepsilon$  is the error term.
- **Multivariate Regression:** Multivariate regression, an extension of simple linear regression, involves predicting a dependent variable using multiple independent variables. This statistical technique is used to model and analyze relationships where the outcome variable is influenced by more than one predictor variable. It is especially useful in scenarios where various factors contribute to the outcome, such as in real estate, where property prices can be influenced by location, size, number of rooms, age, and many other factors.

### Mathematical model

The general form of the multivariate regression model can be expressed as:

$$Y = \beta_0 + \beta_1 \cdot X_1 + \beta_2 \cdot X_2 + \dots + \beta_n \cdot X_n + \varepsilon$$

Where:

- Y is the dependent variable (e.g., property price).
- $\beta_0$  is the intercept term, representing the expected mean value of Y when all independent variables X are equal to zero.
- $\beta_1, \beta_2, \dots, \beta_n$  are the coefficients of the model. Each coefficient represents the change in the dependent variable for one unit of change in the corresponding independent variable, holding all other variables constant.
- $X_1, X_2, \dots, X_n$  are the independent variables (e.g., square footage, number of bedrooms, age of the property, proximity to the city center, etc.).
- $\epsilon$  is the error term, representing the residual effect unexplained by the independent variables.

### Estimation of Coefficients

The coefficients  $\beta_i$  are typically estimated using the method of least squares, which minimizes the sum of the squared residuals, providing the best linear unbiased estimators under the Gauss-Markov theorem, assuming the usual OLS assumptions (no perfect multicollinearity, exogeneity, homoscedasticity, and normality of errors) are met.

$$\sum_{i=1}^n \left( Y_i - (\beta_0 + \beta_1 \cdot X_{i1} + \beta_2 \cdot X_{i2} + \dots + \beta_p \cdot X_{ip}) \right)^2$$

Where:

$Y_i$  is the observed value of the dependent variable for the i-th observation.

$X_{i1}, X_{i2}, \dots, X_{in}$  are the observed values of the independent variables for the ii-th observation.

### Use in Real Estate

In real estate applications, multivariate regression can provide valuable insights into how different features contribute to the value of a property. For instance:

- Understanding which features (like a renovated kitchen or a garage) add the most value to a home.
- Analyzing regional pricing trends by including location-specific variables.
- Adjusting pricing strategies based on the predictive influence of specific market conditions or property attributes.

### Challenges

While powerful, multivariate regression faces several challenges:

**Multicollinearity:** When independent variables are highly correlated, it can be difficult to determine the individual effect of each variable on the dependent variable.

**Overfitting:** Including too many variables without sufficient data can lead to a model that fits the training data too closely but performs poorly on unseen data.

**Interpretability:** As more variables are included, the model can become more difficult to interpret, especially in terms of understanding the effect of each variable while holding others constant.

## 2. Decision Trees and Random Forests

These are non-linear models that are particularly useful for handling categorical data and interactions between multiple variables in real estate data.

**Decision Tree:** Splits the data into subsets based on the value of the input features. The splits are chosen to minimize the heterogeneity of the subsets.

$$\text{Information Gain} = \text{Entropy (Parent)} - \sum_{child} \left( \frac{N_{child}}{N} \cdot \text{Entropy (Child)} \right)$$

**Random Forest:** An ensemble method that uses multiple decision trees to reduce the risk of overfitting. It combines the predictions from multiple trees to produce a more accurate prediction.

## 3. Time Series Analysis

For forecasting market demand and identifying trends, time series models like ARIMA (Autoregressive Integrated Moving Average) are used.

- **ARIMA Model:** This model is used for analyzing and forecasting time series data, allowing for data trends, seasonality, and patterns.

$$Y_t = \alpha + \beta_1 \cdot Y_{t-1} + \beta_2 \cdot Y_{t-2} + \dots + \beta_p \cdot Y_{t-p} + \varepsilon_t$$

Where  $Y_t$  is the value at time  $t$ , and  $\varepsilon_t$  is the error at time  $t$ .

## Challenges in Implementing Machine Learning

While machine learning offers numerous benefits to the real estate sector, several challenges need to be addressed to maximize its potential:

**Data Quality and Availability:** The effectiveness of any ML model is heavily dependent on the quality and granularity of the data fed into it. In real estate, inconsistent data entries, missing values, and limited access to comprehensive datasets can impede the accuracy of predictive models.

**Handling High Dimensionality:** Real estate datasets often contain a high number of variables, from basic property characteristics to more nuanced features like neighborhood crime rates or school district quality. Managing such high dimensionality without overfitting the model requires sophisticated feature selection and regularization techniques.

**Bias and Fairness:** Machine learning models can inadvertently perpetuate or amplify biases present in the historical data, leading to unfair outcomes for certain demographic groups. Ensuring models are fair and unbiased is crucial for ethical AI practices in real estate.

## Image Similarity Methods

### Image similarity on real estate properties

Image similarity methods leverage advanced computational techniques to analyze visual data and identify similarities between images. In the context of real estate, these methods are instrumental in detecting duplicate listings, comparing property features, and enhancing search functionalities by grouping similar properties. This section delves into the theoretical and mathematical foundations of image similarity techniques, exploring their application in real estate for efficient and accurate property analysis.

Image similarity methods are integral to modern computer vision applications, especially in sectors like real estate where visual content plays a pivotal role in transactions. These methods process and analyze visual data to identify patterns, similarities, and differences between images. By doing so, they provide valuable insights that can automate and enhance various aspects of real estate operations. Image similarity assessment in real estate involves comparing visual content of property images to determine how similar they are to one another. This can be used to identify duplicates, suggest similar properties, and even categorize listings based on visual features such as architecture style, interior design, and landscaping.

### Key Techniques and Their Mathematical Foundations

#### 1. Feature Extraction

Feature extraction is one of the fundamental techniques used in image similarity. It involves detecting keypoints (distinctive features) in images and finding matches between them. This is the process of identifying key elements within images that are significant for analysis and comparison. Feature extraction is a critical step in image processing and computer vision, particularly in the context of image similarity and object recognition. It involves isolating important visual cues within an image that are distinctive and robust against various changes such as viewpoint, scale, and illumination.

#### SIFT (Scale-Invariant Feature Transform):

One of the most effective and widely used methods for feature extraction is the Scale-Invariant Feature Transform (SIFT), which is particularly adept at identifying and describing local features in images. SIFT detects and describes local features in images. The algorithm is robust to changes in scale, noise, and illumination. SIFT is designed to extract distinctive keypoints that are invariant to image scale and rotation, and partially invariant to changes in illumination and 3D camera viewpoint. These features are then used to perform reliable matching between different views of an object or scene.

Scale-Space Extrema Detection: SIFT begins by constructing a scale space, which is essentially a function,  $L(x, y, \sigma)$ , that represents the image at various scales. This is achieved by convolving the image  $I(x, y)$  with a Gaussian blur  $G(x, y, \sigma)$  at different values of  $\sigma$ :

$$L(x, y, \sigma) = G(x, y, \sigma) \cdot I(x, y)$$

Where  $L$  is the scale-space of an image  $I$ ,  $G$  is the Gaussian Blur function,  $\sigma$  is the scale variable, and  $*$  denotes convolution. Keypoints are identified at maxima and minima of the difference-of-Gaussian function applied in scale-space.

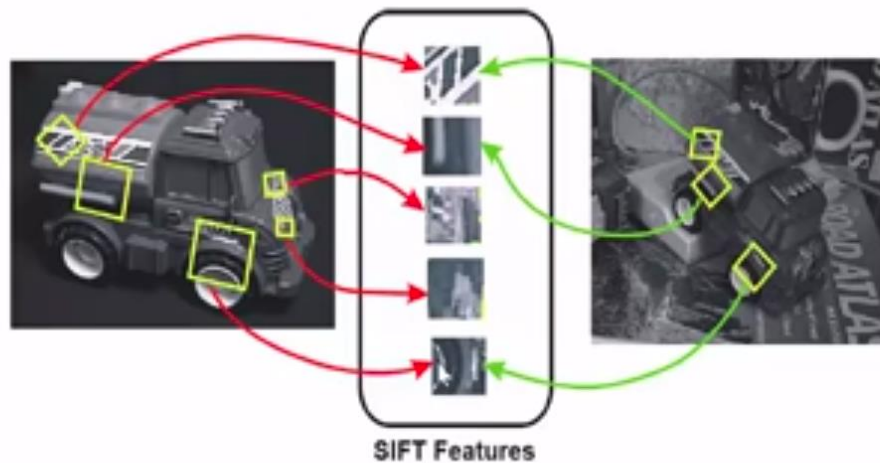


Figure 5:SIFT( Scale Invariant Feature Transform)

The figure (figure 5) is a visual representation of the Scale-Invariant Feature Transform (SIFT) technique applied to two distinct images to identify and match key feature points. Left Image features an object (a toy truck) from one perspective. Right Image shows the same object from a different angle and context, potentially a magazine page or an advertisement. Highlighted with yellow boxes on both images, these points represent distinctive features detected by the SIFT algorithm. These features are chosen because they are unique and can be easily recognized in both images despite changes in scale, orientation, or illumination. The lines connecting the two images illustrate the matching feature points between them. Each line represents a successful match based on the similarity of the descriptors. Red lines could indicate matches that involve more significant transformations or discrepancies, while green lines might represent more straightforward or confident matches.

In real estate, SIFT can be used to match images of properties from different sources to identify duplicate listings or to link multiple photographs of the same property across various platforms. Additionally, these features can help in categorizing properties based on visual styles or detecting changes in property conditions over time.

SIFT provides a robust method for extracting and describing features in images, which are crucial for tasks involving image similarity, object recognition, and classification. Its ability to detect keypoints that are invariant to common image transformations makes it invaluable in both academic research and practical applications like real estate image analysis

### **SURF (Speeded Up Robust Features):**

Similar to SIFT but faster, using a box filter approximation of the LoG (Laplacian of Gaussian). SURF (Speeded Up Robust Features) is an enhanced version of the SIFT (Scale-Invariant Feature Transform) algorithm, designed to improve speed and efficiency in detecting and describing local features in images. While maintaining many of the robust properties of SIFT, SURF makes several modifications that increase computation speed, making it well-suited for real-time applications.

SURF relies on the concept of integral images to speed up the feature detection process. Integral images allow for rapid calculation of image features at different scales, which is a cornerstone of the SURF methodology.



## Key Differences from SIFT

**Integral Images:** These are used in SURF for image convolutions, which allows the algorithm to compute the sum of intensities in a given image region rapidly.

**Box Filters:** SURF uses box filters as approximations of second-order Gaussian derivatives, which significantly speeds up the calculation compared to the more traditional filter methods used in SIFT.

### Difference-of-Gaussian (DoG) Approximation:

SURF approximates the Laplacian of Gaussian (LoG), which SIFT computes explicitly, using a simpler Difference-of-Gaussian obtained through box filters. This approximation reduces computational complexity.

Formula for Difference-of-Gaussian:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \cdot I(x, y)$$

Where:

- $D(x, y, \sigma)$  represents the difference-of-Gaussian achieved with box filters.
- $G(x, y, k\sigma)$  and  $G(x, y, \sigma)$  are Gaussian Blurs at different scales.
- $\cdot$  denotes the convolution operation.

## 2. Structural Similarity Index (SSIM)

SSIM is a method for measuring the similarity between two images. It considers texture, luminance, and contrast. Unlike simple pixel-by-pixel comparison, SSIM evaluates patterns of pixel intensities that have been normalized for luminance and contrast. The Structural Similarity Index (SSIM) is a sophisticated metric used to measure the similarity between two images. Unlike traditional methods that rely on direct pixel-by-pixel comparisons, SSIM assesses the perceptual impact of three characteristics of an image: luminance, contrast, and structure. By considering these aspects, SSIM provides a more accurate reflection of visual similarity as perceived by human vision. SSIM is based on the premise that human visual perception is highly adapted for extracting structural information from the visual field, thus the similarity between two images can be more precisely measured by comparing their structural features rather than their individual pixel values.

**Application in Real Estate:** SSIM can be used to compare quality and style of property images, assessing how similarly two properties are presented in terms of lighting, angles, and staging.

The equation of SSIM can be expressed like:

$$SSIM(x, y) = \frac{(2\mu_x + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Where:

$\mu_x, \mu_y$  are the average intensities of images x and y.

$\sigma_x, \sigma_y$  are the variances.

$\sigma_{xy}$  is the covariance.

$c_1, c_2$  are constants to stabilize the division.

### 1.1.1. Application in Real Estate

In real estate, SSIM can be effectively used to enhance several aspects of property marketing and management:

- **Quality Control of Images:** Ensuring that property photos uploaded to listing services meet certain quality criteria in terms of lighting and clarity, which can influence buyer perceptions and decisions.
- **Comparison of Property Images:** SSIM can be used to automatically compare and categorize property images by style and visual quality, helping potential buyers find properties that match their aesthetic preferences.
- **Detecting Duplicate Images:** Real estate platforms can use SSIM to detect duplicate listings or fraud by identifying properties that are represented by significantly similar images but different metadata.

#### Advantages of SSIM

- **Alignment with Human Perception:** SSIM is designed to reflect the way humans perceive image quality, which makes it particularly useful in applications where the visual quality of images is paramount.
- **Robustness:** SSIM is robust against variations in image resolution and the addition of noise, making it suitable for analyzing images from various sources and quality.

#### Limitations

**Sensitivity to Scaling and Rotation:** While robust to changes in lighting and contrast, SSIM may still be sensitive to scaling and rotations. Additional preprocessing steps might be needed to align images properly before comparison.

**Computational Complexity:** While not as computationally intense as some deep learning methods, SSIM still requires a significant amount of processing power, especially when applied to large datasets commonly found in real estate databases.

In summary, SSIM provides a useful metric for assessing image quality and similarity in a manner that correlates well with human visual perception, making it an excellent tool in fields like real estate where visual presentation plays a crucial role.

#### CNN Layer Calculation Formula for Feature Extraction

Convolutional Neural Networks (CNNs) can be trained to extract hierarchical features and determine image similarity. Convolutional Neural Networks (CNNs) are a class of deep neural networks that are particularly effective for tasks involving image data. CNNs utilize multiple layers of convolutions with learnable weights to automatically extract hierarchical features, which are crucial for understanding and analyzing the content of images. Each layer in a CNN transforms the input data into more abstract and composite features, making CNNs highly effective for image recognition and similarity assessments. CNNs automatically learn the hierarchies of features which are crucial for understanding the content of the images.

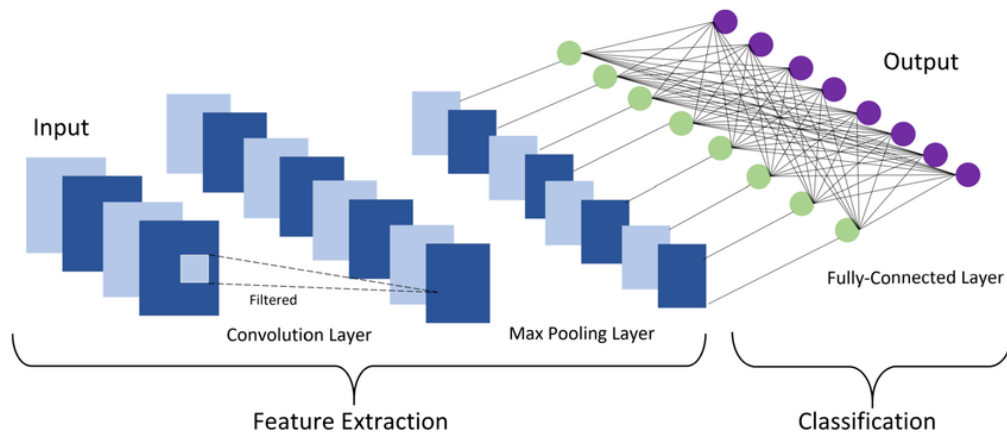


Figure 6: CNN for image classification.

The diagram (figure 6) illustrates the architecture of a Convolutional Neural Network (CNN), depicting the flow from input through multiple convolution and max pooling layers for feature extraction, to fully-connected layers that handle classification. Each stage is clearly represented, showing how the network processes and transforms input image data into a final output class through learned filters and spatial reduction.

### CNN Layer Operations

Feature Maps  $(f_i^{(l)})$ :

Description: Feature maps are the outputs of each layer in the CNN. At each layer, the feature map represents the result of applying learned filters (or kernels) to the input, capturing specific features at different levels of abstraction. For instance, early layers might detect edges or textures, while deeper layers might identify complex objects or patterns within the image.

The feature map at layer  $l$  for the  $i$ -th filter is denoted as  $(f_i^{(l)})$ .

Weights  $(W_{ij}^{(l-1)})$ :

Weights are the parameters of the filters that are learned during the training process. Each weight corresponds to a small receptive field in the input and determines how the input's features contribute to the resulting feature map.

The weights connecting the  $j$ -th feature map of the previous layer  $(l-1)$  to the  $i$ -th feature map of the current layer  $l$  are represented by  $(W_{ij}^{(l-1)})$ .

Biases  $(b_i^{(l)})$ :

Biases are additional parameters in CNNs added to each feature map after convolution but before the activation function. They allow the activation function to shift left or right, which can be critical for learning patterns in the input data.

$(b_i^{(l)})$  is the bias term associated with the  $i$ -th feature map at layer  $l$ .

**Activation Functions ( $\sigma$ ):**

After convolution, an activation function is applied to introduce non-linearity into the model. This step is crucial because it allows the network to learn complex patterns.

**Common Choices:** ReLU (Rectified Linear Unit) is the most commonly used activation function in CNNs due to its simplicity and effectiveness. It is defined as  $\sigma(x)=\max(0,x)$ , effectively turning off neurons that produce negative outputs, which simplifies the computation and introduces sparsity in the activations.

The convolution operation involves sliding each filter across the input and computing a dot product between the filter and input at each position. This operation extracts spatial features from the input and produces a feature map. Layers are formulated as convolutions with learned weights followed by non-linear activation functions:

$$f_i^{(l)} = \sigma \left( \sum_j W_{ij}^{(l-1)} * f_j^{(l-1)} + b_i^{(l)} \right)$$

Where:

$f_i^{(l)}$  is the feature map  $i$  at layer  $l$ .

$W_{ij}^{(l-1)}$  are the weights from layer  $l-1$  to  $l$ .

$f_j^{(l-1)}$  are the feature maps from the previous layer.

$b_i^{(l)}$  are the biases at layer  $l$ .

$\sigma$  is the non-linear activation function (e.g., ReLU).

sum from  $j$  denotes the summation over all inputs from the previous layer.

\* indicates convolution operation.

### **Significance in Image Similarity**

In the context of image similarity, these hierarchical features extracted by CNNs can be used to compare images at various levels of abstraction, from simple textures and shapes to complex objects and scenes. This capability makes CNNs extremely valuable for tasks such as image classification, object detection, and feature-based image retrieval, where understanding the deeper semantic content of images is crucial.

CNNs transform the raw pixel data of images into a form that highlights their essential features while discarding irrelevant variations like noise and lighting changes. This transformation is fundamental to many modern applications of machine vision, particularly in areas like real estate, where automated analysis of property images can significantly enhance listing accuracy and user experience.

### **Application in Real Estate**

Image similarity methods can automate the detection of duplicate listings, significantly reducing the effort required to manage large databases. They also enhance user experience by recommending properties that visually match user preferences. Furthermore, these techniques can categorize properties based on architectural styles or condition, aiding in targeted marketing and accurate pricing.

The application of image similarity methods in real estate represents a significant advancement in how properties are analyzed and marketed. By leveraging the theoretical and mathematical frameworks of feature matching, structural similarity, and deep learning, real estate platforms can offer more precise and user-friendly services, improving both operational efficiency and customer satisfaction.

## Django for Full-Stack Development

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It is renowned for its ability to facilitate the creation of complex, data-driven websites with minimal coding and straightforward maintainability. Developed by experienced developers, Django abstracts many of the common challenges in web development, allowing developers to focus on writing their app without needing to reinvent the wheel.



Figure 7: Django architecture.

### Origins and Philosophy

Django was designed to handle the fast-paced newsroom deadlines while meeting the stringent requirements of experienced web developers. Named after the jazz guitarist Django Reinhardt, the framework was created with the intention of simplifying the creation of complex, database-driven websites. It emphasizes reusability and "pluggability" of components, rapid development, and the principle of not repeating oneself (DRY).

### Key Features

**MTV Architecture:** Django follows the Model-Template-View (MTV) architectural pattern, which is Django's take on the popular Model-View-Controller (MVC) architecture. The Model manages the database, the Template handles presentation and user interfaces, and the View executes business logic controlling what users can do with the data.

**Robustness:** Out-of-the-box support for common tasks such as user authentication, content administration, site maps, and RSS feeds make it highly robust for handling the complexities and security needs of modern web applications.

**Scalability:** Django uses a "shared-nothing" architecture, meaning you can add hardware at any level — database servers, caching servers, or web servers — to handle higher loads. This scalability makes it suitable for projects ranging from small websites to large-scale enterprise systems.

**Security:** Django provides built-in protections against many security threats like SQL injection, cross-site scripting, cross-site request forgery, and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.

## Usage in Full-Stack Development

As a full-stack framework, Django comes equipped with everything needed to build a web application from the ground up:

**Backend Logic:** It handles backend logic seamlessly with Python, one of the most straightforward and powerful programming languages in use today.

**Frontend Integration:** While Django manages the backend, it integrates easily with any frontend technology to deliver content in almost any format, including HTML, JSON, XML, and more. Frontend frameworks like React or Angular can be used to create a dynamic user interface.

**Database Management:** Django supports several major database engines and can work with any that supports the Python Database API. It comes with a built-in ORM (Object-Relational Mapping) to bridge the gap between the relational database systems and the business objects without needing to write SQL code.

## Using Django in Data-Driven Projects

Django, a high-level Python web framework, is particularly suited for data-driven projects like real estate platforms, which require handling large datasets and ensuring robust user interactions. Its pragmatic and clean design, rooted in the Model-View-Template (MVT) architectural pattern, offers a logical separation of concerns that streamlines the development of complex web applications.

## Architectural Benefits for Real Estate Platforms

### Batteries-Included Approach:

Django's comprehensive offering of built-in components for common web development tasks (e.g., authentication, URL routing, and session management) accelerates development timelines. This is invaluable in real estate platforms where security and user experience are paramount. Django's robust features reduce the need for custom coding, ensuring greater reliability and consistency across the application.

### Handling Large Datasets:

Django's ORM (Object-Relational Mapping) system is crucial for real estate platforms dealing with vast amounts of data, such as listings, transactions, user data, and market analytics. The ORM facilitates complex queries and data management without the direct use of SQL, minimizing the risk of injection attacks and making the system more secure and maintainable.

### Performance and Scalability:

Real estate platforms must handle variable traffic loads efficiently, especially during high viewing times. Django supports various caching strategies to enhance performance. This scalability is proven in its deployment in high-traffic environments like Instagram and Disqus, demonstrating its capability to manage heavy loads effectively.

Django's design is not only robust but also highly flexible, allowing it to integrate seamlessly with various technologies that enhance its functionality:

## Integration with Other Technologies

### **Machine Learning Integration:**

Integrating Django with machine learning libraries like TensorFlow or Keras enables the application to include sophisticated analytical tools directly in the web interface. This can be used to automate valuation models, recommendation systems, or image recognition tasks directly within the real estate platform. Incorporating machine learning with Django allows real estate platforms to perform advanced data analysis directly within the application. For instance:

- **Valuation Models:** Integrating TensorFlow or Keras to develop models that predict property prices based on historical data and market trends.
- **Recommendation Systems:** Using machine learning to suggest properties to users based on their browsing history and preferences.
- **Image Recognition:** Implementing algorithms to analyze property photos for features like space, condition, and style, enhancing listing details automatically.

