CM3070-final-report

September 9, 2024

Deep Learning for Personalised Property Recommendations System: Data Collection and Model Development Using Public Datasets

Guillermo Olmos Ranalli

September 9, 2024

GitHub Repository:
https://github.com/georn/final-project-uk-real-estate-recommendation-system

Contents

GitHub Repository:
https://github.com/georn/final-project-uk-real-estate-recommendation-system

1 Introduction

1.1 Project Concept

The Personalised Property Recommendation System aims to assist potential homebuyers in the UK real estate market by providing tailored property recommendations based on individual preferences and financial situations. By integrating historical transaction data from HM Land Registry with current property listings from OnTheMarket, the system delivers customised property suggestions. This project involves developing and evaluating various deep learning models to determine the most effective approach for property recommendation, framed as a classification task.

1.2 Motivation

Navigating the real estate market can be particularly challenging for new generations, including Millennials, Gen Z, and Gen Alpha, who face unique financial and lifestyle constraints. First-time buyers often struggle to find properties that meet their specific criteria within their budget. The abundance of property options and the complexity of property features necessitate a tool that can streamline the search process and offer personalised recommendations. This project seeks to address this need by developing a robust deep learning-based recommendation system.

1.3 Project Template

This project, based on "Project Idea Title 1: Deep Learning on a Public Dataset" from CM3015, uses publicly accessible real estate data from HM Land Registry and OnTheMarket to develop and compare deep learning models for personalised property recommendations. Following the methodology in "Deep Learning with Python" by F. Chollet, the prototype aims to improve model performance.

This project builds upon my previous work in CM2015, where I developed data collection scripts and utilised similar datasets. The current project expands this foundation by incorporating advanced machine learning for better recommendations.

1.4 Scope and Limitations

This prototype analyses various types of residential properties in Buckinghamshire, Bedfordshire, Oxfordshire, Northamptonshire, Hertfordshire, and Berkshire, excluding shared ownership, retirement homes, new builds, auction listings, farms/land, park homes, and properties over £900,000. These filters were applied to data collected from OnTheMarket.com.

The prototype has limitations, including reliance on online data accuracy and availability. It does not cover commercial real estate or the rental market in Buckinghamshire, Bedfordshire, Oxfordshire, Northamptonshire, Hertfordshire, and Berkshire.

1.5 Data Sources and Selection

1.5.1 Data Requirements

The primary goal of this prototype is to validate the feasibility of using machine learning techniques to predict property prices and recommend properties across multiple shires in the UK. To achieve this, we utilized two main data sources:

- **HM Land Registry Data**: This dataset provides comprehensive information on property sales across multiple counties, including Buckinghamshire, Bedford, Oxfordshire, North Northamptonshire, West Northamptonshire, Hertfordshire, and West Berkshire. It includes crucial details such as sale prices, transaction dates, and property types.
- **Web Scraped Data from OnTheMarket.com**:This dataset comprises current property listings from multiple shires, including Buckinghamshire, Bedfordshire, Oxfordshire, Northamptonshire, Hertfordshire, and Berkshire. It includes asking prices, property types, and other relevant details, providing a real-time perspective on the market.
- **Synthetic User Profiles**: To complement the property data and enable more robust testing of our recommendation system, we generated synthetic user profiles. These profiles simulate various user preferences, financial situations, and property requirements, allowing us to test our model's performance across a diverse range of scenarios.

To comprehensively analyse Buckinghamshire's property market, data from diverse sources were gathered:

- **HM Land Registry Data**: This dataset provides comprehensive information on property sales in Buckinghamshire, including sale prices and transaction dates.
- **Real Estate Listings**: Scraped data from OnTheMarket.com helps in understanding the ongoing trends and fluctuations in property prices and demands within Buckinghamshire. Python scripts were developed, organised in the `src` folder of this prototype, to efficiently collect and process the data, ensuring a robust and reliable dataset for analysis.

1.5.2 Choice of Data Sources

- **HM Land Registry Data**: Chosen for its reliability and comprehensive coverage of actual property sales across multiple UK counties. This data is accessed via gov.uk under the Open Government Licence v3.0.
- **OnTheMarket.com**: Selected as a current and active source for property listings across multiple shires, offering a real-time perspective on the market. Data was scraped in compliance with the site's terms and conditions.
- **Synthetic User Profiles**: Generated to provide a diverse set of user data for testing the recommendation system. This approach allows us to evaluate the model's performance across a wide range of user preferences and financial situations without compromising real individuals' privacy.