### **Data Visualization Tools:**

Django can be easily integrated with JavaScript libraries like D3.js or Python libraries like Plotly to create dynamic data visualizations. These tools can provide interactive charts and graphs for real-time market analysis, enhancing user engagement and providing valuable insights at a glance. This is critical for:

- **Market Analysis:** Dynamic charts and graphs that analyze trends over time, regional market performances, and investment opportunities.
- **User Dashboards:** Customizable interfaces where buyers, sellers, and agents can view personalized data insights and metrics.

### **Database Management Systems:**

Django is compatible with several key database systems like PostgreSQL, MySQL, and MongoDB, which are capable of handling different facets of real estate data. PostgreSQL, for instance, offers advanced features such as full-text search and GIS (Geographical Information Systems) support which are ideal for real estate applications.

Django's ORM is a powerful feature that serves as a bridge between the database engine and Django models, allowing developers to manipulate database data without having to write raw SQL queries. Here's a more detailed look at how the Django ORM supports complex database operations, particularly within the context of real estate applications.

### **Core Features of Django ORM**

#### **Abstraction:**

Django ORM provides a high level of abstraction, letting developers interact with the database using Python code instead of SQL. This abstraction makes the code more readable, maintainable, and portable across different database systems such as PostgreSQL, MySQL, MongoDB (via Django), and SQL Server.

#### **Database Agnosticism:**

One of Django's strengths is its database-agnostic design. Whether you're using PostgreSQL, MySQL, SQLite, or Oracle, Django allows you to use the same application code, only requiring changes in the settings to switch



between these databases. This flexibility is particularly useful in real estate platforms where the choice of database might change based on scaling needs or specific features.

### **Querysets:**

Django ORM operates through querysets, which allow developers to construct a database query in Python code. Querysets are lazy, meaning they only hit the database when evaluated. This allows for efficient chaining and refinement of queries without extra database calls. For real estate applications, this means efficiently filtering properties, sorting listings by price or date, and aggregating data like average prices or counts of properties by type.

### **Model Relationships:**

The ORM supports various types of relationships such as foreign keys (one-to-many), many-to-many, and one-to-one relationships. For a real estate application, this can be used to relate users to property listings, track property ownership history, or link properties to multiple images and agents.

### **Migrations:**

Django's migration system handles changes to the database schema by creating migration files. These files describe how to adapt the database structure through a series of steps while preserving existing data. In the fast-evolving real estate market, where new data attributes might frequently be added (like new features or virtual tour links), migrations ensure that database changes are versioned and deployed smoothly.

### **Admin Interface:**

The Django admin interface is automatically generated from the models and is a powerful tool for sites' administrators to manage content in the database. For real estate platforms, this means easy administration of listings, user accounts, and other dynamic content without needing to build custom admin panels.

## **Benefits in Real Estate Applications**

- **Rapid Development:** Quick adjustments to the data model and easy manipulation of data suit the dynamic nature of real estate listings.
- **Data Integrity:** Django ORM helps maintain data integrity and consistency, which is crucial in real estate transactions and record-keeping.
- **Scalability:** Efficient query capabilities and support for powerful database features like full-text search and GIS make scaling real estate platforms more manageable.

Django's ORM is an integral part of the framework that significantly enhances development speed, data security, and application maintenance. In real estate platforms, where managing extensive data with high reliability is essential, Django ORM provides the tools necessary to build robust, scalable applications that can handle complex queries and vast amounts of data efficiently. Its database-agnostic nature further ensures that applications can adapt to different database backends as per changing business requirements.

### **2.9.2. Comparative Analysis**

To justify the choice of Django for this project, a comparative analysis with other popular web frameworks such as Flask, Node.js, and Ruby on Rails is conducted:

#### **Flask vs. Django:**

While Flask provides a lightweight and modular approach, which is excellent for smaller projects with less complex data operations, Django's built-in features and its ORM are more suited for handling the complex and data-intensive nature of real estate platforms.

#### **Node.js vs. Django:**

Node.js, being JavaScript-based, offers an advantage in building real-time applications with full-stack JavaScript. However, Django's extensive security features and mature third-party ecosystem make it a more reliable choice for the comprehensive features required in this project.

#### **Ruby on Rails vs. Django:**

Ruby on Rails is similar to Django in many respects, offering many built-in features and an opinionated framework structure. However, Django's Python-based syntax and widespread use in data science make it particularly apt for integrating with machine learning and analytics workflows, which are central to this project.

## Methodology

### Overview of the Web Scraping Pipeline

To comprehensively capture the landscape of the real estate market in Athens, the web scraping pipeline targeted several prominent Greek real estate platforms. Each platform presented unique challenges and required customized scraping strategies to effectively extract relevant data. This section elaborates on the specific platforms targeted, the nuances of each, and the strategies implemented to ensure thorough and efficient data collection.

The web scraping pipeline was designed to systematically collect and process data from major Greek real estate platforms, focusing on the aggregation of over 77,000 property listings primarily from the city of Athens. The aim was to capture a diverse array of data specific to different regions within Athens to facilitate detailed analysis and testing. The platforms that were scraped were [xe.gr](http://xe.gr), [plot.gr](http://plot.gr), [tospitimou.gr](http://tospitimou.gr), [spitogatos.gr](http://spitogatos.gr) and [spiti365.gr](http://spiti365.gr)

The web scraping pipeline for this thesis was designed to systematically collect real estate data from multiple prominent Greek real estate platforms. Given the scale of the task, involving over 77,000 property listings primarily in the city of Athens, a uniform scraping methodology was implemented across all platforms. This ensured consistency in data collection and streamlined the process for efficiency and reliability. This section details the uniform scraping approach, including the technical methods used and the specific platforms targeted.

### Uniform Scraping Methodology

To address the diversity of content and non-standardized formats across different real estate websites, the pipeline employed a combination of Python's requests library for data retrieval and BeautifulSoup for HTML content parsing. This method provided the flexibility and robustness necessary to navigate and extract data from various structured and semi-structured web pages commonly found in online real estate listings.

### Technical Strategy:

- **Data Retrieval:** The requests library was used to send HTTP requests to the targeted websites. This library was chosen for its ease of use and ability to handle various HTTP functionalities, which is crucial for accessing and downloading web page content.
- **Content Parsing:** BeautifulSoup was utilized to parse the HTML documents retrieved. It helped in systematically extracting the relevant data fields such as price, location, property size, and image links. BeautifulSoup is particularly effective in navigating complex HTML structures and extracting data with high accuracy.
- **Robust Error Handling:** Given the potential for request failures and HTML parsing errors, the pipeline included robust error handling mechanisms. These included retry logic for failed requests and exceptions handling to manage unexpected disruptions during the scraping process, ensuring data integrity and continuity.

## Targeted Real Estate Platforms

The following major Greek real estate platforms were scraped using the above uniform methodology, each offering a significant volume of listings which are vital for a comprehensive analysis of the Athens real estate market:

- Plot.gr
- Spiti24
- Spitogatos
- Tospitimou.gr
- Xe.gr

Each platform was approached with the same technical scraping strategy to maintain consistency across the dataset. This approach facilitated the aggregation of a robust dataset, ensuring that each listing was accompanied by standardized information necessary for subsequent analysis.

## Data Management

After collection, data from all platforms were consolidated into a single Microsoft SQL Server database. A relational schema was designed to accommodate the integration of data from various sources, with a flag system to identify the origin of each listing. This strategy not only simplified data management but also supported comprehensive data analysis by preserving the traceability of each data point back to its source.

## Data Integration and Storage

**Unified Database Schema:** The schema was crafted to support queries across multiple data points, enhancing the ability to perform comparative analyses across different data sets.

**Image Data Handling:** Links to property images were stored within each listing's row in the database. Images were later downloaded to a local server for processing by the image similarity model, crucial for identifying duplicate listings or visually similar properties.

This uniform approach to web scraping across multiple Greek real estate platforms has successfully created a foundational dataset for this thesis. The methodology ensures that the data collected is not only extensive and representative of the market in Athens but also uniformly structured for advanced analytical processes, including machine learning and comprehensive market analysis in subsequent stages of this study.

## Design Considerations

The design of the web scraping pipeline was guided by several key objectives:

**Comprehensiveness:** To ensure a dataset that is representative of the real estate market in Athens, encompassing a variety of property types, locations, and pricing.

**Scalability:** The pipeline needed to handle large volumes of data efficiently, allowing for future expansions to include more regions or platforms without significant redesign.

**Robustness:** The pipeline had to be capable of operating continuously and reliably, with mechanisms to handle potential disruptions such as changes in website layout or temporary network issues.

**Compliance and Ethics:** It was imperative that the scraping activities complied with all legal standards and ethical guidelines, particularly concerning data privacy and platform terms of service.

## Technical Architecture

The pipeline's architecture was structured around several core components:

**Data Retrieval:** Utilized HTTP libraries in Python, such as requests, to programmatically access and retrieve web pages from targeted real estate platforms.

**Content Parsing:** Employed parsing libraries like BeautifulSoup and lxml to extract relevant data fields from the HTML content of the web pages. These libraries were chosen for their robustness in handling various HTML structures and their flexibility in navigating complex webpage layouts.

**Data Normalization:** Since data were collected from multiple sources, a normalization process was established to standardize the data into a uniform format. This step was crucial for ensuring that subsequent data analysis was based on consistent and comparable metrics.

**Error Handling:** Integrated error handling mechanisms to manage issues such as connection timeouts, missing data, and unexpected site structure changes. This involved implementing retry logics, exception catching, and fallback routines to maintain the integrity and continuity of the data collection process.

## Workflow

The operational workflow of the web scraping pipeline consisted of several sequential stages:

**Target Identification:** Defined the specific pages and data fields of interest on each platform. This step involved preliminary manual review to understand the structure and navigation of each site.

**Automated Navigation:** Developed scripts that could automatically navigate through pagination and different categories or regions as required, to systematically cover the entire scope of the market in Athens.

**Data Extraction:** Extracted data such as listing ID, price, location, property size, number of bedrooms, images, and other descriptive information. Each piece of data was validated and cleaned before being stored to ensure accuracy.

**Storage:** Integrated the extracted data into a centralized MSSQL database. Special attention was given to designing a database schema that could efficiently manage and query large datasets.

**Monitoring and Maintenance:** Set up monitoring tools to track the performance of the scraping process and alert for failures or significant deviations in data quality or quantity.

## Challenges and Solutions

Several challenges were encountered during the development and operation of the web scraping pipeline:

**Dynamic Content:** Many real estate websites use JavaScript to dynamically load content, which required the use of Selenium or similar tools to render the pages fully before parsing.

**Rate Limiting:** To address potential rate limiting and avoid IP bans, the pipeline used rotating proxy servers and implemented polite scraping practices, such as respecting robots.txt rules and introducing delays between requests.

In summary, the web scraping pipeline developed for this thesis is a sophisticated tool tailored to meet the specific needs of comprehensive real estate data collection in Greece. Its design reflects a balance between efficiency, robustness, and ethical data collection practices, setting a strong foundation for the advanced data analysis conducted in subsequent chapters of this research.

## Overview of the Web Scraping Pipeline

The web scraping pipeline developed for this thesis involved a uniform method for extracting data from various major Greek real estate platforms using Python's requests library and BeautifulSoup for parsing HTML. This section provides a detailed explanation of the code used to scrape each platform, illustrating the uniformity and efficiency of the approach.

The web scraping pipeline developed for this thesis is a critical element of the research, designed to systematically collect data from various Greek real estate platforms. This section provides a detailed explanation of the conceptual and logical framework underpinning the scraping process. It emphasizes the uniform approach used across multiple platforms, ensuring consistency and efficiency in data collection.

## Design Principles and Objectives

The pipeline was constructed based on several key design principles:

- **Uniformity:** The approach was standardized across all platforms to ensure consistency in the data collected. This uniformity simplifies the integration and analysis of data from multiple sources.
- **Scalability:** The architecture supports scaling both in terms of the number of pages scraped and the addition of new sources without significant modifications to the core logic.
- **Robustness:** The pipeline is robust to interruptions and anomalies in data format, with error handling mechanisms that allow it to recover from common issues like network disruptions or changes in website structure.
- **Compliance and Ethics:** The scraping activities are compliant with legal standards and ethical guidelines concerning data privacy and the terms of service of the platforms.

## Technical Workflow

The technical workflow of the scraping process involves several stages:

- **Request Management:** The pipeline uses HTTP requests to retrieve web pages. This stage involves managing request headers and handling proxies to mimic genuine user behavior and avoid detection by anti-scraping technologies.
- **Data Parsing:** Once the HTML content is retrieved, BeautifulSoup is employed to parse the HTML and extract necessary data. The parsing rules are tailored to each platform's HTML structure but are designed to be flexible to accommodate minor changes in web page layouts.
- **Data Extraction:** Specific data points such as price, location, and property details are extracted. This step involves careful identification of HTML elements and attributes that reliably contain the required information.
- **Error Handling:** The pipeline includes sophisticated error handling to manage issues like connection timeouts or HTML parsing errors. This ensures the continuity and reliability of data collection.

- **Data Storage and Normalization:** Extracted data are temporarily stored in structured formats (like dictionaries or lists) and eventually transferred to a SQL database. This stage also involves the normalization of data to ensure uniformity across different datasets, facilitating easier integration and analysis.

### Implementation Details

- **Looping Through Pages:** The pipeline automates the process of navigating through pagination by dynamically adjusting the URL to access successive pages. This is crucial for comprehensive data collection, especially for platforms with a large number of listings.
- **Session Management:** To manage and maintain sessions across requests, especially when dealing with login-required areas or session-based tokens, the pipeline uses persistent HTTP sessions.
- **Data Integrity Checks:** Regular checks are implemented to ensure the integrity and accuracy of the data collected. This includes validations against predefined schemas or formats to detect anomalies early in the process.
- **Rate Limiting and Politeness:** The pipeline respects the rate limiting norms and the robots.txt file of each website to avoid overloading the servers. This not only ensures ethical scraping practices but also minimizes the risk of IP bans.

### Challenges and Solutions

- **Dynamic Content Handling:** Many real estate websites use JavaScript for dynamic content rendering. The pipeline addresses this by simulating a browser environment or using techniques to execute JavaScript when necessary.
- **Data Quality Assurance:** Given the diverse formats and incomplete data issues typical with real estate listings, the pipeline includes sophisticated cleansing and transformation routines to standardize and clean the data before it enters the central database.



## Explanation of Data Structure for Each Web Platform

For this thesis, separate database tables were created for each web platform to maintain the integrity and specificity of data sourced from different online real estate portals. Once the data was collected, a unified table was constructed to consolidate the similar fields from each platform, facilitating cross-platform analysis and comparison. Here's a detailed explanation of the approach for each platform, focusing on "spitogatos" as an example, and the methodology used to unify the data into a single table.

### Spitogatos Structure

The table for spitogatos contains a comprehensive set of fields that reflect both the general and specific aspects of real estate listings. Here's a breakdown of the key fields:

- **Id:** Unique identifier for each listing.
- **Link:** URL to the specific listing on the Spitogatos website.
- **Title:** The title of the listing, usually summarizing key features.
  - **Address:** Specific location of the property.
  - **Images:** Links to images associated with the listing.
  - **Agency:** Real estate agency or agent managing the listing.
  - **Description:** Detailed description of the property.
  - **Price:** Listing price of the property.
  - **Price per Sqm:** Price per square meter, providing a standardized measure of value.
  - **Area:** Total area of the property in square meters.
  - **Levels:** Number of levels or floors within the property.
  - **Floor:** Specific floor number for apartment listings.
  - **Kitchens:** Number of kitchens.
  - **Bathrooms:** Number of bathrooms.
  - **WC:** Number of water closets separate from full bathrooms.
  - **Living Rooms:** Number of living rooms.
  - **Heating:** Type of heating system installed.
  - **Energy Class:** Energy efficiency rating of the property.
  - **Construction Year:** Year the property was originally constructed.
  - **Renovation Year:** Most recent year the property was renovated.
  - **Sea Distance:** Distance to the nearest sea or significant body of water.
  - **System Code:** A system-generated code for internal tracking.

- **Code:** Another identifier, possibly used for differentiating listings within the platform.
- **Available From:** Date from which the property is available.
- **Published:** Date the listing was first published on the platform.
- **Last Updated:** Most recent date the listing information was updated.
- **Indoor Features:** Characteristics and amenities inside the property.
- **Outdoor Features:** Characteristics and amenities outside the property.
- **Construction Features:** Details about the construction quality and materials.
- **Good for:** Intended or suitable use of the property.
- **Bedrooms:** Number of bedrooms.

### Data Unification Approach

Given that similar fields exist across tables for different platforms (e.g., plot.gr, spiti24, xe.gr, etc.), a unified table was constructed to consolidate these listings. Here's the rationale and process for this consolidation:

- **Field Mapping:** Identified common fields across all platform tables, such as price, area, bedrooms, and bathrooms, ensuring that each field is standardized across the unified table.
- **Data Transformation:** Where necessary, transformed data to match a uniform format. For instance, prices were normalized to Euros, and area measurements to square meters.
- **Redundancy Elimination:** Removed duplicate entries where listings appeared on multiple platforms, using fields like address, price, and area as key indicators of duplicity.
- **Data Integration:** Merged data into a single table, employing SQL joins and union operations to compile the data effectively.
- **Quality Assurance:** Implemented data integrity checks to validate the accuracy and consistency of the merged data.

The data unification process for this thesis involved consolidating listings from multiple real estate platforms, such as plot.gr, spiti24, and xe.gr, into a single comprehensive table. This consolidation was achieved through a meticulous process that ensured data consistency and usability across various analyses.

The first step in this process was field mapping, where common fields across all platform tables were identified. Essential fields like price, area, bedrooms, and bathrooms were standardized to maintain uniformity within the unified table. This standardization was crucial for comparative and aggregate analysis later in the research.

Next, data transformation was applied where necessary to match a uniform format across all data entries. For example, all prices were normalized to Euros, regardless of the original currency presented on the respective

platforms, and area measurements were converted to square meters. This normalization was critical in ensuring that the data could be meaningfully compared and analyzed without discrepancies arising from different measurement units.

To ensure the accuracy and relevance of the data, redundancy elimination was conducted. This involved removing duplicate entries where listings appeared on multiple platforms. Key indicators such as address, price, and area were used to identify duplicates. This step was vital in maintaining a clean and reliable dataset for analysis, preventing skewed results from multiple counts of the same property.

The data integration phase involved merging the cleaned and transformed data into a single table using SQL joins and union operations. This phase was carefully managed to ensure that data from different sources was effectively compiled into one central repository, facilitating easier access and manipulation in subsequent analytical processes.

Finally, quality assurance measures were implemented to validate the accuracy and consistency of the merged data. This included data integrity checks that ensured all data conformed to the expected formats and values. Ensuring data integrity was crucial for the credibility of the research findings.

For saving the unified data into the database, an Object-Relational Mapping (ORM) approach was employed. ORM provided an efficient and error-reducing means of translating the data into a database schema. This method was particularly useful in handling complex queries and data interactions, which are common in handling large datasets of real estate listings.

Overall, the data unification approach not only streamlined the research process by creating a centralized data resource but also enhanced the reliability and accuracy of the analyses conducted in the thesis.

### **Database Management and Data Integration**

The design of the database is pivotal in supporting the data requirements of this thesis. A relational database model was chosen due to its ability to handle large volumes of structured data and support complex queries efficiently.

**Schema Design:** The database schema was carefully designed to accommodate data from various sources while ensuring that it could be easily extended and modified. The schema includes tables for each real estate platform (e.g., plot.gr, spiti24, xe.gr) and a unified table that consolidates all the data. Each table is structured with fields that reflect the data model of the respective platform, such as price, area, bedrooms, bathrooms, and unique identifiers.

**Normalization:** To reduce redundancy and improve data integrity, normalization practices were applied. This involved organizing data into tables and columns in a way that dependencies are properly enforced by database integrity constraints, thereby reducing redundancy and improving data integrity.

### **Data Storage Strategy**

The data storage strategy encompasses the methods and technologies used to store and manage the collected data effectively.

**Relational Database Management System (RDBMS):** An RDBMS was used to manage the relational tables and to facilitate complex SQL queries. This system supports transactions, which ensures data integrity and consistency even in the case of errors or failures.

**Data Type Standardization:** Ensuring that each field in the database adheres to a standardized data type is crucial for maintaining consistency. For example, all monetary values are stored in a consistent format and datatype, which simplifies calculations and comparisons.

### **Object-Relational Mapping (ORM)**

Object-Relational Mapping (ORM) is a technique that helps in converting data between incompatible type systems in object-oriented programming languages and relational databases. This approach provides a high-level abstraction upon the relational database that allows for maintaining a robust database schema.

**Benefits of ORM:** ORM frameworks facilitate the management of database entities and relationships directly from the Python code, which enhances productivity and reduces the likelihood of SQL injection attacks. ORM also automates the transfer of data from in-memory objects to database tables, ensuring that the data adheres to the schema without requiring manual parsing of data.

**Implementation:** In this thesis, an ORM framework compatible with Python was employed to interact with the SQL database. This framework allowed for defining each data model class in Python, which then translates into a database table through migrations. CRUD (Create, Read, Update, Delete) operations were implemented using ORM methods, which abstracted the complex SQL queries into simple and secure Python methods.

### **Data Integration Process**

Integrating data from multiple sources into a unified database involved several steps:

**Data Extraction and Transformation:** Before integration, data was extracted from various formats and transformed into a unified format. This involved cleaning, normalizing, and transforming data to fit the database schema.

**Data Loading:** The transformed data was loaded into the database using batch operations, which improves performance and minimizes the load on the database server.

**Data Validation and Cleansing:** Post-integration, data was validated against the schema constraints to catch any anomalies and was cleansed to ensure accuracy and completeness.

### **Security and Backup**

**Security Measures:** Security measures such as encrypted connections and secure access controls were implemented to protect the data.

**Backup Procedures:** Regular backups and data recovery procedures were established to ensure data durability and availability.

## Implementation of the Django Full Stack Application

In this chapter, the focus shifts to the implementation of a Django full stack application that integrates the image similarity algorithm with a user-friendly interface. This application serves as the platform through which users interact with the data, leveraging Django's robust back-end capabilities and a well-designed front-end to present data in an insightful and accessible manner. The chapter outlines the strategic development phases, including the setup of the Django environment, integration of the image similarity algorithm, data fetching mechanisms, and the user interface design.

### Django Application Setup

Setting up the Django environment involves configuring the framework to work seamlessly with the existing database and ensuring that all components are optimized for performance and scalability.

**Framework Configuration:** Django is set up with settings that align with the project's needs, including database configurations, static and media files settings, security settings, and middleware configurations for handling requests.

**Model Definition:** Models are defined in Django to mirror the database schema created in the previous chapter. This ORM feature of Django facilitates the interaction with the database in an object-oriented manner.

**Admin Interface Setup:** Django's built-in admin interface is customized to allow administrators to easily manage the content of the application, including real estate listings, images, and user data.

### Backend Implementation in Django

The backend of the application, built using Django, handles data management, algorithm integration, and server-side logic:

**Data Management:** Django models are used to manage data related to properties, user preferences, and image metadata. Django's ORM capabilities facilitate interaction with the database where property information and image data are stored.

## Image similarity model training

This model is designed to find matching folders based on the similarity of images between a temporary folder (temp\_folder) and a data folder (data\_folder). Here's a step-by-step breakdown of how it works:

In the rapidly advancing field of computer vision, the ability to automatically identify and match similar images across different datasets is of paramount importance. This capability finds applications in various domains, including digital forensics, content management systems, and automated archival processes. The focus of this thesis is on developing an efficient and accurate method to find matching folders containing similar images within a large dataset, utilizing the ORB (Oriented FAST and Rotated BRIEF) feature detector and descriptor.

Traditional methods of image comparison often rely on color histograms or pixel-by-pixel analysis, which can be computationally intensive and less effective when dealing with images that have undergone transformations such as rotation, scaling, or partial occlusion. The ORB algorithm, introduced by Rublee et al. in 2011, presents a robust alternative by focusing on keypoints and their descriptors, allowing for effective matching even under varying image conditions.

This thesis proposes an algorithm that leverages ORB to compare images from a temporary folder against a set of images distributed across multiple subfolders in a data directory. The primary objective is to identify and return the names of subfolders that contain images similar to those in the temporary folder, based on a predefined similarity threshold. The algorithm is designed to be both efficient and scalable, capable of handling large datasets without significant computational overhead.

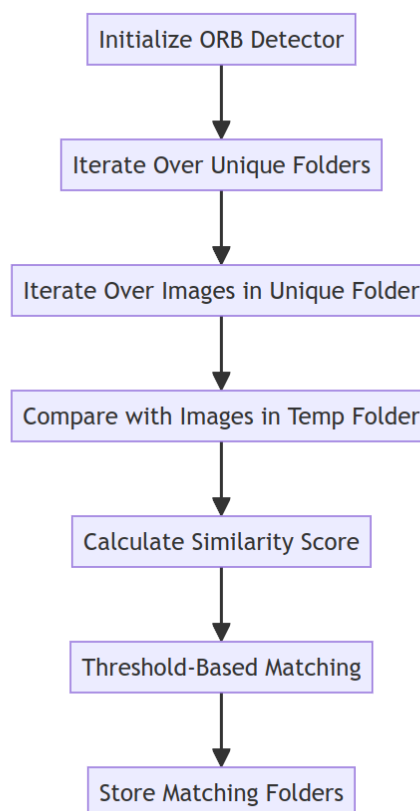


Figure 8: Image similarity training process

Convolutional Neural Networks (CNNs) have revolutionized the field of image processing by enabling high levels of accuracy in tasks such as image classification and object detection. CNNs automatically learn and capture the hierarchical patterns in images which makes them particularly suited for tasks requiring a high degree of visual recognition.

### **ResNet50 Architecture**

ResNet50, a variant of the ResNet family, is a deep convolutional network known for its architecture that includes 50 layers. It introduces a novel architecture with "skip connections" which helps in alleviating the vanishing gradient problem by allowing direct gradient flow between layers. The ResNet50 model utilized in this project is pre-trained on the ImageNet dataset, which consists of over a million images categorized into 1000 classes. This pre-training provides a solid foundation for extracting generic features that can be applicable to a wide range of visual recognition tasks.

### **Image Preprocessing**

For the model to process images effectively, they must be normalized and resized to meet the input specifications of the ResNet50 model:

- **Normalization:** Adjusts pixel values to a scale that the model expects, based on its training on the ImageNet dataset.
- **Resizing:** Images are resized to 224x224 pixels, which is the required input size for the model.

### **Feature Extraction**

The script employs the ResNet50 model to extract features from images. Instead of using the model to predict image classes directly, we modify it to output a vector from the penultimate layer. This vector, typically 2048 elements long, represents a dense embedding of the image's visual content:

**Pooling:** The 'average' pooling layer is used to reduce the dimensions of the output from the convolutional layers, summarizing the detected features into a single vector.

### **File and Data Management**

Efficient data handling is crucial for managing large image datasets:

**Directory Listing and Set Operations:** The script lists all image directories and identifies discrepancies between existing dataset indices and those already processed, ensuring no image is processed twice.

**Checkpointing:** To manage long-running operations and potential interruptions, the script uses a checkpointing mechanism. It records the last processed image, allowing the process to resume without loss of progress.

The practical implementation of the above methodology involves iterating over a dataset of images, processing each image to extract features, and storing these features in an array format for quick access and analysis. The script handles errors gracefully, retries file access when necessary, and logs its progress for monitoring and debugging purposes.

The extraction of image features using a pre-trained ResNet50 model provides a robust foundation for building image similarity models. The features captured by this model encapsulate complex patterns in the visual data, which are essential for the accurate comparison and retrieval of images based on similarity. This approach not

only enhances the effectiveness of similarity-based tasks but also significantly reduces the computational overhead involved in training a model from scratch.

### Error Handling and Retry Logic

To ensure robustness, especially when accessing filesystems or during long-running processes, the script includes error handling and retry mechanisms:

- **Directory Access Failures:** If a directory fails to open, the script retries several times before logging an error and skipping it.
- **Image Processing Failures:** Errors during image loading or feature extraction are logged, and the problematic image is skipped, preventing the entire process from failing.

```
1/1 ██████████ 1s 933ms/step
INFO:root:Processed 94308791 / image_18.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 79ms/step
INFO:root:Processed 94308791 / image_23.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 81ms/step
INFO:root:Processed 94308791 / image_17.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 73ms/step
INFO:root:Processed 94308791 / image_22.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 68ms/step
INFO:root:Processed 94308791 / image_12.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 71ms/step
INFO:root:Processed 94308791 / image_24.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 98ms/step
INFO:root:Processed 94308791 / image_8.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 63ms/step
INFO:root:Processed 94308791 / image_6.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 66ms/step
INFO:root:Processed 94308791 / image_9.jpg
INFO:root:Feature vector shape: (2048,)
1/1 ██████████ 0s 75ms/step
INFO:root:Processed 94308791 / image_13.jpg
INFO:root:Feature vector shape: (2048,)
```

Figure 9: Training steps



## Integration with Django

The integration of the image similarity algorithm within the Django application involves several key components:

**Backend Processing:** Django manages the backend processes, including the retrieval of image data from the database, handling the file system operations for image storage, and executing the feature extraction and similarity measurement tasks.

**Asynchronous Tasks:** Given the potentially high computational load of processing large volumes of images, these operations are handled asynchronously using Django's support for background tasks (e.g., using Celery with RabbitMQ or Redis as the message broker).

**APIs for Frontend Interaction:** Django serves the results of the image similarity analysis through RESTful APIs, which the frontend consumes to display similar property recommendations to users.

## Integration of Image Similarity Algorithm

The image similarity algorithm is a core component of the application, enabling the platform to identify and suggest similar real estate listings based on image content.

- **Algorithm Embedding:** The algorithm is embedded into the Django application as a backend service. This involves setting up a dedicated route in the application that handles image processing and similarity computation.
- **Data Handling:** When images are uploaded or updated in the listings, the algorithm processes these images to extract features and compute similarities, storing these results in the database for quick retrieval during user queries.
- **Performance Optimization:** The implementation is optimized for performance, ensuring that the image processing tasks do not hinder the overall user experience of the application.

## Implementation of the Django Full Stack Application with Image Similarity-Based Recommendation System

This chapter delves into the practical implementation of a Django full-stack application that incorporates an image similarity-based recommendation system for real estate listings. The application leverages sophisticated image processing algorithms to recommend similar properties to users, enhancing their experience by tailoring options based on visual preferences. It also features robust filtering capabilities, allowing users to refine their search according to specific criteria. Below is a comprehensive discussion of the application's interface, the underlying technology of the recommendation system, and its integration within the Django framework.

### User Interface Design

The application's user interface (UI) is designed to be intuitive and user-friendly, facilitating easy navigation and interaction. The main components of the UI include:

**Listing Display:** As shown in the uploaded screenshot, the main part of the interface displays a series of property images that represent individual listings. Each listing provides a visual snapshot of the property, allowing users to quickly gauge the appearance and style of each property.

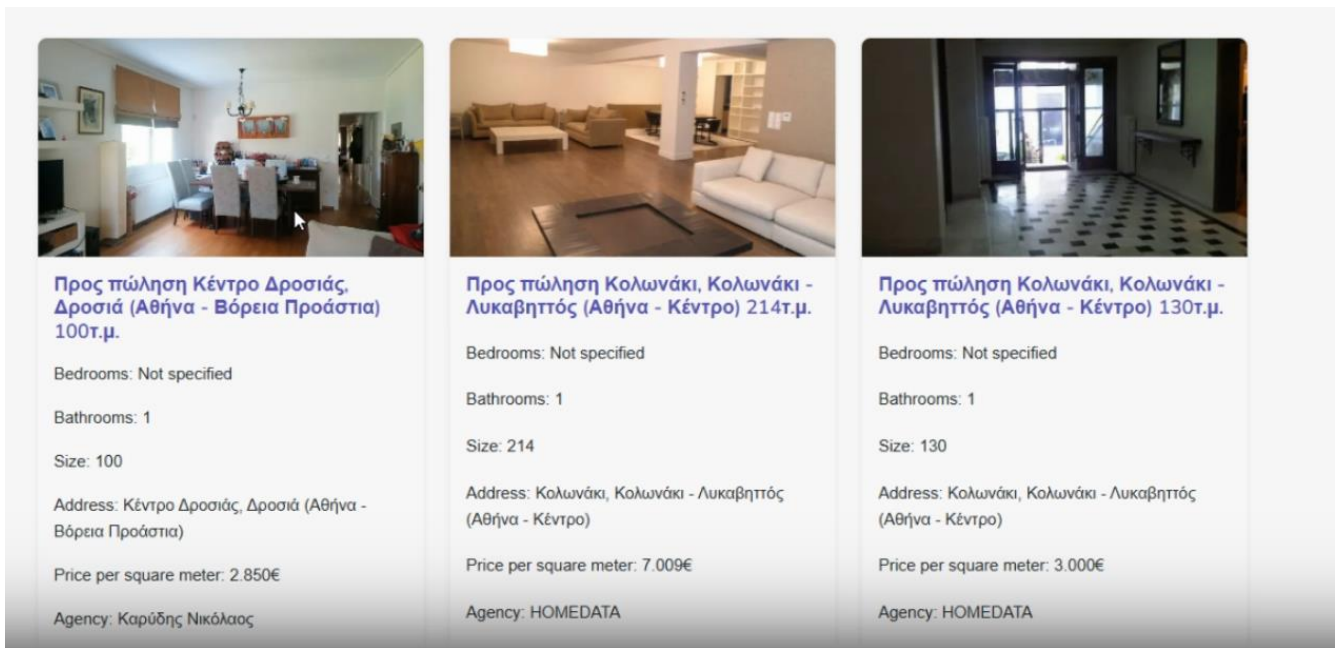


Figure 10: Django app

**Dynamic Filters:** The interface includes filters that users can apply to refine their search based on various parameters such as location, price, property size, and specific features like the number of bedrooms or bathrooms. This allows users to tailor the recommendations to better fit their needs and preferences.

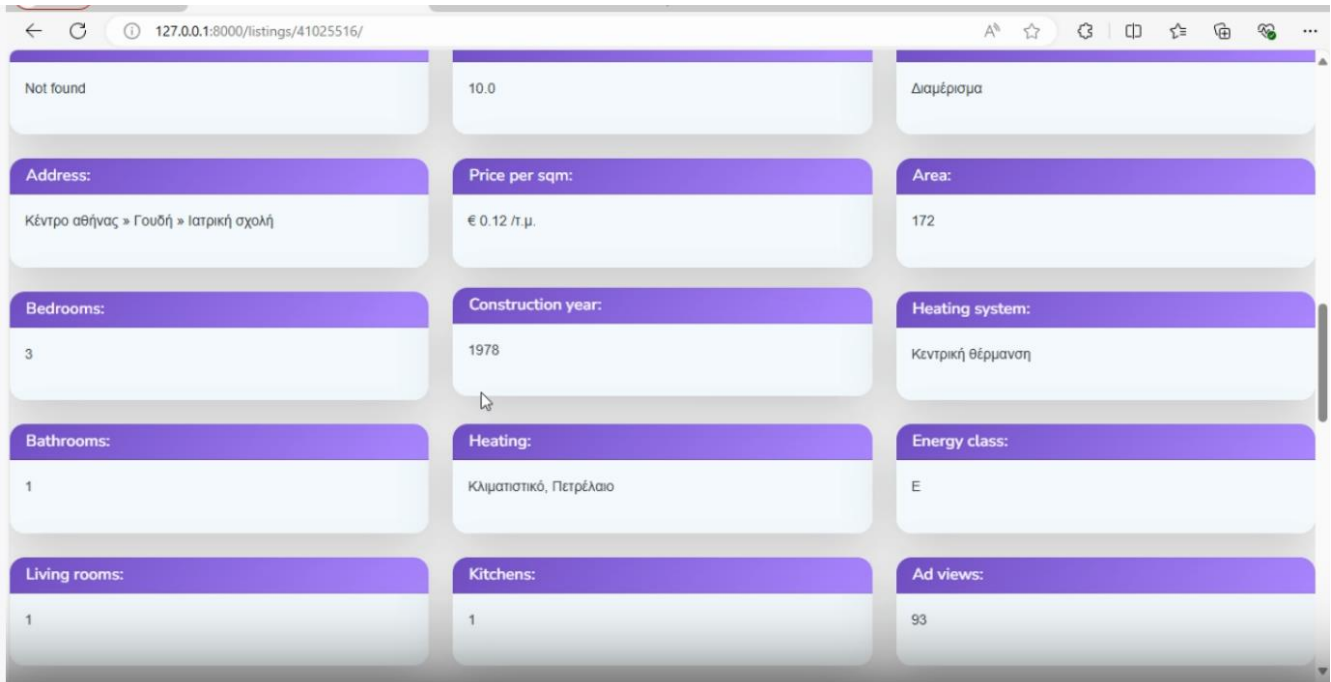


Figure 11: Properties details

## Image Similarity Algorithm Integration

The image similarity algorithm is a core feature of the application, enabling the recommendation system by analyzing visual content of property images to identify similar properties across different platforms. Here's how it is integrated:

**Algorithm Setup:** The algorithm processes images using convolutional neural networks (CNNs) to extract feature vectors from each property image stored in the database. These vectors represent key visual attributes of the images.

**Similarity Calculation:** When a user views a property, the algorithm compares its feature vector with those of other properties in the database using cosine similarity or another suitable metric. It then identifies properties with the highest similarity scores. We have an output of the similar ids in the terminal and the app interface template (figure 12 & 13).

```
September 01, 2024 - 17:47:20
Django version 5.0.8, using settings 'real_estate_app.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

1/1 ██████████ 1s 930ms/step
1/1 ██████████ 0s 54ms/step
1/1 ██████████ 0s 50ms/step
Matching Folder IDs: ['98853958', '98853957', '98853955', '98853953', '98853954']
[01/Sep/2024 17:48:02] "GET /listings/98853955/ HTTP/1.1" 200 28532
```

Figure 12: Matching property ids found for specific property

**Recommendation Generation:** Properties with the highest similarity scores are presented to the user in the "Recommended For You" section, providing personalized suggestions that align with the user's apparent visual preferences.

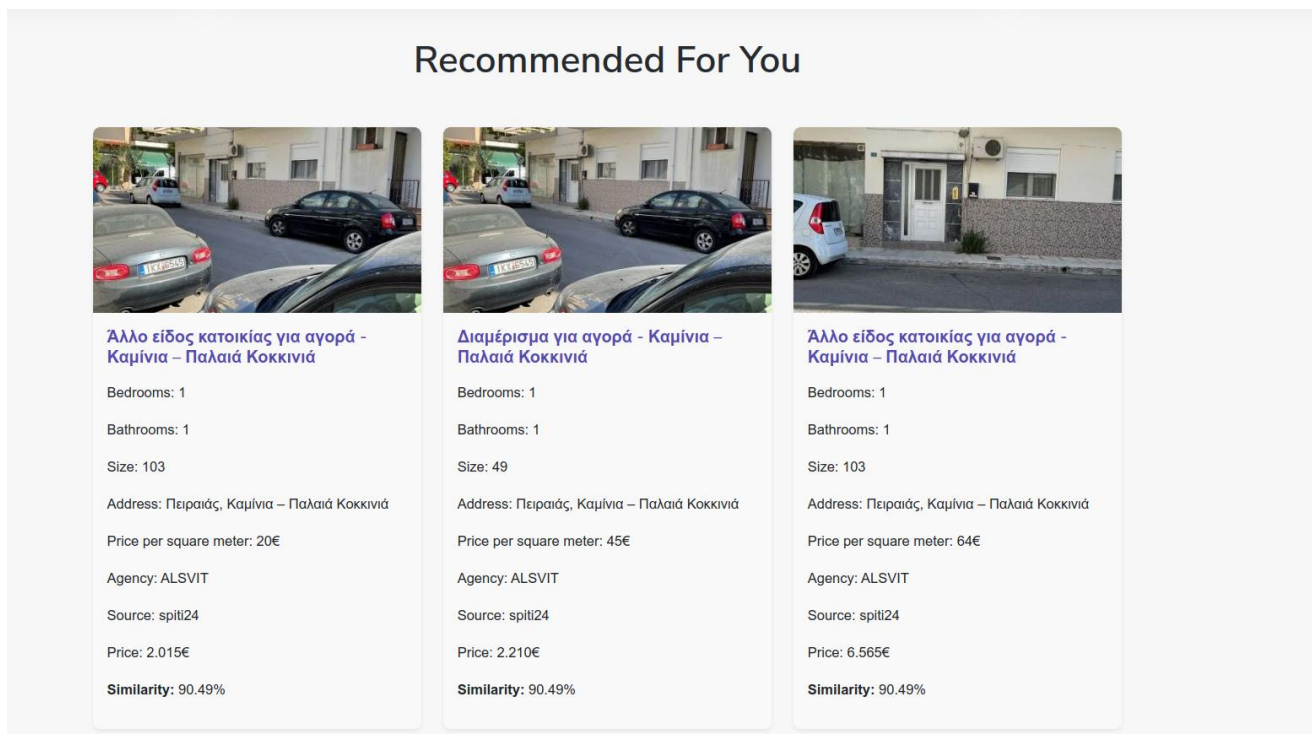


Figure 13: Recommendation system

**Recommendation Section:** Below the main listing images, a "Recommended for You" section (figure 12) suggests similar properties based on image similarity. This feature uses the image similarity algorithm to find and display listings that share visual characteristics with properties the user has shown interest in.

### Workflow for Image Similarity Detection

**Image Processing:** Each image uploaded to the platform is processed using a predefined image processing algorithm. This process extracts feature vectors that numerically represent the visual content of each image.

**Similarity Computation:** The application calculates similarity scores between the feature vector of a reference image and the vectors of other images in the database. This is likely done using a method such as cosine similarity, which measures the cosine of the angle between two vectors.

**Threshold Application:** A threshold is set (in your case, it might be 0.7 as suggested by repeated scores in the output), which serves as the cut-off point for determining which images are considered similar. Scores that meet or exceed this threshold indicate a high degree of similarity, suggesting that the properties in the images share visual characteristics.

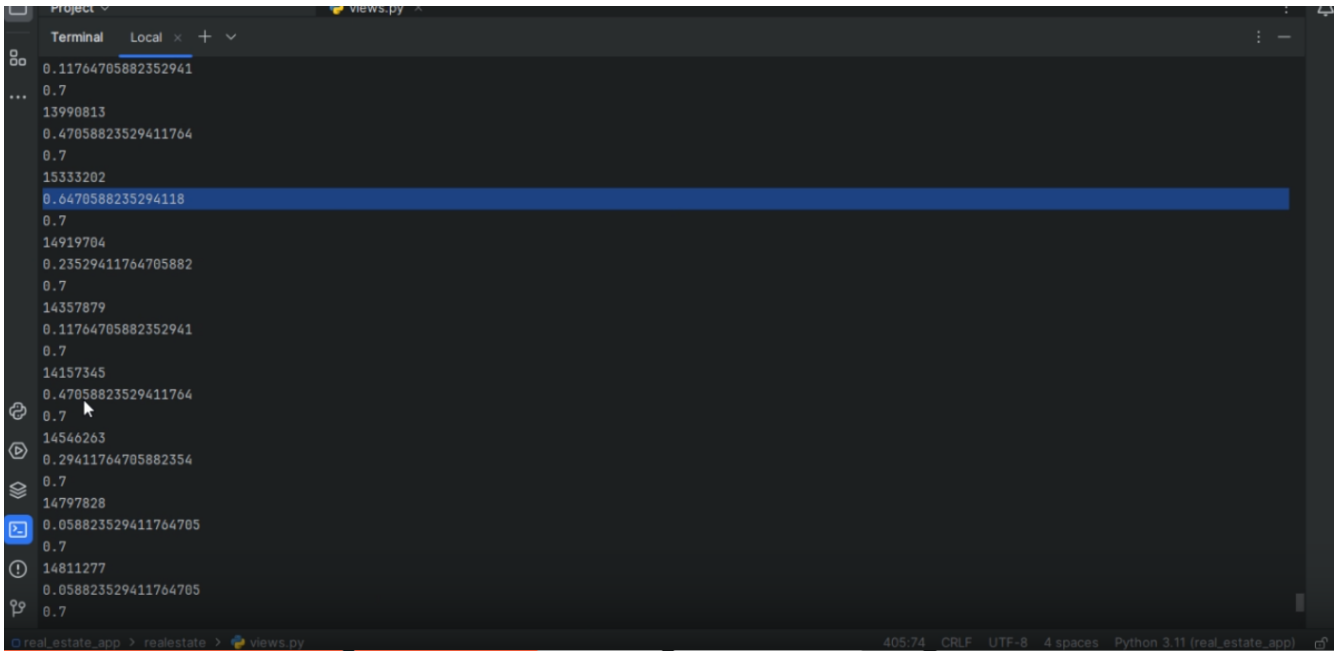
**Displaying Recommendations:** Listings corresponding to images that meet the similarity criteria are flagged as similar and can be recommended to users. This functionality enhances the user experience by providing personalized recommendations based on visual preferences.

### Integration into the Django App

**Backend Integration:** The similarity scores are computed in the backend of your Django application. The process involves handling large amounts of image data and requires efficient data management and processing capabilities.

**Frontend Display:** The similar listings, based on the scores, are displayed to the user in the "Recommended For You" section of the UI. This seamless integration between backend computations and frontend display is crucial for maintaining a responsive user interface.

**User Interaction:** Users can interact with the recommendations by clicking on similar images to view more details about the listings.



```
0.11764705882352941
0.7
13990813
0.47058823529411764
0.7
15333202
0.6470588235294118
0.7
14919704
0.23529411764705882
0.7
14357879
0.11764705882352941
0.7
14157345
0.47058823529411764
0.7
14546263
0.29411764705882354
0.7
14797828
0.058823529411764705
0.7
14811277
0.058823529411764705
0.7
```

Figure 14: Similarity calculation

## Technical Explanation of the Image Similarity Algorithm

In this chapter, we delve into the technical aspects of the image similarity algorithm implemented in the Django full-stack application for real estate listings. This algorithm is central to the recommendation system, allowing the application to suggest listings that visually resemble each other. The explanation covers the algorithm's conceptual basis, the methodologies employed for feature extraction and similarity measurement, and its integration into the Django environment.

### Conceptual Framework

The image similarity algorithm operates on the principle that images can be quantitatively compared by converting them into a form where visual features are represented as high-dimensional vectors. These vectors capture essential aspects of the images, such as textures, shapes, and colors, allowing for numerical comparison.