1.5.3 Methodology for Data Collection and Processing

- HM Land Registry data was downloaded in CSV format, covering transactions for the year 2023 across the specified counties.
- Web scraping was conducted on OnTheMarket.com using Python scripts, focusing on gathering current listings across multiple shires. The scraping process adhered to the website's robots.txt file and was conducted using a unique user agent.

- Synthetic user profiles were generated using a custom algorithm that creates realistic user preferences based on demographic data and real estate market trends.

1.5.4 Limitations and Constraints

- **Timeframe**: The HM Land Registry data covers only sales within 2023, and the scraped data reflects listings at the time of scraping. This temporal limitation means the analysis might not fully capture long-term market trends.
- **Geographical Scope**: The focus on Buckinghamshire alone may not provide a complete picture of broader regional or national property market trends.
- **Data Completeness**: While the Land Registry data is comprehensive for sales, the scraped data from OnTheMarket.com might not capture every property listing in the region, leading to potential gaps in the dataset.
- **Synthetic Data Limitations**: While synthetic user profiles allow for diverse testing scenarios, they may not perfectly reflect real-world user behaviors and preferences.

1.6 Ethical Considerations

1.6.1 Data Sources and Permissions

HM Land Registry Data - HM Land Registry data is used under the Open Government Licence v3.0. - Proper attribution has been given as per the OGL requirements: "Contains HM Land Registry data © Crown copyright and database right 2021. This data is licensed under the Open Government Licence v3.0."

OnTheMarket.com Data - OnTheMarket.com data was collected via web scraping, strictly adhering to their robots.txt file, using a unique user agent with contact information, and employing rate limiting to respect their servers.

This prototype focuses on objective real estate data, avoiding personal judgements or assumptions. It aims to maintain neutrality, preventing negative impacts like market manipulation. Any personal data has been anonymized to protect privacy.

Chapter 1 Word Count: 993

2 Chapter 2: A Literature Review

2.1 Introduction

In the realm of real estate, accurate property price prediction and personalised recommendation systems are crucial for assisting potential homebuyers and real estate professionals. The intersection of machine learning and real estate has gained substantial attention, with various methodologies explored to enhance prediction accuracy and personalisation. This literature review critically examines existing research on housing price prediction models and personalised recommendation systems, highlighting their methodologies, strengths, and limitations.

2.2 Housing Price Prediction Models

2.2.1 Hedonic-Based Regression Approaches

Historically, hedonic-based regression models have been utilised to determine the impact of various housing attributes on property prices. These models estimate prices based on factors such as location, size, and age of the property. Despite their widespread use, hedonic models face limitations such as difficulty in capturing nonlinear relationships and the need for extensive data preprocessing to handle heteroscedasticity and multicollinearity issues [1].

2.2.2 Machine Learning Techniques

The advent of machine learning has provided more sophisticated tools for housing price prediction, capable of handling complex, non-linear relationships between variables. The study by Park and Bae (2020) investigates the application of several machine learning algorithms to predict housing prices using data from Fairfax County, Virginia. The algorithms examined include C4.5, RIPPER, Naïve Bayesian, and AdaBoost [5].

- **C4.5 Algorithm**: This algorithm is an extension of the earlier ID3 algorithm and generates a decision tree used for classification purposes. In the context of housing price prediction, the decision tree helps identify the most significant variables influencing prices [6].
- **RIPPER Algorithm**: This is a rule-based learning algorithm that generates a set of rules to classify data. According to Park and Bae (2020), the RIPPER algorithm demonstrated superior performance in terms of accuracy compared to other models tested in their study [5].
- **Naïve Bayesian**: This probabilistic classifier is based on Bayes' theorem and assumes independence between predictors. Despite its simplicity, it can be effective for certain types of classification problems [2].
- **AdaBoost**: This ensemble method combines multiple weak classifiers to create a strong classifier. It adjusts the weights of misclassified instances, thereby improving the model's accuracy over successive iterations [3].

Park and Bae's study concludes that the RIPPER algorithm consistently outperformed the other models in terms of classification accuracy for housing price prediction. This finding is significant as it highlights the potential of rule-based algorithms in capturing the complexities of housing market data [5].

2.3 Content-Based Recommender Systems

Content-based recommender systems are crucial for providing personalised suggestions based on user preferences and item attributes. Lops, Gemmis, and Semeraro (2011) provide a comprehensive overview of the state-of-the-art techniques and trends in content-based recommendation systems [4].

- **Feature Extraction**: Content-based systems rely heavily on extracting meaningful features from items. In the context of real estate, features such as property type, location, price, and amenities are essential.
- **Similarity Calculation**: These systems calculate the similarity between items based on their features. For real estate, properties with similar attributes (e.g., location, price range)

are considered similar and thus recommended to users with matching preferences.