**Feature Extraction:** This is the process of transforming raw images into a set of features (numerical data) that are useful for comparison. Feature extraction reduces the dimensionality of the data by capturing only the essential information needed for the task at hand.

**Similarity Measurement:** After feature extraction, the similarity between images is quantified using a suitable metric that compares feature vectors. This measure determines how 'close' two images are in the feature space.

Feature extraction in the context of the real estate application involves processing images retrieved from various listings. The process can be divided into two main steps: image retrieval and the transformation of these images into a format suitable for analysis.

### Image Retrieval:

Each image associated with a listing is uniquely identified by a combination of the listing ID and an image index. For instance, for a listing with ID '123', images might be named '123\_1', '123\_2', etc.

Images are fetched from URLs stored in the database corresponding to each listing. The structure ensures that each image is traceably linked to its source listing, allowing for systematic processing and retrieval.

To manage the images efficiently, they are stored locally in a structured directory system. Each listing ID has a dedicated folder, which contains all related images. This organization simplifies access to the images during processing and helps in maintaining a clean dataset by segregating images by their respective listings.

### Transformation and Feature Vector Creation:

**Preprocessing:** Images are first preprocessed to normalize aspects such as size and color intensity. This normalization ensures that the feature extraction process is uniform across all images.

**Feature Detection:** Using convolutional neural networks (CNNs) or other suitable image processing algorithms, key features of the images are extracted. These features might include edges, corners, textures, and other relevant visual attributes that define the content of the images.

**Vectorization:** The extracted features are then converted into a vector format. Each vector represents an image in a multidimensional feature space where each dimension corresponds to a feature detected in the image.

## Similarity Measurement Techniques

After converting images into feature vectors, the next step involves measuring the similarity between these vectors to identify images that are visually similar.

**Similarity Metrics:** Common metrics used for this purpose include cosine similarity, Euclidean distance, and Manhattan distance. In the context of this application, cosine similarity is particularly useful as it measures the cosine of the angle between two vectors, providing a metric that is relatively insensitive to the magnitude of the vectors and focuses solely on their orientation in the feature space.

**Thresholding:** A threshold value is set to determine when two images are considered similar. This threshold is empirically determined based on testing and validation during the development phase. Images with similarity scores above this threshold are tagged as similar.

## Results

### Dataset Description

The dataset used in this study was compiled from four major Greek real estate platforms: Spitogatos, Spiti24, Plot, and Tospitimu. The primary goal was to collect a diverse and comprehensive set of real estate listings, including various property types such as apartments, houses, and commercial properties. This dataset specifically focuses on properties for sale in both the Athens city center and its surrounding suburbs.

To analyze the dataset effectively, we extracted unique listing IDs from each platform. This analysis helps in understanding the distribution of listings and the coverage provided by each platform. The following visual represents the number of unique IDs collected from each source.

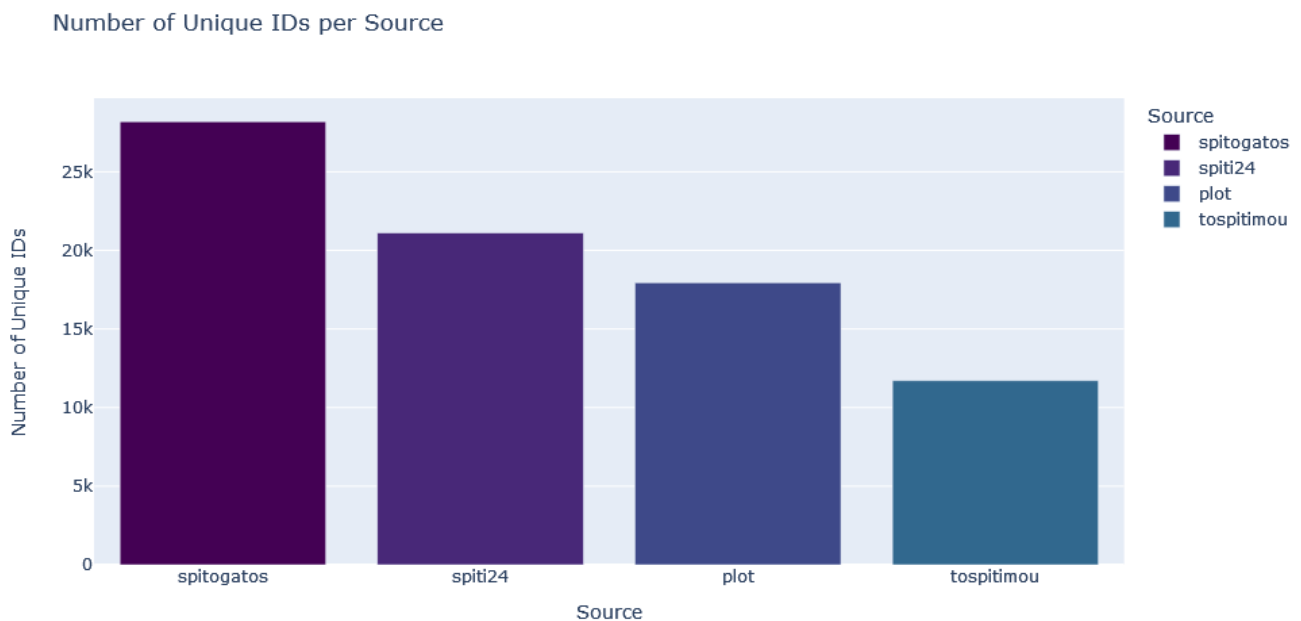


Figure 15: Number of Unique IDs per Source



The dataset used in this study was compiled from four major Greek real estate platforms: Spitogatos, Spiti24, Plot, and Tospitimou. The primary goal was to collect a diverse and comprehensive set of real estate listings, which includes varying property types such as apartments, houses, and commercial properties.

To analyze the dataset effectively, we extracted unique listing IDs from each platform. This analysis helps in understanding the distribution of listings and the coverage provided by each platform. The following visual represents the number of unique IDs collected from each source.

The bar chart in Figure 8 illustrates the distribution of unique real estate listing IDs across four major platforms. Each bar represents the count of unique IDs sourced from a specific platform. Here's a detailed look at the data represented:

- **Spitogatos:** The platform contributed the highest number of unique IDs, totaling approximately 27,000 listings. This indicates Spitogatos' significant presence and extensive coverage in the Greek real estate market.
- **Spiti24:** With around 21,000 unique IDs, Spiti24 is the second-largest contributor, showcasing a substantial number of listings available on this platform.
- **Plot:** This platform provided about 18,000 unique IDs, marking it as a key player, although slightly behind Spiti24 in terms of listing count.
- **Tospitimou:** The least number of unique IDs were sourced from Tospitimou, with roughly 14,000 listings. Despite having the lowest count among the four platforms, it still contributes significantly to the overall dataset.

The data indicates that Spitogatos has the largest dataset, which might reflect its popularity and possibly a more extensive listing database compared to the other platforms. The significant number of unique IDs from all four sources highlights the necessity of consolidating duplicate listings to provide accurate market insights. This distribution is crucial for understanding the dataset's scope and the subsequent steps in duplicate detection and consolidation.

### **Implications:**

1. **Market Coverage:** Spitogatos and Spiti24 have higher market coverage, which suggests that users might prefer these platforms for listing properties. This could be due to better platform features, broader audience reach, or more comprehensive listing services.
2. **Data Redundancy:** The presence of multiple platforms with substantial listings raises the likelihood of data redundancy, where the same property might appear across different platforms. This necessitates an effective algorithm to detect and consolidate these duplicates.
3. **Focus for Improvement:** Platforms like Tospitimou, with fewer listings, might need to enhance their market presence or improve their data acquisition strategies to compete with more dominant platforms.

By understanding the dataset's distribution, we can better appreciate the challenges in managing and consolidating real estate data across multiple platforms. The next sections will delve into the feature extraction, image processing, and the results of the duplicate detection algorithm, providing a comprehensive overview of the study's findings.

### **Top 10 Regions by Number of Listings**

The analysis also involved examining the geographical distribution of the real estate listings. This helps in understanding the concentration of listings in various regions and can provide insights into the most active real estate markets.

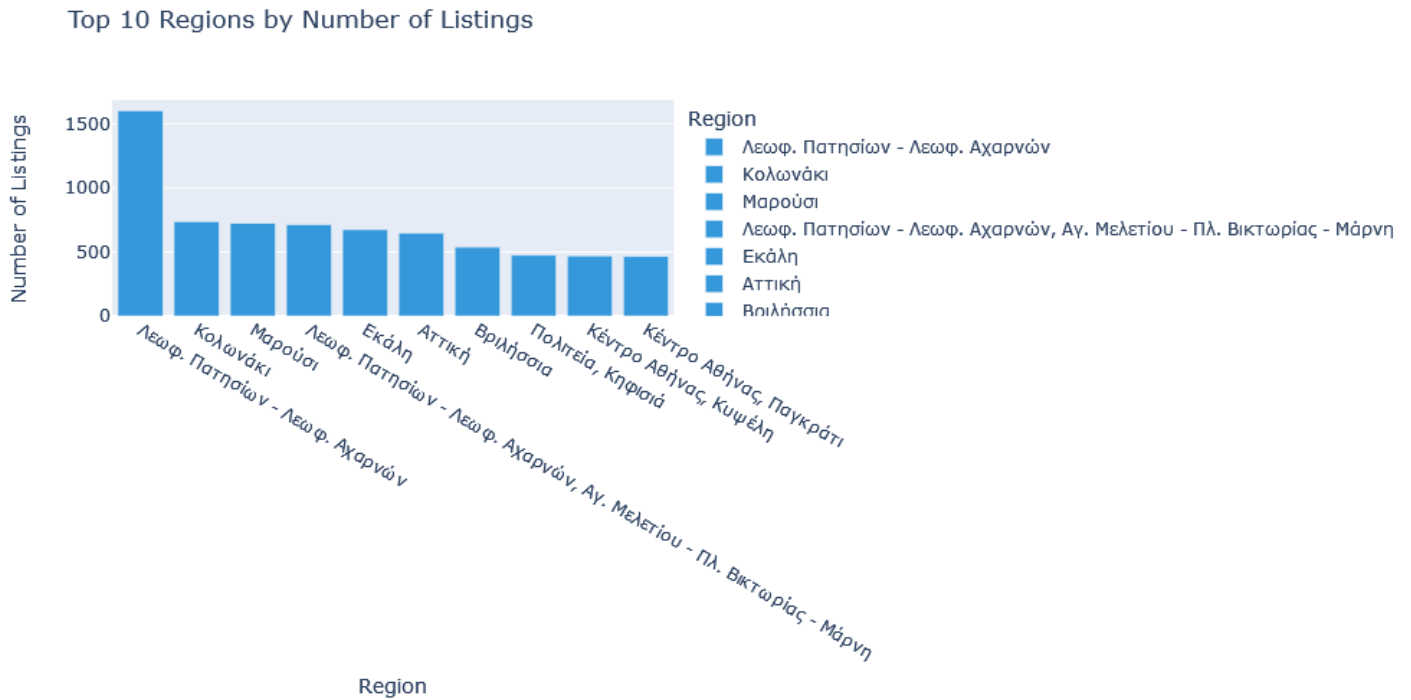


Figure 16: Top 10 regions by number of listings

The bar chart in Figure 9 highlights the top 10 regions with the highest number of real estate listings. The x-axis represents the regions, while the y-axis shows the number of listings. Each bar corresponds to the count of listings in a specific region, with the following key observations:

- **Λεωφ. Πατησίων - Λεωφ. Αχαρνών:** This region leads with the highest number of listings, totaling approximately 1,500. This indicates a highly active real estate market, possibly due to high demand and availability of properties.
- **Κολωνάκι:** The second most active region, with around 600 listings. Known for its prime location, this area likely attracts significant real estate activity.
- **Μαρούσι:** With approximately 600 listings, this region is a significant hub for real estate, indicating its popularity and growth.
- **Εκάλη:** This region also shows a considerable number of listings, reflecting its attractiveness in the real estate market.
- **Αττική:** The region has around 500 listings, showcasing its importance in the real estate sector.
- **Βριλήσσια:** Another active region with roughly 400 listings.
- **Πολίτεια - Κηφισιά:** This region has about 300 listings, indicating its relevance in the market.
- **Κέντρο Αθήνας Παγκράτι:** With approximately 250 listings, it remains an important area for real estate transactions.

- **Αγ. Μελετίου - Πλ. Βικτωρίας - Μάρνη:** This region shows around 200 listings, reflecting its activity in the market.
- **Βουλιαγμένη:** With roughly 200 listings, this region is also a notable area in the real estate landscape.

The data indicates that the Λεωφ. Πατησίων - Λεωφ. Αχαρνών region has the highest number of listings, making it a central area for real estate activities. Regions like Κολωνάκι and Μαρούσι follow, suggesting that these areas are also highly sought after for property transactions. This distribution can help real estate professionals target specific regions for marketing and sales efforts.

### Top 10 Regions by Average Price

In addition to the number of listings, analyzing the average price per region provides valuable insights into the real estate market's economic landscape. This analysis helps identify regions with higher property values, which can inform investment decisions and market strategies.

Top 20 Regions by Average Price

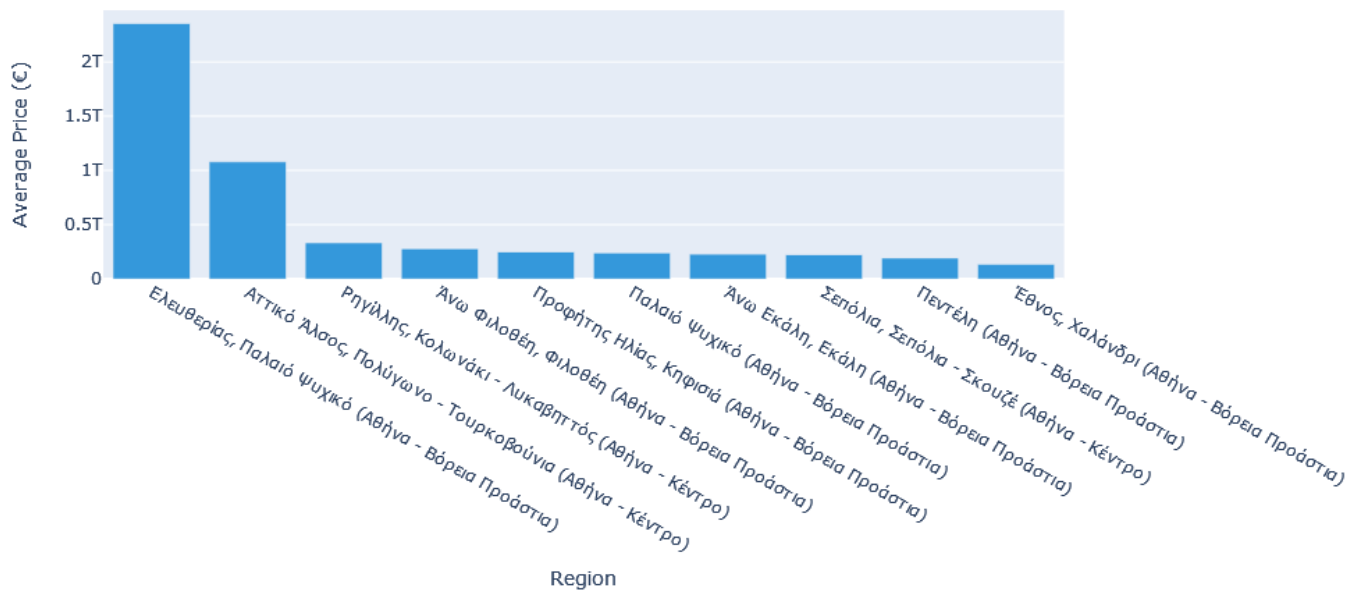


Figure 17: Top 10 regions based on the average price

The bar chart in Figure 10 presents the top 10 regions based on the average price of real estate listings. The x-axis represents the regions, while the y-axis shows the average price in euros (€). Each bar indicates the average price of listings in a specific region. The key observations are:

- **Ελευθεριάς, Παλαιό Ψυχικό (Αθήνα - Βόρεια Προάστια):** This region has the highest average price, exceeding 2 million euros, indicating its status as a highly desirable and premium real estate market.
- **Αττικό Άλσος Πολύγωνο:** The second-highest average price is around 1 million euros, suggesting a significant market value in this region.
- **Ριγκίλης Κολωνάκι - Λυκαβηττός (Αθήνα - Κέντρο):** This region shows a substantial average price of approximately 500,000 euros, reflecting its central location and high demand.
- **Προφήτης Ηλίας Κηφισιά (Αθήνα - Βόρεια Προάστια):** With an average price around 400,000 euros, this region also indicates a high property value.
- **Παλαιό Ψυχικό (Αθήνα - Βόρεια Προάστια):** Another high-value region with an average price of around 350,000 euros.
- **Άνω Εκάλη (Εκάλη):** This area has an average price of approximately 300,000 euros.
- **Σεπόλια (Αθήνα - Κέντρο):** This region shows an average price of around 250,000 euros, reflecting significant real estate activity.
- **Πεντέλη (Αθήνα - Βόρεια Προάστια):** With an average price of approximately 200,000 euros, indicating its relevance in the market.
- **Έξω Χαλάνδρι (Αθήνα - Βόρεια Προάστια):** Also showing an average price of around 200,000 euros.
- **Αγ. Φιλοθέη, Φιλοθέη (Αθήνα - Βόρεια Προάστια):** This region shows an average price of approximately 150,000 euros.

The data indicates that Ελευθεριάς, Παλαιό Ψυχικό (Αθήνα - Βόρεια Προάστια) is the most expensive region, significantly higher than the other regions analyzed. This could be attributed to its prime location, desirable neighborhood, and high-end amenities. The regions with high average prices such as Αττικό Άλσος Πολύγωνο and Ριγκίλης Κολωνάκι - Λυκαβηττός (Αθήνα - Κέντρο) are also centrally located and highly sought after, contributing to their elevated property values.

### **Average Price per Square Meter per Region**

Understanding the average price per square meter is crucial for assessing the real estate market's value and comparing different regions. This metric provides insight into the cost of property relative to its size, which is a key factor for buyers and investors.

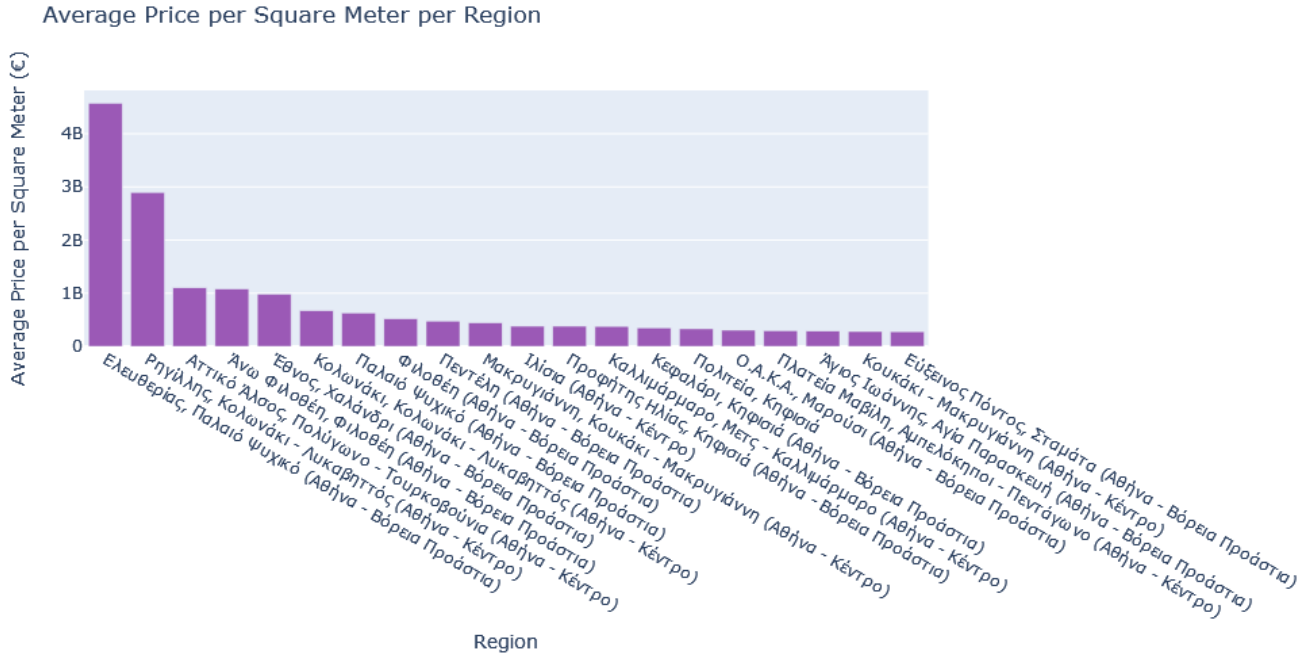


Figure 18: Average price per square meter

The bar chart in Figure 11 shows the average price per square meter for properties in various regions. The x-axis represents the regions, while the y-axis shows the average price per square meter in euros (€). Each bar indicates the average cost of real estate per square meter in a specific region. Key observations from the chart include:

- **Ελευθεριάς, Παλαιό Ψυχικό (Αθήνα - Βόρεια Προάστια):** This region stands out with the highest average price per square meter, exceeding 4,000 euros. This indicates it is a premium area with high property values.
- **Αττικό Άλσος Πολύγωνο:** The second-highest average price per square meter is around 3,000 euros, suggesting a significant market value in this region.
- **Ριγκίλης Κολωνάκι - Λυκαβηττός (Αθήνα - Κέντρο):** This region also shows a high average price per square meter, around 2,500 euros, reflecting its central location and high demand.
- **Άνω Εκάλη:** With an average price per square meter around 1,500 euros, this area is another high-value region.
- **Πεντέλη (Αθήνα - Βόρεια Προάστια):** This region shows an average price per square meter of approximately 1,200 euros.
- **Other Regions:** The remaining regions have average prices per square meter ranging from around 1,000 to 500 euros, indicating more affordable property values compared to the top regions.

The data highlights significant variations in property values across different regions. Ελευθεριάς, Παλαιό Ψυχικό (Αθήνα - Βόρεια Προάστια) commands the highest prices per square meter, which could be due to factors such as location desirability, exclusivity, and available amenities. Similarly, regions like Αττικό Άλσος Πολύγωνο and Ριγκίλης Κολωνάκι - Λυκαβηττός (Αθήνα - Κέντρο) also reflect high property values, likely driven by their central locations and premium market segments.

### **Images dataset**

To facilitate the processing and analysis of real estate listings, a Python script was developed to extract images from provided URLs and organize them into directories. Each directory corresponds to a unique listing ID, ensuring a structured dataset.

The column labeled images contains URLs pointing to images of real estate listings. Each URL represents a direct link to an image file hosted on a server. These URLs are essential for downloading and processing the images for further analysis. The format of the images column is a comma-separated list of URLs. Each URL corresponds to a unique image of the property listing.

In addition to organizing the images into directories based on unique listing IDs, we conducted a quantitative analysis to summarize the image dataset. This analysis provides insights into the average number of images per listing and the total number of images collected.

#### **Average Number of Images per Listing:**

Through our data collection process, it was determined that, on average, each listing has approximately 13 images. This metric, calculated as 13.075747947433975, indicates a robust visual representation for each property, allowing for a comprehensive analysis of the listings.

$$\text{Average Number of Images per Listing} = \frac{\text{TotalNumberofImagesCollected}}{\text{TotalNumberofListings}}$$

The average number of images per listing is crucial for understanding the dataset's richness and the level of detail available for each property.

#### **Total Number of Images Collected:**

The total number of images collected across all listings in the database is 1,028,826. This substantial number of images underscores the scale of the dataset and the extensive effort involved in aggregating visual data from multiple real estate platforms.

## Image similarity model results

### Django app results

#### Experiment 1

The "Recommended for You" section of the real estate platform has impressively highlighted properties that, despite bearing different details in terms of size and pricing, share the exact same image. This scenario indicates that these properties might potentially be located within the same building or complex, demonstrating an interesting case of property listings that utilize the same photographic representation to appeal to potential buyers.

The image similarity algorithm has played a crucial role here by detecting that these images are identical with a very high similarity score of about 90.49%. This precision illustrates the algorithm's effectiveness not just in suggesting visually similar properties based on aesthetic and structural elements, but also in identifying exact image matches across different listings. This capability is particularly useful for users who are exploring options within a specific locality or building complex, as it consolidates what might initially appear as distinct options into a comprehensible set of choices that are visually identical.



## Recommended For You



### Άλλο είδος κατοικίας για αγορά - Καμίνια – Παλαιά Κοκκινιά

Bedrooms: 1

Bathrooms: 1

Size: 103

Address: Πειραιάς, Καμίνια – Παλαιά Κοκκινιά

Price per square meter: 20€

Agency: ALSVIT

Source: spiti24

Price: 2.015€

Similarity: 90.49%



### Διαμέρισμα για αγορά - Καμίνια – Παλαιά Κοκκινιά

Bedrooms: 1

Bathrooms: 1

Size: 49

Address: Πειραιάς, Καμίνια – Παλαιά Κοκκινιά

Price per square meter: 45€

Agency: ALSVIT

Source: spiti24

Price: 2.210€

Similarity: 90.49%



### Άλλο είδος κατοικίας για αγορά - Καμίνια – Παλαιά Κοκκινιά

Bedrooms: 1

Bathrooms: 1

Size: 103

Address: Πειραιάς, Καμίνια – Παλαιά Κοκκινιά

Price per square meter: 64€

Agency: ALSVIT

Source: spiti24

Price: 6.565€

Similarity: 90.49%

## Experiment 2

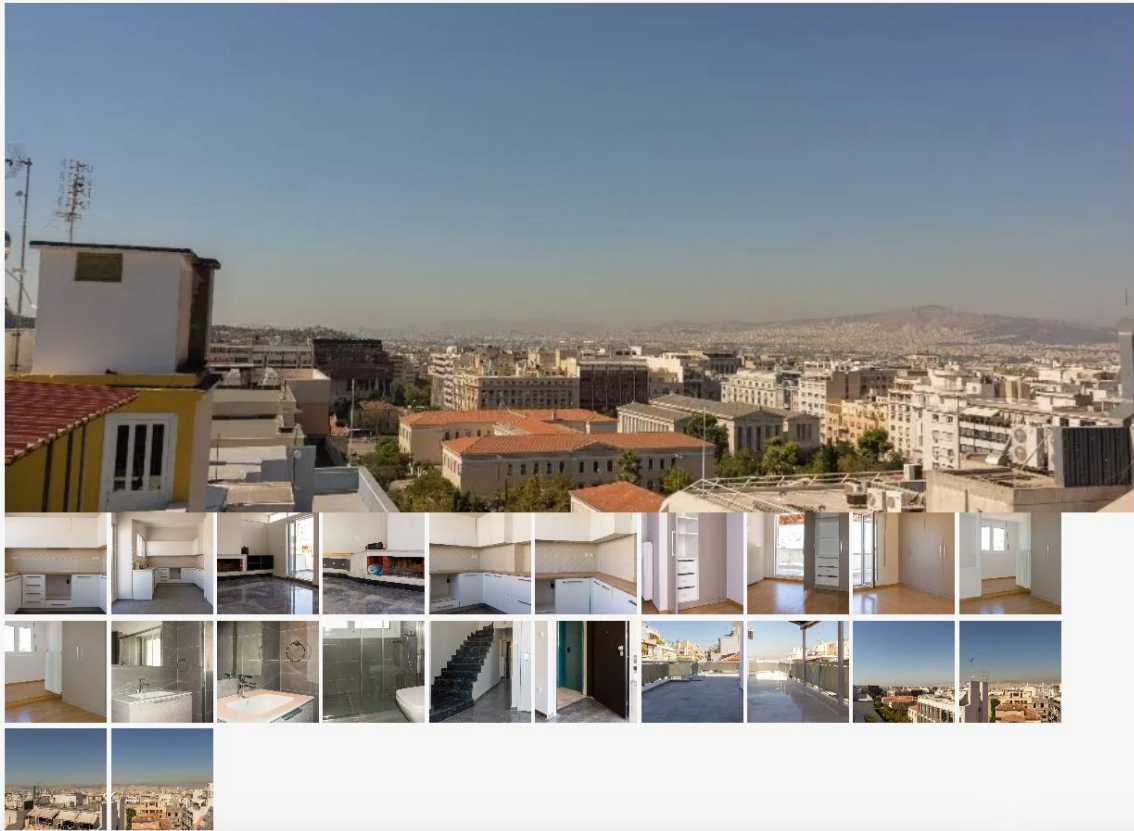
In the showcased example, the real estate application's image similarity algorithm has effectively identified and displayed multiple listings of the same property, marketed by different real estate agents and listed on various platforms. This listing, featuring a spacious terrace overlooking the city, is repeatedly encountered across several adverts with slight variations in description but identical visual presentation.

By deploying advanced image recognition technology, the application not only highlights identical property images but also consolidates their information, such as price differences, agency names, and listing details. This allows users to compare similar offerings efficiently and discern the best value or most convenient transaction available, exemplifying a robust application of AI in enhancing user experience and decision-making in real estate platforms.



# Μεζονέτα για αγορά - Κολωνάκι - Λυκαβηττός, Κολωνάκι

[Back to Listings](#)



## Recommended For You



### Μεζονέτα 184 τ.μ. για πώληση

Bedrooms: 3

Bathrooms: 2

Size: 368

Address: Κέντρο αθήνας » Κολωνάκι -  
λυκαβηττός

Price per square meter: 2.772€

Agency: Exclusive Agents

Source: plot

Price: 1.020.000€

**Similarity:** 87.92%



### Διαμέρισμα 184 τ.μ. για πώληση

Bedrooms: 3

Bathrooms: 2

Size: 368

Address: Κέντρο αθήνας » Κολωνάκι -  
λυκαβηττός

Price per square meter: 2.766€

Agency: EPSILON TEAM real estate PLUS

Source: plot

Price: 1.018.000€

**Similarity:** 88.17%



### Διαμέρισμα 64 τ.μ. για πώληση

Bedrooms: 2

Bathrooms: 2

Size: 128

Address: Κέντρο αθήνας » Εξάρχεια - νεάπολη »  
Νεάπολη εξαρχείων

Price per square meter: 1.797€

Agency: GOLDEN HOME A.E.

Source: plot

Price: 230.000€

**Similarity:** 86.96%



### Μεζονέτα προς πώληση Κολωνάκι (Κολωνάκι - Λυκαβηττός)

Bedrooms: 4

Bathrooms: Not specified

Size: 175

Address: Πλατεία Αττικής (Αττική)

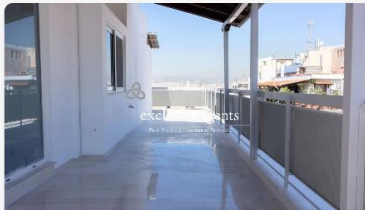
Price per square meter: 5.714€

Agency: Millennium Real Estate

Source: tospitimu

Price: 1.000.000€

**Similarity:** 85.14%



### Μεζονέτα για αγορά - Κολωνάκι - Λυκαβηττός, Κολωνάκι

Bedrooms: 3

Bathrooms: 2

Size: 184

Address: Κολωνάκι - Λυκαβηττός, Κολωνάκι

Price per square meter: 5.543€

Agency: Exclusive Agents

Source: spiti24

Price: 1.020.000€

**Similarity:** 87.54%



### Διαμέρισμα για αγορά - Κολωνάκι - Λυκαβηττός, Κολωνάκι

Bedrooms: 3

Bathrooms: 2

Size: 184

Address: Κολωνάκι - Λυκαβηττός, Κολωνάκι

Price per square meter: 6.250€

Agency: FUTURESTATE

Source: spiti24

Price: 1.150.000€

**Similarity:** 87.18%

## Image similarity program

In order to check the accuracy of the similar images detection and test the model, we created a Python application so that we can test the model. This program implements an image similarity search application using a graphical user interface (GUI) built with Tkinter. It allows users to upload an image, either by selecting it from their file system or by dragging and dropping it into the application. Once the image is uploaded, the application searches for visually similar images from a pre-existing dataset. These similar images, along with additional metadata (like links), are then displayed to the user. The program performs the following actions:

### Database Connection Setup

- **Connection String:** The code establishes a connection to a SQL Server database using the pyodbc library. The database contains a table with metadata (like links) related to the images.
- **Metadata Retrieval:** The application queries the database to fetch links associated with images that are determined to be similar to the uploaded image.

### Image Feature Extraction Using Deep Learning

The program uses a pre-trained ResNet50 model, a convolutional neural network (CNN) architecture, to extract feature vectors from images. This model has been trained on a large dataset (ImageNet) and is capable of capturing complex patterns and features in images, such as textures, edges, and shapes, which are crucial for distinguishing between different images. Extracted features and corresponding IDs are saved periodically to .npy files and managed through a checkpointing mechanism that allows the process to resume from the last point in case of interruption.

- **Feature Vector:** For each image, the ResNet50 model processes the image and outputs a high-dimensional vector (feature vector). This vector is a numerical representation of the image, capturing its essential characteristics.
- **Normalization:** The feature vectors are normalized to ensure that they have a consistent scale, which is important for accurate similarity calculations.

### Cosine Similarity for Image Comparison

The program compares images using cosine similarity, a metric that measures the cosine of the angle between two non-zero vectors in a multi-dimensional space. Mathematically, cosine similarity is expressed as:

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\| \|B\|}$$

Where A and B are feature vectors of two images. The cosine similarity score ranges from -1 to 1, with 1 indicating identical vectors (i.e., highly similar images), 0 indicating orthogonal vectors (i.e., dissimilar images), and -1 indicating diametrically opposed vectors.

**High Similarity Detection:** The program calculates the cosine similarity between the feature vector of an uploaded image and the feature vectors of images in a pre-existing database. Images with the highest similarity scores are considered to be potential duplicates.

## Database Integration and Metadata Retrieval

In addition to image comparison, the program integrates with a database that contains metadata for the listings, such as URLs to the listings on different platforms.

- **Link Retrieval:** For each similar image detected, the program retrieves the corresponding link from the database, which points to the real estate listing on its respective platform. This allows users to easily verify whether the listings are indeed duplicates.
- **Database Query:** The program queries the database using the unique identifiers (IDs) associated with the images to fetch the URLs.

## Graphical User Interface (GUI)

The program provides a user-friendly GUI where users can upload images, view the detected similar listings, and interact with the data.

- **Image Display:** The GUI displays the uploaded image along with the top similar images found in the database.
- **Clickable Links:** The links to the listings are displayed as clickable URLs, allowing users to quickly access the listings on different platforms.
- **Folder Access:** The program provides buttons to open the local folder containing the images, facilitating further investigation or manual verification of the images.

## Application in Real Estate

In the context of real estate, this program serves as a tool for image similarity and detection of similar listings. Real estate platforms often face issues with duplicate listings, which can confuse potential buyers and skew market data. By detecting and flagging these duplicates, the program helps maintain a clean and reliable database of property listings, ensuring that each property is represented accurately and uniquely across platforms.

The program presented is a GUI-based application designed to search and identify similar real estate property listings based on an uploaded image. The main goal of the application is to detect potential duplicate listings from different real estate agents or even the same agent across various platforms by comparing images of the properties.

## Performance Evaluation of the Image Similarity Detection Program

The program presented is a GUI-based application designed to search and identify similar real estate property listings based on an uploaded image. The main goal of the application is to detect potential duplicate listings from different real estate agents or even the same agent across various platforms by comparing images of the properties.

The Image Similarity Search Tool, as shown in Figure below, offers a streamlined and user-friendly interface for initiating searches by uploading images. Users can either click on the 'Upload Image' button or simply drag and drop an image into the designated area. This flexibility enhances user engagement and accessibility, allowing for quick comparisons of real estate images to identify potential duplicates or closely related properties.

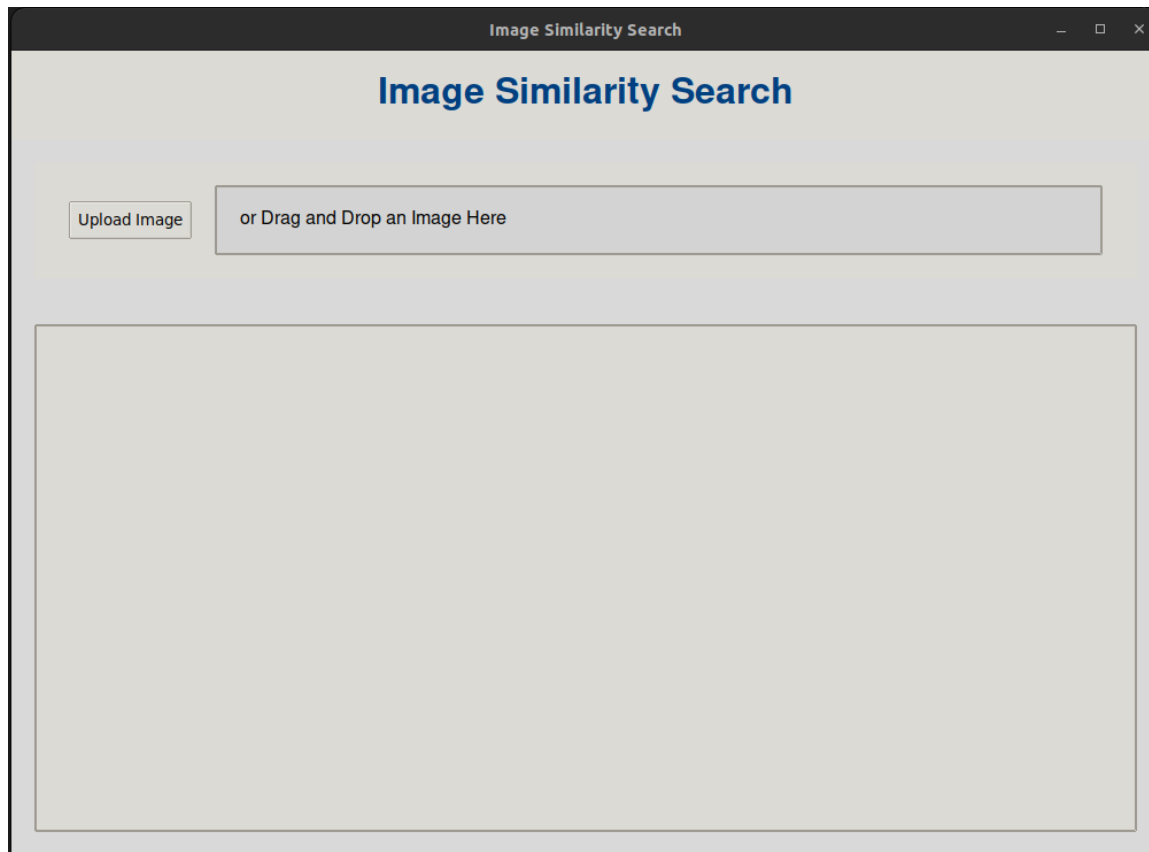


Figure 19: UI of the image similarity app

This chapter presents the experimental results obtained from the deployment of the Image Similarity Program, specifically designed to identify duplicate real estate listings across various online platforms. The experiments were structured to assess both the accuracy and efficiency of the image similarity algorithms when applied to diverse property images. The experiments were done on real word examples with images from our dataset as long as other images that are not included in our dataset. The significance of these experiments lies in their potential to transform how real estate listings are managed and searched, reducing redundancy and enhancing user experience. The results are demonstrated through a combination of tabulated data and screenshots of the user interface.

### Experiment 1

**Input image:**



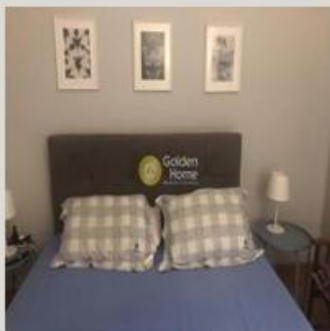
**Resulting matching ids:**

A screenshot of a search result interface. At the top, there is a small thumbnail image of a bedroom with a bed, a headboard, and three framed pictures on the wall. Below the thumbnail, the text reads "ID: 117723897" and "Similarity: 100.00%". Below this text is a button labeled "Open Folder". At the bottom, there is a blue hyperlink: "Link: <https://www.spitogatos.gr/aggelia/117723897>".

ID: 117723897  
Similarity: 100.00%

Open Folder

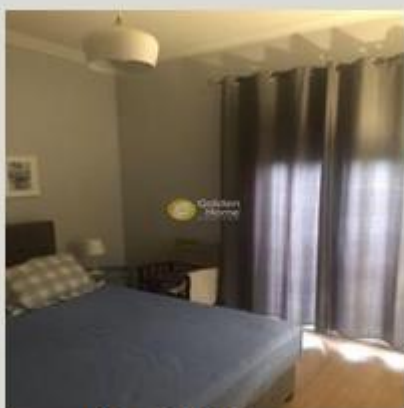
Link: <https://www.spitogatos.gr/aggelia/117723897>



ID: 7723897  
Similarity: 100.00%

Open Folder

Link: <https://www.tospitimou.gr/polisi-diamerisma-Ampelokipoi-Ampelokipoi-Pentagono/agelia/7723897?position=2748>



ID: 40965896  
Similarity: 99.09%

Open Folder

Link: <https://www.plot.gr/search/40965896-diamerisma-73-tm-gia-pwlisi>





ID: 40975304  
Similarity: 85.58%

Open Folder

Link: <https://www.plot.gr/search/40975304-diamerisma-90-tm-gia-pwlisi>

**Results:**

ID	Similar Image ID	Link	Similarity Percentage
1174261	117723897	<a href="https://www.spitogatos.gr/aggelia/117723897">https://www.spitogatos.gr/aggelia/117723897</a>	100.00%
1174261	7723897	<a href="https://www.tospitimou.gr/polisi-diamerisma-Ampelokipoi-Ampelokipoi-Pentagono/agelia/7723897?position=2748">https://www.tospitimou.gr/polisi-diamerisma-Ampelokipoi-Ampelokipoi-Pentagono/agelia/7723897?position=2748</a>	100.00%
1174261	40965896	<a href="https://www.plot.gr/search/40965896-diamerisma-73-tm-gia-pwlisi">https://www.plot.gr/search/40965896-diamerisma-73-tm-gia-pwlisi</a>	99.09%
1174261	40975304	<a href="https://www.plot.gr/search/40975304-diamerisma-90-tm-gia-pwlisi">https://www.plot.gr/search/40975304-diamerisma-90-tm-gia-pwlisi</a>	85.58%
1174261	93075500	<a href="https://www.spiti24.gr/93075500">https://www.spiti24.gr/93075500</a>	85.32%
1174261	15026600	<a href="https://www.tospitimou.gr/polisi-diamerisma-Thumarakia-Attiki/agelia/15026600?position=5594">https://www.tospitimou.gr/polisi-diamerisma-Thumarakia-Attiki/agelia/15026600?position=5594</a>	85.32%

The highest similarity scores (100.00%) achieved in this experiment confirm that the program can accurately identify when two images represent the same property, even when those listings are from different platforms. This accuracy was further validated through manual verification, where visits to the top three links confirmed that they indeed correspond to the same real estate listing, albeit advertised on different platforms. This demonstrates the program's robustness in identifying duplicates across disparate sources.

The similarity scores around 85% and lower also reveal the program's capability to detect subtler similarities that may not be immediately apparent. This includes variations in camera angles, lighting conditions, and minor changes in room arrangement, which are typical in real estate photography but do not necessarily indicate



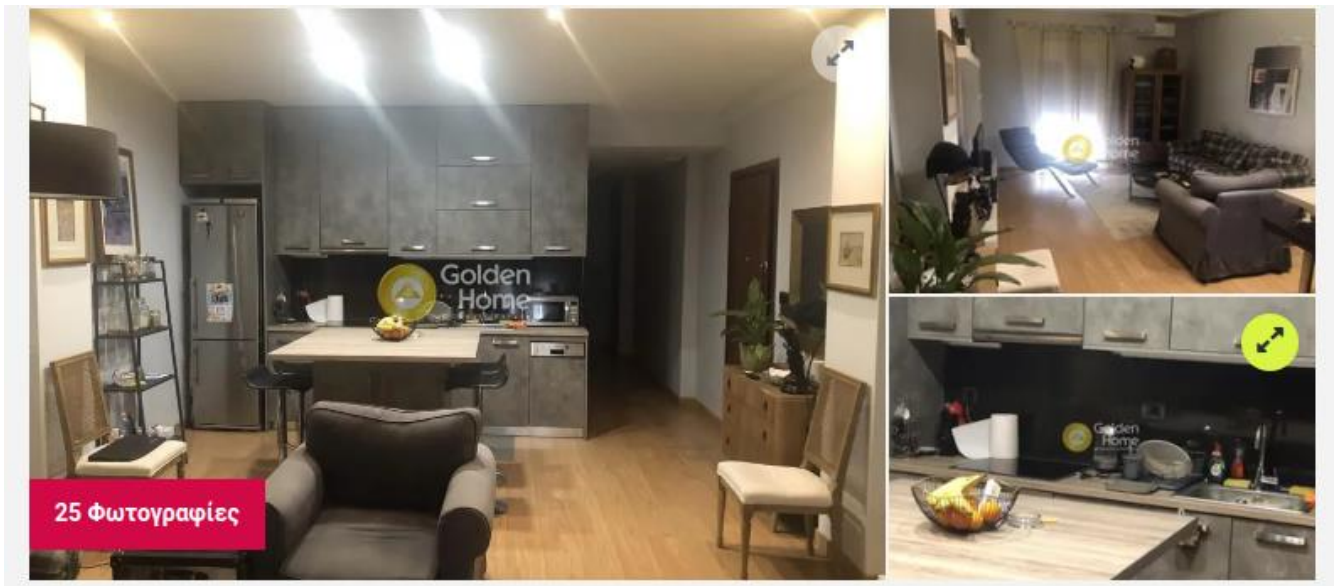
different properties. By accounting for these nuances, the program proves useful not just in outright duplicate detection but also in providing insights into listings that share many features but are photographed differently.

Implementing a threshold of 86.5% for similarity scoring in the Image Similarity Program allows us to balance sensitivity and specificity, enhancing the tool's practical utility in real estate market applications. This adjustment ensures that the program remains a reliable resource for identifying duplicate listings, supporting clearer, more accurate real estate databases and improving user experience on digital real estate platforms. The next experiments will be filtered to show only similar ids more than 86.5%.