- **User Profiles**: Content-based systems maintain profiles for users, capturing their preferences and interaction history. This allows the system to tailor recommendations based on individual user needs.

Lops et al. (2011) highlight the challenges in content-based recommender systems, such as the cold start problem, where new users or items lack sufficient data for effective recommendations. However, integrating advanced machine learning techniques can mitigate some of these issues by improving feature extraction and similarity calculations [4].

2.4 Discussion

The research conducted by Park and Bae (2020) underscores the importance of selecting appropriate machine learning algorithms for housing price prediction. Their comparative analysis provides valuable insights into the strengths and weaknesses of different models. For instance, while ensemble methods like AdaBoost are generally robust, rule-based algorithms such as RIPPER can offer higher accuracy for specific datasets [5].

This study also emphasizes the need for comprehensive data preprocessing, including the selection of relevant features and handling missing values, to enhance the predictive performance of machine learning models. Additionally, the integration of various algorithms can potentially lead to the development of a hybrid model that leverages the strengths of each approach [5].

2.5 Conclusion

The literature on housing price prediction demonstrates that machine learning techniques, particularly rule-based algorithms like RIPPER, can significantly improve the accuracy of price predictions. The study by Park and Bae (2020) serves as a critical reference point for developing advanced models that can aid real estate stakeholders in making informed decisions [5].

Further research should focus on integrating these models with personalised recommendation systems to provide comprehensive solutions for real estate buyers and sellers. By leveraging machine learning's capabilities, the real estate industry can enhance its analytical tools, leading to more accurate and reliable property valuations.

Chapter 2 Word Count: 775

3 Chapter 3: A Design

3.1 Project Overview

The Personalised Property Recommendation System aims to integrate historical transaction data from HM Land Registry and current property listings from OnTheMarket to provide tailored property recommendations based on user preferences and financial situations. The project covers multiple shires including Buckinghamshire, Bedfordshire, Oxfordshire, Northamptonshire, Hertfordshire, and Berkshire. It follows "Project Idea Title 1: Deep Learning on a Public Dataset" and aims to find the most effective model for property recommendation using deep learning techniques.

3.2 Domain and Users

3.2.1 Domain

The project is situated in the real estate domain, focusing on residential properties in the UK. It leverages publicly available datasets to build a recommendation system that aids potential home-buyers in making informed decisions.

3.2.2 Users

The primary users of the system are:

- **First-time homebuyers**: Individuals looking for their first property purchase who need tailored recommendations based on their budget and preferences.
- **Real estate agents**: Professionals who can use the system to provide clients with data-driven property suggestions.
- **Property investors**: Individuals or companies looking to invest in real estate who require accurate property price predictions and recommendations.

3.3 Justification of Design Choices

3.3.1 User Needs

The design choices are informed by the needs of users in the real estate market:

- **Personalization**: Users require personalised property recommendations that match their financial constraints and preferences.
- **Accurate Predictions**: Accurate property price predictions help users make informed decisions.
- **Usability**: The system must be easy to use and provide quick, relevant recommendations.

3.3.2 Domain Requirements

The real estate domain requires:

- **Integration of Diverse Data Sources**: Combining historical transaction data with current listings to provide a comprehensive view.
- **Handling Non-linear Relationships**: Using advanced machine learning models to capture complex patterns in the data.

3.4 Project Structure

3.4.1 Data Collection and Preprocessing

- **Data Sources**: Historical transaction data from HM Land Registry and current property listings from OnTheMarket.
- **Data Collector Service**: Scripts to collect real-time data from OnTheMarket.
- **Data Cleaner Service**: Process and clean HM Land Registry data.
- **Data Standardiser Service**: Integrate and standardize data from both sources.
- **Synthetic User Generator**: Create synthetic user profiles for testing and development.

3.4.2 Model Development

- **Data Loader**: Retrieve and prepare data for model training.
- **Model Builder**: Construct the neural network architecture.
- **Model Trainer**: Train the model on the prepared dataset.
- **Model Evaluator**: Assess the model's performance and generate evaluation metrics.

3.4.3 Recommendation System

- **Web Application**: Flask-based web server to host the user interface and handle requests.
- **Keras Model:**: The trained neural network model for generating property recommendations.
- **Data Scalers**: Property and User scalers to normalize input data.