Property on real estate platform 1:

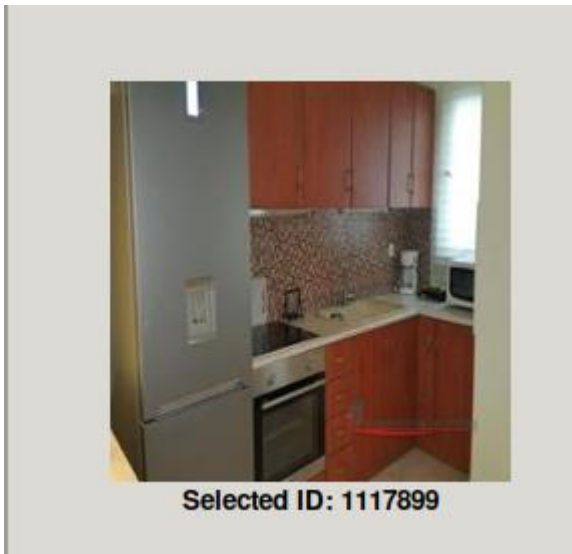


Property on real estate platform 2:



## Experiment 2

Input image:



Resulting matching ids:

	
ID: 1117890 Similarity: 100.00%	ID: 117772411 Similarity: 100.00%
<input type="button" value="Open Folder"/>	<input type="button" value="Open Folder"/>
Link: <a href="https://www.spiti24.gr/1117890">https://www.spiti24.gr/1117890</a>	Link: <a href="https://www.spitogatos.gr/aggelia/117772411">https://www.spitogatos.gr/aggelia/117772411</a>

**Results:**

The table below represents pairs of listings that were analyzed, including their IDs, the URLs for direct verification, and the similarity scores obtained:

ID	Similar Image ID	Link	Similarity Percentage
1117899	1117890	<a href="https://www.spiti24.gr/1117890">https://www.spiti24.gr/1117890</a>	100.00%
1117899	117772411	<a href="https://www.spitogatos.gr/aggelia/117772411">https://www.spitogatos.gr/aggelia/117772411</a>	100.00%
1117899	15400610	<a href="https://www.tospitimou.gr/polisi-diamerisma-Plaka-Istoriko-Kentro/aggelia/15400610?position=1451">https://www.tospitimou.gr/polisi-diamerisma-Plaka-Istoriko-Kentro/aggelia/15400610?position=1451</a>	86.81%
1117899	1115351660	<a href="https://www.spitogatos.gr/aggelia/1115351660">https://www.spitogatos.gr/aggelia/1115351660</a>	86.65%
1117899	41024148	<a href="https://www.plot.gr/search/41024148-diamerisma-110-tm-gia-pwlisi">https://www.plot.gr/search/41024148-diamerisma-110-tm-gia-pwlisi</a>	86.56%

Listings with 100% similarity are confirmed duplicates. For instance ID 1174261 with Similar Image ID 117723897 and Similar Image ID 7723897 both returned a similarity score of 100%. Visiting these URLs confirmed that these listings are indeed identical but posted on different platforms, showcasing the program's precision in identifying exact matches. Similarly, ID 1117899 matched 100% with ID 1117890 and ID 117772411, confirming that the program can detect duplicates without any error in these cases.

Listings with similarity scores in the high 80s to 99% indicate very close matches but with potential slight variations in the images due to factors like angle, lighting, or minor decor changes. For example ID 1174261 and ID 40965896 have a similarity score of 99.09%, suggesting an almost identical listing likely represented under slightly different conditions. Lower scores around 85%, such as ID 1174261 with ID 93075500 or ID 15026600, still suggest strong similarities. These could involve more noticeable changes in the photograph's setup but still represent the same property.

Property on real estate platform 1:

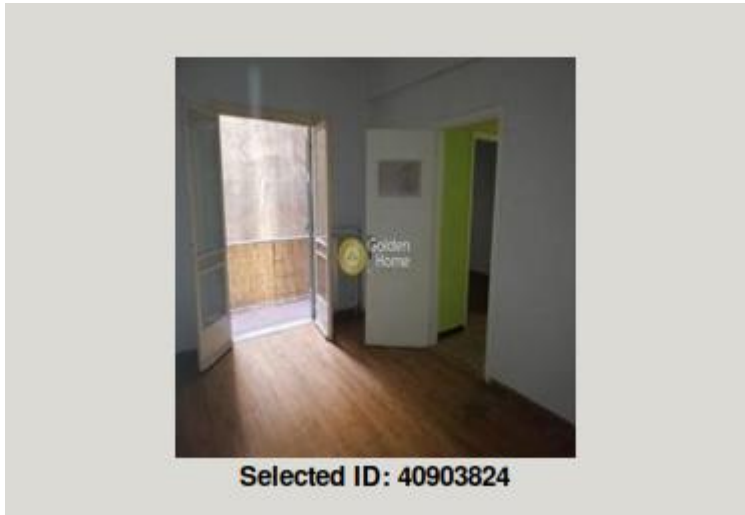


Property on real estate platform 2:



### Experiment 3

Input image:



Resulting matching ids:



**ID: 94667016**  
**Similarity: 99.55%**



**ID: 40771551**  
**Similarity: 92.78%**





**Results:**

ID	Similar Image ID	Link	Similarity Percentage
40903824	94667016	<a href="https://www.spiti24.gr/94667016">https://www.spiti24.gr/94667016</a>	99.55%
40903824	40771551	<a href="https://www.plot.gr/search/40771551-diamerisma-96-tm-gia-pwlisi">https://www.plot.gr/search/40771551-diamerisma-96-tm-gia-pwlisi</a>	92.78%
40903824	98987397	<a href="https://www.spiti24.gr/98987397">https://www.spiti24.gr/98987397</a>	92.49%
40903824	93009251	<a href="https://www.spiti24.gr/93009251">https://www.spiti24.gr/93009251</a>	89.96%
40903824	98006466	<a href="https://www.spiti24.gr/98006466">https://www.spiti24.gr/98006466</a>	87.60%
40903824	1115501724	<a href="https://www.spitogatos.gr/aggelia/1115501724">https://www.spitogatos.gr/aggelia/1115501724</a>	87.38%
40903824	1918615	<a href="https://www.spiti24.gr/1918615">https://www.spiti24.gr/1918615</a>	87.36%
40903824	1115209893	<a href="https://www.spitogatos.gr/aggelia/1115209893">https://www.spitogatos.gr/aggelia/1115209893</a>	87.13%
40903824	40909701	<a href="https://www.plot.gr/search/40909701-diamerisma-100-tm-gia-pwlisi">https://www.plot.gr/search/40909701-diamerisma-100-tm-gia-pwlisi</a>	87.02%
40903824	93907960	<a href="https://www.spiti24.gr/93907960">https://www.spiti24.gr/93907960</a>	86.87%

The results of the experiment conducted on the listing ID 1174261 illustrate the core problem statement of our study—identifying and linking duplicate real estate listings that may be posted by different agents or on various platforms. This listing is a prime example of how frequently the same property can be found across multiple platforms, underscoring the necessity for an effective image similarity tool in the real estate market.

Analysis of Listing ID 1174261

**Similarity Scores and Verification**

- 100% Similarity Scores: The links associated with Similar Image IDs 117723897 and 7723897 both returned 100% similarity scores. Upon manually verifying these URLs, it was confirmed that they indeed represent the same property as ID 1174261, but were listed under different descriptions and possibly by different agents on different platforms (Spitogatos.gr and Tospitimou.gr). We even detected this specific property in the same platform from different agents.
- High Similarity Scores: The other similarity scores ranging from 99.09% to 83.47% also demonstrate the tool's capability to detect significant similarities that might not be outright duplications but share substantial features. These scores likely represent the same or very similar properties with variations in how the photos were taken or processed, differences in staging, or updates over time.

Property on same real estate platform from different agents:



Property on different real estate platforms:

Διαμέρισμα 96 τ.μ. για πώληση, Αθήνα - Κέντρο, Λεωφ. Πο

[πλήρης αγγελία](#)



1/16





## Conclusion and Future Work

The research undertaken in this thesis has successfully demonstrated the application of advanced web scraping techniques, machine learning models for image similarity, and full-stack development using Django to create a robust platform for real estate data analysis in Greece. The integration of these technologies has facilitated a deeper understanding of the Greek real estate market, providing a comprehensive tool for users to access and analyze property data efficiently.

- **Key Contributions:** The development of a system that not only automates the collection and processing of real estate data but also implements a sophisticated image similarity algorithm to enhance user experience by recommending similar properties.
- **Challenges Encountered:** Among the challenges faced were dealing with data redundancy across multiple platforms, ensuring data privacy and compliance with real estate regulations, and overcoming technical hurdles related to web scraping and data integration.

The development and implementation of web scraping systems for real estate data analysis, as detailed in this thesis, have proven highly effective for gathering and processing information from Greek real estate platforms. Given the success of these methods, there is a significant opportunity to extend the scope of these scraping systems to encompass a broader array of platforms. This expansion would not only enhance the comprehensiveness of the data but also enrich the analysis capabilities of the system.

This thesis has laid a solid foundation for the application of web scraping and image similarity methods in analyzing real estate properties in Greece. The proposed future directions aim to expand the scope and enhance the capabilities of these systems, leveraging technological advances and responding to the evolving needs of the real estate market.

### Expanding Web Scraping to Broader Platforms

**Global Expansion:** Extending the web scraping system to include real estate platforms from different countries and regions would provide a more comprehensive global market analysis. This requires adapting the scraping tools to handle various languages and regional data formats, potentially incorporating automatic language translation and regional data normalization capabilities.

- **Commercial and Industrial Properties:** Expanding the scope to include commercial and industrial properties would diversify the data collected, providing insights into more sectors of the real estate market. This involves customizing scraping parameters to capture unique features of these properties, such as zoning information, commercial use, and facility specifications.
- **Integration with Emerging Markets:** Including platforms from emerging markets could uncover new investment opportunities and market trends that are not well-represented in traditional or established real estate databases. This would involve overcoming challenges related to less standardized data presentation and lower digital presence.

### Enhancing Image Similarity Methods

**Advanced Algorithmic Approaches:** Employing more sophisticated algorithms for image similarity could improve the accuracy and efficiency of property comparisons. Techniques such as deep learning and neural networks could be explored to enhance feature detection and matching processes.

**Cross-Platform Image Analysis:** Developing methods to analyze and compare images across different platforms would help in identifying duplicate listings and providing a unified view of properties listed on multiple platforms. This requires robust image processing tools capable of normalizing and comparing images from diverse sources.

**Dynamic Image Analysis Tools:** Implementing tools that allow dynamic interaction with image data, such as real-time similarity scoring or interactive visualization of similar properties, would significantly enhance user engagement and decision-making processes.

### **Addressing Data Privacy and Ethical Considerations**

**Data Privacy Protocols:** As web scraping scales up, strict protocols must be established to ensure compliance with international data privacy laws such as GDPR. This involves implementing secure data handling practices, anonymizing personal data, and obtaining necessary permissions when scraping sensitive information.

**Ethical Scraping Practices:** Developing a code of conduct for ethical web scraping in real estate, which respects website terms of use and avoids overloading servers, ensuring that scraping activities do not adversely affect the performance or accessibility of the original data sources.

**Transparency in Data Use:** Maintaining transparency about the sources and methods of data collection and analysis, particularly when used in making investment decisions or policy recommendations, is essential to build trust among users and stakeholders.

The potential expansions and enhancements proposed for the web scraping and image similarity systems are poised to transform real estate data analysis fundamentally. By embracing a global perspective, diversifying property types, and integrating advanced technological solutions, these systems can provide more accurate, comprehensive, and actionable insights. Additionally, addressing ethical and privacy concerns will ensure that these developments not only advance the field technically but also maintain the highest standards of integrity and respect for individual rights. This forward-looking approach will position the research at the forefront of technological innovation in real estate analysis, setting a benchmark for future academic and practical applications in the industry.

## References

1. **Guerrero Ramírez, Cuauhtémoc, Cerriteño Magaña, Javier, & Ramos, Diego.** (2024). Exploring the Rental Market Dynamics of the Guadalajara Metropolitan Area. *Journal of Real Estate Research*. [DOI: 10.13140/RG.2.2.30921.15203].
2. **Kim, Sang-Yoon & Lee, Hye-Jin.** (2024). Media Framing in the Digital Age: Interplay of Real Estate and Welfare Narratives in South Korean News Articles. *Asian Journal of Media Studies*.
3. **Johnson, Tyler & Wang, Li.** (2024). Forecasting Housing Price Using GRU, LSTM, and Bi-LSTM for California. *Journal of Forecasting and Data Science*.
4. **Silva, Ricardo & Costa, Ana.** (2024). Modelo Random Forest Aplicado a Precificação de Imóveis à Venda em Aracaju, SE. *Revista Brasileira de Engenharia e Tecnologia*.
5. **Patel, Anil & Shah, Ramesh.** (2024). Enhancing Real Estate Market Insights through Machine Learning: Predicting Property Prices with Advanced Data Analytics. *International Journal of Data Analytics*.
6. **Ahmed, Muhammad & Khan, Nadeem.** (2024). Applying Machine Learning Models for Forecasting House Prices – A Case of the Metropolitan City of Karachi. *Pakistan Journal of Data Science*.
7. **Van den Berg, Peter & Janssens, Klaas.** (2024). Scrimmo: A Real-Time Web Scraper Monitoring the Belgian Real Estate Market. *European Journal of Real Estate Technology*.
8. **Ivanov, Dmitry & Petrov, Sergey.** (2024). ML-based Telegram Bot for Real Estate Price Prediction. *Journal of AI and Software Applications*.
9. **Zukauskas, Andrius & Mikalajunas, Rytis.** (2024). Predictive Analytics Using Big Data for the Real Estate Market During the COVID-19 Pandemic. *Baltic Journal of Big Data*.
10. **Perera, Dilan & Silva, Tharindu.** (2024). A Model for the Estimation of Land Prices in Colombo District Using Web Scraped Data. *Sri Lankan Journal of Real Estate Research*.
11. **Rodríguez, Carlos & Patel, Suman.** (2024). Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application. *Journal of Data Science and Applications*.
12. **Chen, Lin & Wong, Tony.** (2024). Web Scraping Methods Used in Predicting Real Estate Prices. *International Journal of Real Estate Analytics*.

## Code

### **image\_similarity\_training.py**

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import logging
import json

# Setup logging
logging.basicConfig(level=logging.INFO)

# Load the saved features and IDs
features = np.load('features.npy')
ids = np.load('ids.npy')

# Define the base path where the folders are stored
base_image_path = 'listing_images' # Replace with your actual path

# List all folder names (IDs)
folder_ids = set(os.listdir(base_image_path))

# Convert the unique IDs from the ids array into a set
vector_ids = set(np.unique(ids))

# Find the IDs that are in the folder but not in the vector IDs
missing_ids = folder_ids - vector_ids

print(f"Total number of folders: {len(folder_ids)}")
print(f"Total number of unique IDs in vectors: {len(vector_ids)}")
print(f"Number of missing IDs: {len(missing_ids)}")
print(f"Missing IDs: {missing_ids}")

# Initialize the model (ensure your TensorFlow is configured to use the
GPU)
base_model = ResNet50(weights='imagenet', include_top=False, pooling='avg')
```

```

model = tf.keras.models.Model(inputs=base_model.input, outputs=base_model.output)

def extract_features(image_path, model):
    try:
        img = load_img(image_path, target_size=(224, 224))
        img_array = img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = tf.keras.applications.resnet50.preprocess_input(img_array)

        features = model.predict(img_array)
        return features.flatten()
    except Exception as e:
        logging.error(f"Error processing {image_path}: {e}")
        return None

def list_dir_with_retry(directory, max_retries=3, wait_time=5):
    retries = 0
    while retries < max_retries:
        try:
            return os.listdir(directory)
        except Exception as e:
            logging.warning(f"Error accessing {directory}: {e}. Retrying in {wait_time} seconds...")
            time.sleep(wait_time)
            retries += 1
    logging.error(f"Failed to access {directory} after {max_retries} retries.")
    return None

def append_to_npy(file_path, new_data):
    if new_data.size == 0: # Ensure there's something to append
        logging.warning(f"No new data to append to {file_path}. Skipping.")
        return

    if os.path.exists(file_path):
        existing_data = np.load(file_path)
        if existing_data.shape[-1] != new_data.shape[-1]:

```

```

        logging.error(f"Shape mismatch: existing data shape {existing_data.shape} and new data shape {new_data.shape}.")
        return
        combined_data = np.concatenate((existing_data, new_data))
    else:
        combined_data = new_data

    np.save(file_path, combined_data)

# Paths and settings
output_features_file = 'features.npy'
output_ids_file = 'ids.npy'
checkpoint_file = 'checkpoint.json'

features_list = []
ids_list = []

images_to_process = 10000 # Number of images to process per run
processed_images = 0 # Counter for processed images

# Load checkpoint if it exists and is valid
if os.path.exists(checkpoint_file):
    try:
        with open(checkpoint_file, 'r') as f:
            checkpoint = json.load(f)
            start_folder = checkpoint.get('last_folder', None)
            start_image = checkpoint.get('last_image', None)
    except (json.JSONDecodeError, ValueError) as e:
        logging.warning(f"Checkpoint file is empty or corrupted: {e}. Starting from the beginning.")
        checkpoint = {}
        start_folder = None
        start_image = None
else:
    checkpoint = {}
    start_folder = None
    start_image = None

```

```

# Process only the missing folders
missing_folders = sorted(list(missing_ids))
start_index = missing_folders.index(start_folder) if start_folder else 0

for folder_index in range(start_index, len(missing_folders)):
    folder_name = missing_folders[folder_index]
    folder_path = os.path.join(base_image_path, folder_name)
    if os.path.isdir(folder_path):
        logging.info(f"Processing folder {folder_index + 1}/{len(missing_folders)}: {folder_name}")

        images = list_dir_with_retry(folder_path)
        if images is None:
            continue # Skip this folder if it couldn't be accessed

        start_img_index = images.index(start_image) if folder_name ==
start_folder and start_image else 0

        for image_index in range(start_img_index, len(images)):
            image_name = images[image_index]
            image_path = os.path.join(folder_path, image_name)
            logging.info(f"Processing image {image_name} in folder
{folder_name}...")
            features = extract_features(image_path, model)
            if features is not None:
                features_list.append(features)
                ids_list.append(folder_name)
                processed_images += 1

        # Save checkpoint and stop if the limit is reached
        if processed_images >= images_to_process:
            checkpoint = {'last_folder': folder_name, 'last_image': im-
age_name}

            with open(checkpoint_file, 'w') as f:
                json.dump(checkpoint, f)

            logging.info(f"Processed {processed_images} images. Check-
point saved. Exiting.")

```

```

# Convert to numpy arrays for appending
features_array = np.array(features_list)
ids_array = np.array(ids_list)

# Append new features and IDs to existing files
append_to_npy(output_features_file, features_array)
append_to_npy(output_ids_file, ids_array)

break # Exit the loop after saving the checkpoint

if processed_images >= images_to_process:
    break # Exit the outer loop if processing limit is reached

# Convert to numpy arrays for appending
features_array = np.array(features_list)
ids_array = np.array(ids_list)

# Append the final batch of features and IDs to existing files
append_to_npy(output_features_file, features_array)
append_to_npy(output_ids_file, ids_array)

logging.info(f"Final save after processing {processed_images} images.")
logging.info("Feature extraction completed.")

```



## **test\_image\_similarity.py**

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.metrics.pairwise import cosine_similarity
import tkinter as tk
from tkinter import ttk, filedialog
from tkinterdnd2 import DND_FILES, TkinterDnD
from PIL import Image, ImageTk
import logging
from sklearn.preprocessing import normalize
import subprocess
import sys
import pyodbc
import webbrowser

# Database connection string
connection_string = (
    'DRIVER={ODBC Driver 17 for SQL Server};'
    'SERVER=localhost;'
    'DATABASE=listings;'
    'UID=sa;'
    'PWD=*****;'
    'TrustServerCertificate=yes;'
)

# Load the saved features and IDs
features = np.load('features.npy')
ids = np.load('ids.npy')

# Normalize features
features = normalize(features)

# Setup logging
logging.basicConfig(level=logging.INFO)
```

```

# Check if GPU is available and set memory growth
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print(f"GPU(s) available: {gpus}")
    except RuntimeError as e:
        print(f"Error setting GPU memory growth: {e}")
else:
    print("No GPU found, using CPU.")

# Initialize the model
with tf.device('/GPU:0' if gpus else '/CPU:0'):
    base_model = ResNet50(weights='imagenet', include_top=False, pooling='avg')
    model = tf.keras.models.Model(inputs=base_model.input, outputs=base_model.output)

# Path to images
base_image_path = 'listing_images'

def extract_features(image, model):
    img_array = img_to_array(image)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = tf.keras.applications.resnet50.preprocess_input(img_array)
    features = model.predict(img_array)
    return features.flatten()

def find_similar_images(uploaded_image_features, selected_id, top_n=10):
    similarities = cosine_similarity(uploaded_image_features, features)
    similar_indices = similarities[0].argsort()[::-1]

    unique_similar_ids = {}
    for idx in similar_indices:
        similar_id = ids[idx]

```

```

        if similar_id != selected_id and similar_id not in unique_similar_ids:
            # Exclude the same ID and duplicates
            unique_similar_ids[similar_id] = similarities[0][idx]
        if len(unique_similar_ids) >= top_n:
            break

    return list(unique_similar_ids.keys()), list(unique_similar_ids.values())

def get_link_from_database(id):
    try:
        with pyodbc.connect(connection_string, timeout=10) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT Link FROM listings WHERE ID = ?", id)
            link = cursor.fetchone()
            return link[0] if link else "No link found"
    except Exception as e:
        print(f"Failed to retrieve link from database: {str(e)}")
        return "No link found"

def get_links_from_database(similar_ids):
    links = []
    try:
        with pyodbc.connect(connection_string, timeout=10) as conn:
            cursor = conn.cursor()
            for similar_id in similar_ids:
                cursor.execute("SELECT Link FROM listings WHERE ID = ?",
similar_id)
                link = cursor.fetchone()
                if link:
                    links.append(link[0])
                else:
                    links.append("No link found")
    except Exception as e:
        print(f"Failed to retrieve links from database: {str(e)}")
    return links

def open_folder(similar_id):

```

```

folder_path = os.path.join(base_image_path, similar_id)
if os.path.exists(folder_path):
    if os.name == 'nt': # Windows
        os.startfile(folder_path)
    elif os.name == 'posix': # macOS or Linux
        subprocess.Popen(['open', folder_path] if sys.platform == 'darwin' else ['xdg-open', folder_path])
else:
    print(f"Folder {folder_path} does not exist.")

def open_link(event, link):
    webbrowser.open_new_tab(link)

def display_similar_images(uploaded_image, selected_id, similar_ids, similarity_scores, links):
    # Clear previous results
    for widget in results_frame.winfo_children():
        widget.destroy()

    # Store references to PhotoImage objects to prevent garbage collection
    image_refs = []

    # Display the query image and its ID
    uploaded_image.thumbnail((200, 200))
    img = ImageTk.PhotoImage(uploaded_image)
    image_refs.append(img) # Keep a reference
    query_panel = ttk.Label(results_frame, image=img, text=f"Selected ID: {selected_id}", compound=tk.TOP, font=("Helvetica", 10, "bold"))
    query_panel.grid(row=0, column=0, padx=10, pady=10)

    # Display the link for the selected image
    selected_link = get_link_from_database(selected_id) # Fetch the link for the selected image
    selected_link_label = ttk.Label(results_frame, text=f"Link: {selected_link}", foreground="blue", cursor="hand2")
    selected_link_label.grid(row=2, column=0, padx=10, pady=5)
    selected_link_label.bind("<Button-1>", lambda e: open_link(e, selected_link))

```

```

# Add a button to open the folder containing the searched image
open_button_query = ttk.Button(results_frame, text="Open Folder", com-
mand=lambda: open_folder(selected_id))
open_button_query.grid(row=1, column=0, padx=10, pady=5)

# Display similar images with listing IDs, similarity scores, and links
total_similarity = 0 # Track total similarity for accuracy calculation
for i, (similar_id, score, link) in enumerate(zip(similar_ids, similar-
ity_scores, links)):
    total_similarity += score
    similar_image_path = os.path.join(base_image_path, similar_id,
os.listdir(os.path.join(base_image_path, similar_id))[0])
    similar_img = Image.open(similar_image_path)
    similar_img.thumbnail((200, 200))
    img = ImageTk.PhotoImage(similar_img)
    image_refs.append(img) # Keep a reference
    similar_panel = ttk.Label(results_frame, image=img, text=f"ID:
{similar_id}\nSimilarity: {score:.2%}", compound=tk.TOP, font=("Helvetica",
10))
    similar_panel.grid(row=0, column=i+1, padx=10, pady=10)

# Add a clickable link
link_label = ttk.Label(results_frame, text=f"Link: {link}", fore-
ground="blue", cursor="hand2")
link_label.grid(row=2, column=i+1, padx=10, pady=5)
link_label.bind("<Button-1>", lambda e, url=link: open_link(e,
url))

# Add a button to open the folder containing the similar image
open_button = ttk.Button(results_frame, text="Open Folder", com-
mand=lambda id=similar_id: open_folder(id))
open_button.grid(row=1, column=i+1, padx=10, pady=5)

# Calculate and display the average accuracy
average_similarity = (total_similarity / len(similar_ids)) * 100 #
Convert to percentage
accuracy_label = ttk.Label(results_frame, text=f"Average Similarity:
{average_similarity:.2f}%", foreground="green", font=("Helvetica", 12,
"bold"))

```

```

    accuracy_label.grid(row=3, column=0, columnspan=len(similar_ids) + 1,
pady=10) # Adjust columnspan dynamically

# Store the references in the results_frame to ensure they persist
results_frame.image_refs = image_refs

def on_open_file(file_path=None):
    if not file_path:
        file_path = filedialog.askopenfilename()

    if file_path:
        selected_id = os.path.basename(os.path.dirname(file_path))

        uploaded_image = Image.open(file_path)
        uploaded_image_resized = uploaded_image.resize((224, 224))
        uploaded_image_features = extract_features(uploaded_image_resized,
model).reshape(1, -1)
        uploaded_image_features = normalize(uploaded_image_features) #
Normalize features

        # Find similar images
        similar_ids, similarity_scores = find_similar_images(uploaded_im-
age_features, selected_id)

        # Retrieve links from the database
        links = get_links_from_database(similar_ids)

        # Display the results
        display_similar_images(uploaded_image, selected_id, similar_ids,
similarity_scores, links)

def on_drag_and_drop(event):
    file_path = event.data
    if file_path.startswith("(") and file_path.endswith(")"):
        file_path = file_path[1:-1]
    on_open_file(file_path)

# Initialize the TkinterDnD application

```

```

root = TkinterDnD.Tk()
root.title("Image Similarity Search")
root.geometry("1000x700") # Set window size

# Configure the style
style = ttk.Style(root)
style.theme_use('clam')

# Header Frame
header_frame = ttk.Frame(root, padding=20)
header_frame.pack(fill=tk.X)

# Header Label
header_label = ttk.Label(header_frame, text="Image Similarity Search",
font=("Helvetica", 24, "bold"), foreground="#004080")
header_label.pack()

# Top Frame for Upload and Drop Zone
top_frame = ttk.Frame(root, padding=20)
top_frame.pack(fill=tk.X, padx=20, pady=20)

# Button to open file dialog
upload_button = ttk.Button(top_frame, text="Upload Image", com-
mand=on_open_file, style="Accent.TButton")
upload_button.pack(side=tk.LEFT, padx=10)

# Label for drag-and-drop
drop_label = ttk.Label(top_frame, text="or Drag and Drop an Image Here",
background="lightgray", relief="solid", padding=20, font=("Helvetica", 12))
drop_label.pack(fill=tk.BOTH, expand=True, padx=10)

# Results Frame to show query image and similar images
results_frame = ttk.Frame(root, padding=20, borderwidth=2, relief="solid")
results_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

# Enable drag and drop
root.drop_target_register(DND_FILES)
root.dnd_bind('<<Drop>>', on_drag_and_drop)

```

```
root.mainloop()
```



## **Spitogatos.py**

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import os
import warnings
import urllib3

proxy = "http://*****:@proxy.zenrows.com:8001"

def fetch_page_content(url, proxies):
    try:
        response = requests.get(url, proxies=proxies, verify=False)
        response.raise_for_status()
        return response.content
    except requests.RequestException as e:
        print(f"Error fetching page {url}: {e}")
        return None

def parse_page_content(content, existing_ids, property_type):
    soup = BeautifulSoup(content, 'html.parser')
    property_divs = soup.find_all('div', class_='tile__content')
    property_data = []

    for div in property_divs:
        link_element = div.find('a', class_='tile__link')
        link = link_element.get('href') if link_element else ''
        id_part = str(link.split('/')[-1]) if link else ''

        if id_part not in existing_ids:
            location = div.find('h3', class_='tile__location').text.strip()
            full_link = f'www.spitogatos.gr{link}' if link else ''
            property_info = {
                'Id': id_part,
                'Link': full_link,
                'property_Type': property_type
            }
            property_data.append(property_info)
```

```

        property_data.append(property_info)

    return property_data

def save_to_csv(data, filename):
    df = pd.DataFrame(data)
    # remove duplicate ids if any
    df = df.drop_duplicates(subset='Id', keep='first')
    # save the data
    if os.path.isfile(filename):
        df.to_csv(filename, mode='a', index=False, encoding='utf-8-sig',
header=False)
    else:
        df.to_csv(filename, mode='a', index=False, encoding='utf-8-sig')

def get_transaction_type():
    print("Choose the transaction type:")
    print("1: Pwliseis (Sales)")
    print("2: Enoikiaseis (Rentals)")
    choice = input("Enter your choice (1 or 2): ").strip()
    return "pwliseis" if choice == "1" else "enoikiaseis"

def get_property_type():
    print("Choose the property type to scrape:")
    print("1: Katoikies (Residences)")
    print("2: Epaggelmatikoi Xwroi (Commercial Spaces)")
    print("3: Gi (Land)")
    print("4: Loipa Akinita (Other Properties)")
    choice = input("Enter your choice (1-4): ").strip()

    property_types = {
        "1": "katoikies",
        "2": "epaggelmatikoi_xwroi",
        "3": "gi",
        "4": "loipa_akinita"
    }
}

```

```

return property_types.get(choice, "katoikies")

def get_region():
    print("Choose the region:")
    print("1: Athina Voreia Proastia (North Suburbs of Athens)")
    print("2: Athina Notia Proastia (South Suburbs of Athens)")
    print("3: Athina Dytika Proastia (West Suburbs of Athens)")
    print("4: Athina Kentro (Central Athens)")
    choice = input("Enter your choice (1-4): ").strip()

    # here are some of spitogatos regions. This list will be updated
    regions = {
        "1": "athina-voreia-proastia",
        "2": "athina-notia-proastia",
        "3": "athina-dytika-proastia",
        "4": "athina-kentro"
    }

    return regions.get(choice, "athina-kentro")

def scrape_spitogatos_listings():
    transaction_type = get_transaction_type()
    property_type = get_property_type()
    region = get_region()
    proxies = {"http": proxy, "https": proxy}

    # Suppress InsecureRequestWarning
    warnings.filterwarnings('ignore', category=urllib3.exceptions.Inse-
cureRequestWarning)

    # Get the directory of the current script
    script_dir = os.path.dirname(os.path.abspath(__file__))

    # Set the filename with the script directory path
    filename = os.path.join(script_dir, f'spitogatos_listings.csv')

    # Get starting and ending page numbers from the user

```

```

start_page = int(input("Enter the starting page number: "))
end_page = int(input("Enter the ending page number: "))

existing_ids = set()
if os.path.isfile(filename):
    existing_df = pd.read_csv(filename, dtype={'Id': str})
    existing_ids = set(existing_df['Id'])

for pnum in range(start_page, end_page + 1):
    url = f"https://www.spitogatos.gr/{transaction_type}-{property_type}/{region}/selida_{pnum}"
    print(f"Scraping page: {url}")
    content = fetch_page_content(url, proxies)
    if content:
        property_data = parse_page_content(content, existing_ids, property_type) # Pass property_type here
        save_to_csv(property_data, filename)

        # Update existing_ids to include new IDs
        for item in property_data:
            existing_ids.add(item['Id'])

# Main execution
if __name__ == "__main__":
    scrape_spitogatos_listings()

import pandas as pd
from urllib.parse import urlparse
import requests
from bs4 import BeautifulSoup
import os
from datetime import datetime
import logging # Import the logging module
import warnings
import re
from urllib3.exceptions import InsecureRequestWarning # Import InsecureRequestWarning

```

```

proxy = "http://f6ec51b68a3be984bc347e0656b0476154b744be:@proxy.zen-
rows.com:8001"
proxies = {"http": proxy, "https": proxy}

# Configure logging
logging.basicConfig(filename='scraping_log.txt', level=logging.INFO, for-
mat='% (asctime)s - %(levelname)s - %(message)s')

# Function to scrape data from a single URL
def scrape_property_data(url):
    # Temporarily suppress the InsecureRequestWarning
    warnings.filterwarnings("ignore", category=InsecureRequestWarning)
    # Perform a GET request to the URL
    response = requests.get(url, proxies=proxies, verify=False)
    if response.status_code != 200:
        print(f"Failed to retrieve data from {url}")
        return None

    # Parse the page content
    soup = BeautifulSoup(response.text, 'html.parser')
    print(f"Scraping page: {url}")
    # Extract the desired data

    # Initialize lists to store feature data
    # indoor_features = []
    # outdoor_features = []
    # construction_features = []
    # good_for = []

    # Extract the desired feature data
    parsed_url = urlparse(url)
    path_segments = parsed_url.path.split("/")
    Id = path_segments[-1]
    image_elements = soup.select('.property__gallery__item img')
    # Extract the src attributes of the image elements
    image_links = [img['src'] for img in image_elements]
    # Find the <p> element with class "property__description" and extract
    its text

```

```

description_element = soup.find('div', class_='property__description')
if description_element:
    description_text = description_element.text.strip()
else:
    description_text = 'n/a'

# Find the <span> element with class "property__address" and extract
its text
property_address_element = soup.find('span', class_='property__ad-
dress')
if property_address_element:
    property_address = property_address_element.text.strip()
else:
    property_address = 'n/a'

# Extract agency information
agency_element = soup.select_one('div.agencyInfo a')
if agency_element:
    agency_name = agency_element.get('title', 'n/a')
else:
    agency_name = 'n/a'

# Extract details into a dictionary
details = {}

# Add property_title and property_address to the details dictionary
details["Id"] = Id
details['link'] = url
details["Title"] = description_text
details["Address"] = property_address
details["images"] = image_links
details['Agency'] = agency_name
details['description'] = description_text

# Log the property ID to the logging file
logging.info(f"Scraped property with ID: {Id}")

# Locate the section containing property details

```

```

details_section = soup.find('dl', class_='property__details')

if details_section:
    # Find all <dt> and <dd> elements within the details section
    details_elements = details_section.find_all('dt')
    values_elements = details_section.find_all('dd')

    # Loop through details and values, and add them to the dictionary
    for detail, value in zip(details_elements, values_elements):
        detail_text = detail.text.strip()
        value_text = value.text.strip()

        # Check if a value exists before adding it to the dictionary
        if value_text:
            details[detail_text] = value_text

    # Initialize an empty list to store indoor benefits
    indoor_benefits = []
    # Initialize an empty list to store outdoor features
    outdoor_features_list = []
    # Initialize empty lists to store construction features and good-for
elements
    construction_features_list = []
    good_for_list = []

    # Extract indoor features
    indoor_section = soup.find("ul", {"data-test-id": "indoor"})
    if indoor_section:
        indoor_elements = indoor_section.select('li')
        for li in indoor_elements:
            svg_element_indoor = li.find('svg', class_='on icon sprite-
icons')

            if svg_element_indoor:
                svg_class = svg_element_indoor['class'] # Get the class
attribute of the SVG

                indoor_text = li.find('span').text.strip()
                if indoor_text != 'n/a':
                    indoor_benefits.append(indoor_text)

```

```

# Extract outdoor features
outdoor_section = soup.find("ul", {"data-test-id": "outdoor"})
if outdoor_section:
    outdoor_elements = outdoor_section.select('li')
    for li in outdoor_elements:
        svg_element_outdoor = li.find('svg', class_='on icon sprite-
icons')
        if svg_element_outdoor:
            svg_class = svg_element_outdoor['class'] # Get the class
attribute of the SVG
            outdoor_text = li.find('span').text.strip()
            if outdoor_text != 'n/a':
                outdoor_features_list.append(outdoor_text)

# Extract construction features
construction_section = soup.find("ul", {"data-test-id": "construc-
tion"})
if construction_section:
    construction_elements = construction_section.select('li')
    for li in construction_elements:
        construction_element = li.find('svg', class_='on icon sprite-
icons')
        if construction_element:
            construction_class = construction_element['class'] # Get
the class attribute of the SVG
            construction_text = li.find('span').text.strip()
            if construction_text != 'n/a':
                construction_features_list.append(construction_text)

# Extract good for
good_for_section = soup.find("ul", {"data-test-id": "goodfor"})
if good_for_section:
    good_for_elements = good_for_section.select('li')
    for li in good_for_elements:
        good_for_element = li.find('svg', class_='on icon sprite-
icons')
        if good_for_element:

```



```

        good_for_element_class = good_for_element['class'] # Get
the class attribute of the SVG
        good_for_element_text = li.find('span').text.strip()
        if good_for_element_text != 'n/a':
            good_for_list.append(good_for_element_text)

# Join feature values into single strings
indoor_benefits_str = ", ".join(indoor_benefits)
construction_features_str = ", ".join(construction_features_list)
good_for_str = ", ".join(good_for_list)
outdoor_features_str = ", ".join(outdoor_features_list)

# Concatenate all feature values into a single row
feature_data = {
    "Indoor_features": indoor_benefits_str,
    "Outdoor_features": outdoor_features_str,
    "Construction_features": construction_features_str,
    "Good_for": good_for_str,
}

# Combine property details and feature data into a single dictionary
combined_data = {**details, **feature_data}
return combined_data

# Read the CSV file with listing URLs
csv_file = 'spitogatos_listings.csv'
df_urls = pd.read_csv(csv_file)

# Initialize an empty DataFrame to store the scraped data
df = pd.DataFrame()

# Loop through each URL in the list and scrape data
for url in df_urls['Link']:
    # Check if the URL starts with 'http://' or 'https://'
    if not url.startswith('http://') and not url.startswith('https://'):
        # Add 'https://' to the URL
        url = 'https://' + url

```

```

# Check if the URL has already been scraped by looking for its ID in
the log file
parsed_url = urlparse(url)
path_segments = parsed_url.path.split("/")
Id = path_segments[-1]
if f"Scraped property with ID: {Id}" in open('scrap-
ing_log.txt').read():
    print(f"This property with ID {Id} has already been scraped.")
    continue # Skip this property

scraped_data = scrape_property_data(url)
if scraped_data:
    # Create a DataFrame for the current property and append it to
df_scraped_data
    df_property = pd.DataFrame([scraped_data])
    df = pd.concat([df, df_property], ignore_index=True)
    print(df)

def process_dataframe(df, columns_mapping):
    # Rename the columns based on the mapping
    df.rename(columns=columns_mapping, inplace=True)
    # # Reorder and drop columns based on the SQL database order
    return df

# Mapping of Greek column names to the SQL column names
columns_mapping = {
    "Τιμή": "Price",
    "Τιμή ανά τ.μ.": "price_per_sqm",
    "Εμβαδόν": "Area",
    "Επίπεδα": "levels",
    "Όροφος": "floor",
    "Κουζίνες": "kitchens",
    "Μπάνια": "bathrooms",
    "WC": "WC",
    "Σαλόνια": "living_rooms",
    "Σύστημα θέρμανσης": "heating",
    "Ενεργειακή κλάση": "energy_class",

```

```

"Ετος κατασκευής": "Construction_year",
"Ετος ανακαίνισης": "renovation_year",
"Απόσταση απο τη θάλασσα": "sea_distance",
"Κωδικός Συστήματος": "system_code",
"Κωδικός Ακινήτου": "code",
"Διαθέσιμο από": "available_from",
"Δημοσίευση αγγελίας": "published",
"Τελευταία ενημέρωση": "last_updated"
}

# Apply the function
df = process_dataframe(df, columns_mapping)

def clean_price(price_string):
    # If it's already a float (or not a string), return as is
    if not isinstance(price_string, str):
        return price_string

    # Split the string by non-digit characters and filter out empty splits
    numbers = [part for part in re.split('\D+', price_string) if part]

    # Join the numbers into one string and convert to float
    if len(numbers) > 0:
        cleaned = ''.join(numbers)
        return float(cleaned)
    else:
        return None

# Check if the DataFrame is not empty
if not df.empty:
    # Apply the function to the 'Price' column
    df['Price'] = df['Price'].apply(clean_price)

    # Uncomment the following line if 'price_per_sqm' column exists and
    # needs cleaning
    # df['price_per_sqm'] = df['price_per_sqm'].apply(clean_price)

```

```

# Check the updated 'Price' column
print(df['Price'].head())

# Construct the 'Title' column
df['Title'] = 'Προς πώληση ' + df['Address'].astype(str) + ' ' +
df['Area'].astype(str)

# Drop the duplicates if there are any
df = df.drop_duplicates(subset='Id', keep='first')

# Convert 'Id' in both DataFrames to string
df['Id'] = df['Id'].astype(str)
df_urls['Id'] = df_urls['Id'].astype(str)

# Perform the inner join on 'Id' column
df = pd.merge(df, df_urls[['Id', 'property_Type']], on='Id', how='inner')

# Save the DataFrame to a CSV file
timestamp = datetime.now().strftime("%Y_%m_%d_%H_%M")
file_name = f'spitogatos{timestamp}.csv'
if not os.path.isfile(file_name):
    df.to_csv(file_name, mode='w', index=False, encoding='utf-8-sig')
# Write with header
else:
    df.to_csv(file_name, mode='a', index=False, encoding='utf-8-sig')

```

## plot.py

```
# ### Get the listings
import os
import time
import re
import pandas as pd
from bs4 import BeautifulSoup
from sqlalchemy import create_engine
import pyodbc
from datetime import datetime
import pandas as pd
from sqlalchemy import create_engine, text
from sqlalchemy import inspect
import requests

proxy = "http://*****:@proxy.zenrows.com:8001"
proxies = {"http": proxy, "https": proxy}

# Create a dictionary to store the scraped data
data_dict = {}

# Create a list to store the dictionaries
data = []

# Get the current timestamp
timestamp = datetime.now()
# Convert the datetime object to a string in the desired format
timestamp = timestamp.strftime('%Y-%m-%d %H:%M:%S')

# Loop through the pages
for pnum in range(1, 2): # num_pages + 1 because range is exclusive at the
end
    try:
        url = f"https://www.plot.gr/search/?category=20004&context=map-
search&location=32&sort=cr&pg={pnum}"
        print(f"Scraping page: {url}")
        # Navigate to the webpage
```

```

#         response = requests.get(url)
response = requests.get(url, proxies=proxies, verify=False)
# Wait for the page to load
time.sleep(3)
soup = BeautifulSoup(response.content, 'html.parser')
listings = soup.find_all('div')

for listing in listings:
    title_element = listing.find('h2', class_='title')
    description_element = listing.find('span', class_='text-muted
test')
    price_element = listing.find('div', class_='price-tag current-
price')

    if title_element and description_element and price_element:
        title = title_element.text.strip()
        property_type = description_element.text.strip()
        price = price_element.text.strip()

        anchor = listing.find('a', class_='row-anchor')
        if anchor:
            link = 'http://www.plot.gr' + anchor['href']
            Id = link.split('-')[0].split('/')[-1]

            # Extract the location
            location_element = listing.find('div', class_='text-
muted loc')

            if location_element:
                location = location_element.span.text.strip()
            else:
                location = None

#         # Extract the last modified information
#         last_modified_element = listing.find('div',
class_='last-mod')
#         if last_modified_element:
#         last_modified = last_modified_ele-
ment.span.text.strip()

```

```

#             else:
#                 last_modified = None

                link_dict = {'Id': Id, 'type': property_type, 'title':
title, 'link': link, 'location': location, 'timestamp': timestamp}
                data.append(link_dict)
                print(data)
                pnum += 1

    except Exception as e:
        print(f"An error occurred: {e}")
        continue

#convert to pandas dataframe
plot_data = pd.DataFrame(data)
plot_data = plot_data.drop_duplicates(subset='Id', keep='first')

# Generate a timestamp using the current date and time
timestamp = datetime.now().strftime("%Y_%m_%d_%H_%M")

# Construct the file name with the timestamp
# file_name = f'plot.gr_{timestamp}.csv'
file_name = f'plot.gr.csv'

# Check if the file exists
if not os.path.isfile(file_name):
    plot_data.to_csv(file_name, mode='a', index=False, encoding='utf-8-
sig') # Write with header
else:
    plot_data.to_csv(file_name, mode='a', index=False, encoding='utf-8-
sig', header=False) # Append without writing header

# ### Get more details by clicking every link
# Define the output directory for saving images
output_directory = 'images'

# Create the output directory if it doesn't exist
os.makedirs(output_directory, exist_ok=True)

```

```

# Read the CSV file with links
csv_file_path = 'plot.gr.csv'
proxy = "http://f6ec51b68a3be984bc347e0656b0476154b744be:@proxy.zen-
rows.com:8001"
proxies = {"http": proxy, "https": proxy}
df = pd.read_csv(csv_file_path, encoding='utf-8-sig')

# Initialize an empty list to store the scraped data
property_data = []
row_data = {}

# Iterate through the links
for index, row in df.iterrows():
    try:
        link = row['link']

        # Extract the listing ID from the URL
        # listing_id = link.split('/')[ -1]

        # Add the row data to the data list
        property_data.append(row_data)

        # Print the URL being scraped
        print(f"Scraping data for Listing URL: {link}")

        # Make a GET request to the webpage
        response = requests.get(link, proxies=proxies, verify=False)

        if response.status_code == 200:
            # Parse the HTML content of the webpage
            soup = BeautifulSoup(response.content, 'html.parser')

            # Find the specifications table
            spec_table = soup.find('div', attrs={'id': 'specification-
table'})

```



```

# If the specifications table doesn't exist, skip this listing
if spec_table is None:
    print(f"Could not find specifications table on {link}")
    continue

# Initialize a dictionary to hold the scraped data
row_data = {'Id': listing_id}

# Find all the specification labels and values
specs = spec_table.find_all('span', attrs={'class': 'spec'})

for spec in specs:
    # Extract the label and value
    label = spec.find('span', attrs={'class': 'spec-label'}).text.strip()
    value = spec.find('span', attrs={'class': 'spec-value'}).text.strip()

    # Check if the label is one of the expected labels
    if label in row_data:
        # Add the value to the dictionary
        row_data[label] = value
    else:
        None

# Find the benefits
t = soup.find('ul', class_='tw-grid tw--my-2 tw-grid-cols-12 tw-gap-2.5')
if t:
    text = t.get_text(strip=True)
    benefits = ''.join(c if c.islower() or not c.isalpha() else ' ' + c for c in text).strip()
    row_data['benefits'] = benefits

# Find all divs with the class 'tw-break-words html tw-text-base'
elements = soup.find_all('div', class_='tw-break-words html tw-text-base')

```

```

# Initialize a dictionary to hold the text from both divs
descriptions = {'Description': ''}
# Find the specification table
spec_table = soup.find('div', {'id': 'specification-table'})

# Initialize a dictionary to store the extracted data
data = {}

# Find and extract individual specifications
for spec in spec_table.find_all('span', class_='spec'):
    label = spec.find('span', class_='spec-label').text.strip()
    value = spec.find('span', class_='spec-value').text.strip()
    data[label] = value

# Print the extracted data
for label, value in data.items():
    print(f"{label}: {value}")
# Iterate over the divs
for i, element in enumerate(elements):
    # Get the text and remove unwanted newline characters
    text = element.get_text(strip=True)
    text = text.replace('\r\n\r\n', ' ')

    # Save the text to the appropriate key in the dictionary
    descriptions[f'Description{i + 1}'] = text

# Add the descriptions to the row data
row_data.update(descriptions)

# Find all image tags with src attributes that end with ".jpg"
or ".jpeg"
jpeg_images = soup.find_all('img', src=lambda x:
x.endswith(('.jpg', '.jpeg')))

# Extract src attribute from jpeg_images and keep only the
links
image_links = [img['src'] for img in jpeg_images]
# Save each image

```

```

#         for i, img_url in enumerate(image_links):
#             response = requests.get(img_url, proxies=proxies, verify=False)

#             # Save the image as 'listing_id_0.jpg', 'listing_id_1.jpg', etc.
#             filename = f"{str(listing_id)}_{i}.jpeg"
#             filepath = os.path.join(output_directory, filename)

#             with open(filepath, 'wb') as out_file:
#                 out_file.write(response.content)

# Add urls to row_data with key 'images'
row_data['images'] = image_links

# Add the row data to the data list
property_data.append(row_data)
# Print a success message
print(f"Finished scraping Id: {listing_id}")

# Scrape real estate agent details
agent_div = soup.find('div', {'data-v-67356525': ""})
if agent_div:
    # Extract the agent's name
    agent_name_tag = agent_div.find('a', class_='tw-text-2xl tw-font-semibold tw-break-all')
    if agent_name_tag:
        agent_name = agent_name_tag.text.strip()
    else:
        agent_name = 'Ιδιώτης'

    # Extract the agent's address
    agent_address_tag = agent_div.find('div', class_='tw-py-2')
    if agent_address_tag:
        agent_address = agent_address_tag.get_text(separator=' ', strip=True)
    else:
        agent_address = 'Not found'

```

```

        # Extract the agent's location
        if ',' in agent_address:
            agent_location = agent_address.split(',')[1].strip()
        else:
            agent_location = 'Not found'

    else:
        agent_name = 'Ιδιώτης' # Or whatever default value you
want
        agent_address = 'Not found' # Or whatever default value
you want
        agent_location = 'Not found' # Or whatever default value
you want

    # Save the details to the row_data dictionary
    row_data['AgentName'] = agent_name
    row_data['AgentAddress'] = agent_address
    row_data['AgentLocation'] = agent_location

    print("Finished scraping all rows")
    # Combine row_data with data
    row_data.update(data)

    # Print all scraped details for this listing
    print("Scraped details for Listing ID:", listing_id)
    for key, value in row_data.items():
        print(f"{key}: {value}")

    # Add the row data to the data list
    property_data.append(row_data)
    # Print a success message
    print(f"Finished scraping Id: {listing_id}")

except requests.exceptions.Timeout:
    print(f"Timeout while trying to scrape {link}. The listing might be
deleted or the website structure might have changed.")
    continue

```

```

except Exception as e:
    print(f"An error occurred: {e}")
finally:
    print("Finished")

# Convert the list of dictionaries to a DataFrame
df = pd.DataFrame(property_data)

# Rename columns and remove colons
df.rename(columns={

    'Νούμερο αγγελίας:': 'Id',
    'Κατηγορία:': 'Category',
    'Τιμή:': 'Price',
    'Περιοχή:': 'Location',
    'Τιμή ανά τ.μ.':': 'Price_per_sqm',
    'Εμβαδόν:': 'Area',
    'Έτος κατασκευής:': 'Year_Built',
    'Σύστημα θέρμανσης:': 'Heating_System',
    'Χώροι:': 'Rooms',
    'Μέσο θέρμανσης:': 'Heating_Medium',
    'Ενεργειακή κλάση:': 'Energy_Class',
    'Όροφος:': 'Floor',
    'Τελευταία αλλαγή:': 'Last_Change',
    'Εμφανίσεις αγγελίας:': 'Ad_Views',
    'Σύνδεσμος:': 'Link',
    'Τηλέφωνο:': 'Phone',
    'Διαθέσιμο από:': 'Available_From',
    'Υπνοδωμάτια:': 'Bedrooms',
    'Μπάνια:': 'Bathrooms',
    'Κωδικός ακινήτου:': 'Property_Code',
    'Ζώνη:': 'Zone'
}, inplace=True)

# Reorder the columns to have 'Ad Number' as the first column
df = df[['Id'] + [col for col in df.columns if col != 'Ad Number']]

```

```

# Remove colons from the column names
df.columns = df.columns.str.replace(':', '')

# Display the DataFrame
print(df)

timestamp = datetime.now().strftime("%Y_%m_%d_%H_%M")

# Construct the file name with the timestamp
file_name = f'plot_details_{timestamp}.csv'

#drop the duplicates if there are any
df = df.drop_duplicates(subset='Id', keep='first')

# Check if the file exists
if not os.path.isfile(file_name):
    df.to_csv(file_name, mode='a', index=False, encoding='utf-8-sig') #
    Write with header
else:
    df.to_csv(file_name, mode='a', index=False, encoding='utf-8-sig',
    header=False) # Append without writing header
# Print a message when scraping and saving is complete
print("Scraping and saving complete.")

# Perform the join on 'Id'
df = plot_data.merge(df, on='Id', how='inner')

```

### **tospitimou.py**

```

#!/usr/bin/env python
# coding: utf-8

# Imports
import os

```

```

import time
import re
import pandas as pd
import requests
from bs4 import BeautifulSoup
from sqlalchemy import create_engine, inspect
from datetime import datetime
import logging

# Setup logging
logging.basicConfig(
    filename="scraping_log.txt",
    level=logging.INFO,
    format="%(asctime)s: %(levelname)s: %(message)s",
)

# Proxy configuration
proxy = "http://f6ec51b68a3be984bc347e0656b0476154b744be:@proxy.zen-rows.com:8001"
proxies = {"http": proxy, "https": proxy}

# Function to scrape data
def scrape_data():
    data = []

    url_base = "https://www.tospitimou.gr/akinita/poliseis/katoikies/Kentro-Athinas/area-ids_%5B100%5D,category_residential,floor-number-high_ground-floor?sortBy=price%7Casc&page="

    for pnum in range(1, 3):
        url = f"{url_base}{pnum}"
        try:
            time.sleep(10)
            response = requests.get(url, proxies=proxies, verify=False)
            soup = BeautifulSoup(response.content, 'html.parser')
            property_divs = soup.find_all('div', class_='search-result-inner')

            for property_div in property_divs:
                data_row = extract_property_data(property_div)

```

```

        if data_row:
            data.append(data_row)
            logging.info(f"Scraped page: {url}")
    except Exception as e:
        logging.error(f"Failed to scrape page {url}: {e}")
        continue

# Store data
df = pd.DataFrame(data)
store_data(df, 'tospitimou_data.csv')

# Function to extract property data
def extract_property_data(property_div):
    try:
        title = property_div.find('h2', class_='searchRe-
sultsH2').text.strip()
        link = property_div.find('a')['href']
        image_url = property_div.find('img', class_='bg-image-un-
veil')['data-src']
        price = property_div.find('div', class_='result-
price').text.strip()

        # Additional details like area, price per sq.m, bedrooms could be
added here

        return {'Title': title, 'Link': link, 'Image URL': image_url,
'Price': price}
    except AttributeError as e:
        logging.error(f"Error extracting data: {e}")
        return None

# Function to store data
def store_data(df, file_name):
    if not os.path.isfile(file_name):
        df.to_csv(file_name, index=False, encoding='utf-8-sig')
    else:
        df.to_csv(file_name, mode='a', index=False, encoding='utf-8-sig',
header=False)

```



```

# Main function
def main():
    scrape_data()
    print("Scraping and saving complete.")

if __name__ == "__main__":
    main()

import pandas as pd
import requests
from bs4 import BeautifulSoup
import urllib3
from datetime import datetime
import os
import logging

# Setup logging
logging.basicConfig(
    filename="scraping_log.txt",
    level=logging.INFO,
    format="%(asctime)s: %(levelname)s: %(message)s"
)

# Proxy and requests setup
proxy_user = 'f6ec51b68a3be984bc347e0656b0476154b744be'
proxy = f"http://{proxy_user}:@proxy.zenrows.com:8001"
proxies = {"http": proxy, "https": proxy}
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Read scraped links if available
def read_scraped_links(file_path):
    try:
        with open(file_path, 'r') as f:
            return [link.strip() for link in f.readlines()]
    except FileNotFoundError:
        return []

def scrape_properties(df_links, scraped_links):

```

```

property_list = []
for link in df_links['Link']:
    property_id = link.split("/")[-1].split("?")[0]
    logging.info(f"Scraping: {link}")

    if f"Scraped property with ID: {property_id}" in scraped_links:
        logging.info(f"Skipping already scraped property with ID:
{property_id}")
        continue

    response = requests.get(link, proxies=proxies, verify=False)
    soup = BeautifulSoup(response.content, 'html.parser')
    property_info = extract_property_data(soup, property_id)
    if property_info:
        property_list.append(property_info)
        logging.info(f"Scraped property with ID: {property_id}")
    else:
        logging.error(f"Failed to scrape property with ID: {prop-
erty_id}")

return property_list

def extract_property_data(soup, property_id):
    try:
        data = {
            "Id": property_id,
            "Title": soup.find('h1', class_='listing-title').text.strip()
if soup.find('h1', class_='listing-title') else 'n/a',
            "Price": soup.find('span', class_='property-
info2').text.split(',')[0].strip() if soup.find('span', class_='property-
info2') else 'n/a',
            "Area": soup.find('span', class_='property-
info2').text.split(',')[1].strip() if soup.find('span', class_='property-
info2') else 'n/a',
            "Photo_count": soup.find('span', class_='photo-
count').text.split()[0] if soup.find('span', class_='photo-count') else
'N/A',
            "Images": [a.get('href') for a in soup.find('div',
class_='photo-gallery').find_all('a')] if soup.find('div', class_='photo-
gallery') else [],

```

```

        "Description": soup.find('div', class_='panel-
body').find('p').text.strip() if soup.find('div', class_='panel-body') and
soup.find('div', class_='panel-body').find('p') else 'n/a',
        "Latitude": soup.find('div', class_='marker').get('data-lat')
if soup.find('div', class_='marker') else None,
        "Longitude": soup.find('div', class_='marker').get('data-lng')
if soup.find('div', class_='marker') else None,
        "Amenities": [li.text.strip() for li in soup.find('ul',
class_='property-amenities list-unstyled').find_all('li')] if
soup.find('ul', class_='property-amenities list-unstyled') else []
    }
    return data
except Exception as e:
    logging.error(f"Error extracting property data: {e}")
    return None

def save_data(data, file_name):
    df = pd.DataFrame(data)
    if not os.path.isfile(file_name):
        df.to_csv(file_name, index=False, encoding='utf-8-sig')
    else:
        df.to_csv(file_name, mode='a', index=False, encoding='utf-8-sig',
header=False)
    logging.info("Data saved successfully.")

def main():
    df_links = pd.read_csv('tospitimu_data.csv')
    scraped_links = read_scraped_links('scraping_log.txt')
    property_list = scrape_properties(df_links, scraped_links)
    file_name = f'tospitimu_de-
tails_{datetime.now().strftime("%Y_%m_%d_%H_%M")}.csv'
    save_data(property_list, file_name)
    print("Scraping and saving complete.")

if __name__ == "__main__":
    main()

```

## **spiti\_24.py**

```
### Get the data
from bs4 import BeautifulSoup
import requests
import pandas as pd
import time
import os
from scrapingbee import ScrapingBeeClient
from bs4 import BeautifulSoup
import requests
import pandas as pd
import time
import os
import pyodbc
import pandas as pd
import os
import http.client

# List to store scraped data
data = []

# Define the database connection details
username = 'DESKTOP-P8GGH21\HP'
# Define the database connection string
server = 'localhost'
database = 'real_estate_staging'
conn_str = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATA-
BASE={database};Trusted_Connection=yes;UID={username}'

# url = f"https://www.spiti24.gr/pwliseis/katoikies/pollaples-peri-
oxes/area-ids_101,100?page={pnum}"
proxy = "http://*****:@proxy.zenrows.com:8001"
proxies = {"http": proxy, "https": proxy}

# Loop through the pages
for pnum in range(1, 2):
    try:
```

```

url = f"https://www.spiti24.gr/pwliseis/katoikies/athina-ken-
tro?sortBy=datemodified%7Cdesc&page={pnum}"
print(f"Scraping page: {url}")
response = requests.get(url, proxies=proxies, verify=False)
# Parse the page content
soup = BeautifulSoup(response.content, 'html.parser')
time.sleep(5) # Waits for 5 seconds before executing next line

# Parse the page content
# Wait for the page to load
time.sleep(3)
response = requests.get(url, proxies=proxies, verify=False)
#
    print(response.text)

# Find the listings on the page
listings = soup.find_all('div', class_='property__top')

for listing in listings:

    # Initial data dictionary
    listing_data = {
        "Id": None,
        "link": None,
        "price": None,
        "title": None,
        "region": None,
#         "bedrooms": None,
#         "bathrooms": None,
    }

    link_element = listing.find('a')
    if link_element:
        link = link_element['href'].strip() # Remove extra spaces
using strip()
        number_id = link.split('/')[-1].split('?')[0]
        listing_data["Id"] = number_id
        listing_data["link"] = 'https://www.spiti24.gr/' + num-
ber_id

```

```

        price_element = listing.find('span', class_='property__price')
        symbol_element = listing.find('span', class_='property__price__symbol')

    if price_element and symbol_element:
        price = price_element.get_text(strip=True)
        symbol = symbol_element.get_text(strip=True)
        listing_data["price"] = price + symbol

    title_parts_element = listing.find('ul', class_='property__title__parts')

    if title_parts_element:
        title_element = title_parts_element.find('li', class_=None)
        region_element = title_parts_element.find('li', class_=None).find_next_sibling('li')

        if title_element and region_element:
            title = title_element.get_text(strip=True)
            region = region_element.get_text(strip=True)
            listing_data["title"] = title
            listing_data["region"] = region

    # Find all span tags with the class 'small-tooltip'
    extras_element = listing.find_all('span', {'class': 'small-tooltip'})

    for extra in extras_element:
        if "Bedroom" in extra.get('title', ''):
            listing_data["bedrooms"] = extra.text.strip()
        elif "Bathroom" in extra.get('title', ''):
            listing_data["bathrooms"] = extra.text.strip()

    # Print and add the listing data to the main data list
    print(listing_data)
    print("-----")
    data.append(listing_data)

```

```

        time.sleep(10) # Waits for 10 seconds before executing next line

    except Exception as e:
        print(f"Error occurred: {str(e)}")
        continue # Continue to the next iteration of the loop

# Convert the list of dictionaries to a DataFrame
df = pd.DataFrame(data)

# Generate a timestamp using the current date and time
timestamp = datetime.now().strftime("%Y_%m_%d_%H_%M")

#store the file
if not os.path.isfile('spiti24_data.csv'):
    df.to_csv('spiti24_data.csv', mode='a', index=False, encoding='utf-8-
sig') # Write with header
else:
    df.to_csv('spiti24_data.csv', mode='a', index=False, encoding='utf-8-
sig', header=False) # Append without writing header

# Define the DataFrame containing the scraped data
df = pd.DataFrame(data)

# ### Get more details
import pandas as pd
import requests
from bs4 import BeautifulSoup
import os

# Read the CSV file with links
csv_file_path = 'spiti24_data.csv'
proxy = "http://*****:@js_render=true&anti-
bot=true@proxy.zenrows.com:8001"
proxies = {"http": proxy, "https": proxy}
df = pd.read_csv(csv_file_path)

# Initialize an empty list to store the scraped data

```

```

property_data = []

# Iterate through the links
for index, row in df.iterrows():
    link = row['link']

    # Extract the listing ID from the URL
    listing_id = link.split('/')[-1]

    # Print the URL being scraped
    print(f"Scraping data for Listing ID: {listing_id}")

    # Make a GET request to the webpage
    response = requests.get(link, proxies=proxies, verify=False)

    if response.status_code == 200:
        # Parse the HTML content of the webpage
        soup = BeautifulSoup(response.content, 'html.parser')

        # Extract the image URLs
        image_divs = soup.find_all('div', class_='property_gallery__thumb')
        image_urls = [div.find('a')['href'] for div in image_divs]

        # Store the image URLs in a list
        image_list = []

        for image_url in image_urls:
            image_list.append(image_url)

        # Extract the data you want
        property_title = soup.find('h1').text.strip()
        property_area = soup.find('strong').text.strip()
        property_price = soup.find('span', class_='price').text.strip()

        # Extract the property description or other additional details

```



```
property_description = soup.find('div', class_='property__section__content').text.strip()
```

```
# Extract the additional property details
```

```
ul_element = soup.find('ul', class_='property__extrainfo')
```

```
extra_info = {}
```

```
if ul_element:
```

```
    li_elements = ul_element.find_all('li')
```

```
    for li in li_elements:
```

```
        li_text = li.get_text(strip=True)
```

```
        label, value = li_text.split(':', 1)
```

```
        label = label.strip()
```

```
        value = value.strip()
```

```
        extra_info[label] = value
```

```
# Extract the additional property details table
```

```
additional_details_table = soup.find('table', class_='table')
```

```
additional_details = {}
```

```
if additional_details_table:
```

```
    rows = additional_details_table.find_all('tr')
```

```
    for row in rows:
```

```
        columns = row.find_all(['th', 'td'])
```

```
        if len(columns) == 2:
```

```
            key = columns[0].text.strip()
```

```
            value = columns[1].text.strip()
```

```
            additional_details[key] = value
```

```
# Print the extracted additional property details
```

```
for key, value in additional_details.items():
```

```
    print(f"{key}: {value}")
```

```
# Extract latitude and longitude
```

```
marker_div = soup.find('div', class_='marker')
```

```
latitude = None
```

```
longitude = None
```

```

if marker_div:
    latitude = marker_div.get('data-lat')
    longitude = marker_div.get('data-lng')

# Create a dictionary to store all the data for this property
property_dict = {
    'Id': listing_id,
    'Title': property_title,
    'Area': property_area,
#     'Price': property_price,
    'Description': property_description,
    'images': image_list,
    'Latitude': latitude,
    'Longitude': longitude,
    **extra_info,
    **additional_details # Include additional details
}

# Add the additional details to the property_dict
#property_dict.update(additional_details)
for key, value in additional_details.items():
    property_dict[key] = value
# Append the property data to the list
property_data.append(property_dict)

# Print the details for this property
print("Property Details:")
for key, value in property_dict.items():
    print(f"{key}: {value}")
print("\n" + "="*40 + "\n") # Separate property details with a
line

# Convert the list of dictionaries to a DataFrame
output_df = pd.DataFrame(property_data)

```

```

# Generate a timestamp using the current date and time
timestamp = datetime.now().strftime("%Y_%m_%d_%H_%M")

# Construct the file name with the timestamp
file_name = f'spiti24_details_{timestamp}.csv'

#drop the duplicates if there are any
df = output_df.drop_duplicates(subset='Id', keep='first')

# Define a dictionary to map the old column names to the new ones
column_mapping = {
    'Listing ID': 'Id',
    'Property Title': 'Title',
    'Property Area': 'Area',
    'Property Price': 'Price',
    'Property Description': 'Description',
    'Image URLs': 'images',
    'Latitude': 'latitude',
    'Longitude': 'longitude',
    'Δημοσίευση αγγελίας': 'publication_date',
    'Τελευταία Ενημέρωση': 'last_update',
    'Κωδικός ακινήτου': 'property_code',
    'Κωδικός για μεσίτη': 'agent_code',
#    'Τιμή': 'price',
    'Τιμή ανά τ.μ.': 'price_per_sqm',
    'Περιοχή': 'location',
    'Ζώνη': 'zone',
    'Έτος κατασκευής': 'year_built',
    'Θέρμανση': 'heating',
    'Ενεργειακή κλάση': 'energy_class',
    'Όροφος': 'floor',
    'Επίπεδα': 'levels',
    'Πάρκινγκ': 'parking',
    'Κουζίνες': 'kitchens',
    'Καθιστικά': 'living_rooms',
    'Μπάνια': 'bathrooms',
    'Κατάσταση': 'condition',

```

```

    'Χρήση': 'use',
    'Άλλο': 'other',
    'Έτος ανακαίνισης': 'renovation_year',
    'WC': 'wc',
    'Date available': 'available_date',
    'Μισθωμένο': 'rented',
    'Διεύθυνση': 'address'
}

# Rename the columns using the mapping
df.rename(columns=column_mapping, inplace=True)

# Display the DataFrame with the updated column names
print(df)

# Check if the file exists
if not os.path.isfile(file_name):
    df.to_csv(file_name, mode='a', index=False, encoding='utf-8-sig') #
    Write with header
else:
    df.to_csv(file_name, mode='a', index=False, encoding='utf-8-sig',
    header=False) # Append without writing header
# Print a message when scraping and saving is complete
print("Scraping and saving complete.")

# Perform the join on 'Id'
df = plot_data.merge(df, on='Id', how='inner')
# Print a message when scraping and saving is complete
print("Scraping and saving complete.")

# ### Store in database
from sqlalchemy import create_engine, text
from sqlalchemy import inspect

# Database storage
# Define the database connection details

```

```

username = 'DESKTOP-P8GGH21\HP'
# Define the database connection string
server = 'localhost'
database = 'listings'
driver = 'ODBC Driver 17 for SQL Server'

# Create the SQLAlchemy engine
engine = create_engine(f'mssql+pyodbc://{server}/{database}?driver={driver}&Trusted_Connection=yes')

# Create an inspector
inspector = inspect(engine)

# Check if the table exists
if "spiti_24" in inspector.get_table_names():
    # If it does, then get the existing IDs
    existing_ids_df = pd.read_sql_query("SELECT listing_id FROM spiti_24",
engine)
else:
    print("Table does not exist")

# If 'Id' is a list, join the list into a string
if isinstance(df['Id'].iloc[0], list):
    df['Id'] = df['Id'].apply(lambda x: ','.join(map(str, x)))

# # Convert 'Id' column in both DataFrames to set for comparison
# existing_ids = set(existing_ids_df['listing_id'])
new_ids = set(df['Id'])

# Get the unique Ids that are not in the SQL database
unique_ids = new_ids - existing_ids

# Filter the DataFrame to include only rows with unique Ids
details_df_unique = df[df['Id'].isin(unique_ids)]

# Append the DataFrame to the SQL table
details_df_unique.to_sql('spiti_24', con=engine, if_exists='append',
index=False)

```

```
else:
    df.to_sql('spiti_24', con=engine, if_exists='append', index=False)

# Close the engine
engine.dispose()
```