3.5 Technologies and Methods

3.5.1 Technologies

- **Python**: Primary programming language for data processing and model development.
- **TensorFlow and Keras**: Libraries for building and training deep learning models.
- **Pandas and NumPy**: Libraries for data manipulation and analysis.
- **Scikit-Learn**: Library for implementing various machine learning algorithms and evaluation metrics.
- **BeautifulSoup and Requests**: Libraries for web scraping.
- **Matplotlib**: Library for data visualization.
- **Flask**: A lightweight Python web framework for building the backend of the web application.
- **Docker**: For containerization and easy deployment of the entire system.
- **PostgreSQL**: A relational database management system for storing and managing the application's data.
- **pytest**: Framework for testing Python code to ensure functionality and correctness.
- **HTML, CSS, and JavaScript**: For developing the frontend of the web application.

3.5.2 Neural Network Structure

The neural network model consists of two main branches: a property input branch and a user input branch, which are then combined for the final prediction. Key features include:

- Multiple dense layers with ReLU activation
- Batch Normalization and Dropout for regularization
- L2 regularization and He Normal initialization
- Adam optimizer with gradient clipping
- Binary Cross-Entropy loss function

The model is designed to process property and user features separately before combining them, with various regularization techniques to prevent overfitting and improve generalization.

3.5.3 Methods

- **Data Preprocessing**: Cleaning and integrating data from multiple sources.
- **Machine Learning**: Developing and comparing different machine learning models to identify the most effective one for price prediction.

- **Content-Based Filtering**: Creating a recommendation system based on the features of the properties and user preferences.
- **Evaluation Metrics**: Using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to evaluate model performance.

3.6 Work Plan

3.6.1 Major Tasks and Timeline

- **Data Collection and Preprocessing (Weeks 1-4)**
  - Collect data from HM Land Registry and OnTheMarket
  - Clean and preprocess data
  - Integrate datasets
- **Model Development (Weeks 5-10)**
  - Feature selection
  - Develop and train machine learning models
  - Evaluate models using MAE and RMSE
- **Recommendation System Development (Weeks 11-14)**
  - Develop content-based filtering system
  - Integrate price prediction with user preferences
- **Web Interface Development (Week 11)**
  - Develop a user-friendly web interface for the recommendation system
- **Docker Setup and Integration (Week 12)**
  - Set up Docker to orchestrate the entire system
- **Experiment with Additional Models (Weeks 13-14)**
  - Implement and compare models like Gradient Boosting, Random Forests, and advanced deep learning architectures
  - Optimize hyperparameters for each model and compare their performance using various metrics
- **Expand Features and Incorporate New Parameters (Week 15)**
  - Collect and preprocess additional data for new features (e.g., socioeconomic and environmental factors)
  - Perform feature engineering to create new features from the existing data
  - Integrate these features into the model and evaluate their impact on performance
- **Geographical Expansion (Week 16)**
  - Collect data for regions beyond Buckinghamshire
  - Preprocess and integrate this data into the existing dataset
  - Evaluate the model's performance on the expanded dataset
- **User-Centric Testing and Feedback Collection (Week 17)**
  - Design a user-friendly interface for inputting preferences and receiving recommendations
  - Simulate user interactions using synthetic data
  - Collect feedback through synthetic data to refine the system
- **Replicate and Compare with High-Quality Models (Week 18)**
  - Replicate models from high-quality published papers
  - Compare their performance with your models and analyze the differences
- **Final Model Tuning and Evaluation (Week 19)**
  - Fine-tune the best-performing models
  - Conduct a thorough error analysis and identify areas for further improvement

  - Finalize the model and prepare it for deployment
- **Report Writing & Finalization (Week 20)**
  - Document all findings, methodologies, and results
  - Ensure the report is well-structured, with clear explanations and justifications for each step
  - Prepare for submission, ensuring all requirements are met

Week	Task
1-4	Data Collection & Preprocessing
5-10	Model Development
11	Web Interface Development
12	Docker Setup and Integration
13-14	Experiment with Additional Models
15	Expand Features and Incorporate New Parameters
16	Geographical Expansion (Collect Data for New Areas)
17	User-Centric Testing and Feedback Collection
18	Replicate and Compare with High-Quality Models
19	Final Model Tuning and Evaluation
20	Report Writing & Finalization

3.7 Project Management and Version Control

To ensure efficient project management and maintain a structured development process, this project utilized GitHub Projects in conjunction with a branch-and-merge workflow. This approach allowed for better organization, tracking of tasks, and collaboration throughout the development lifecycle.

3.7.1 GitHub Projects

A GitHub Project board was set up to visualize and manage the workflow, essentially functioning as a digital Gantt chart. The project can be viewed at: https://github.com/users/georn/projects/4

Also the Github repository can be found at:

GitHub Repository:
`https://github.com/georn/final-project-uk-real-estate-recommendation-system`

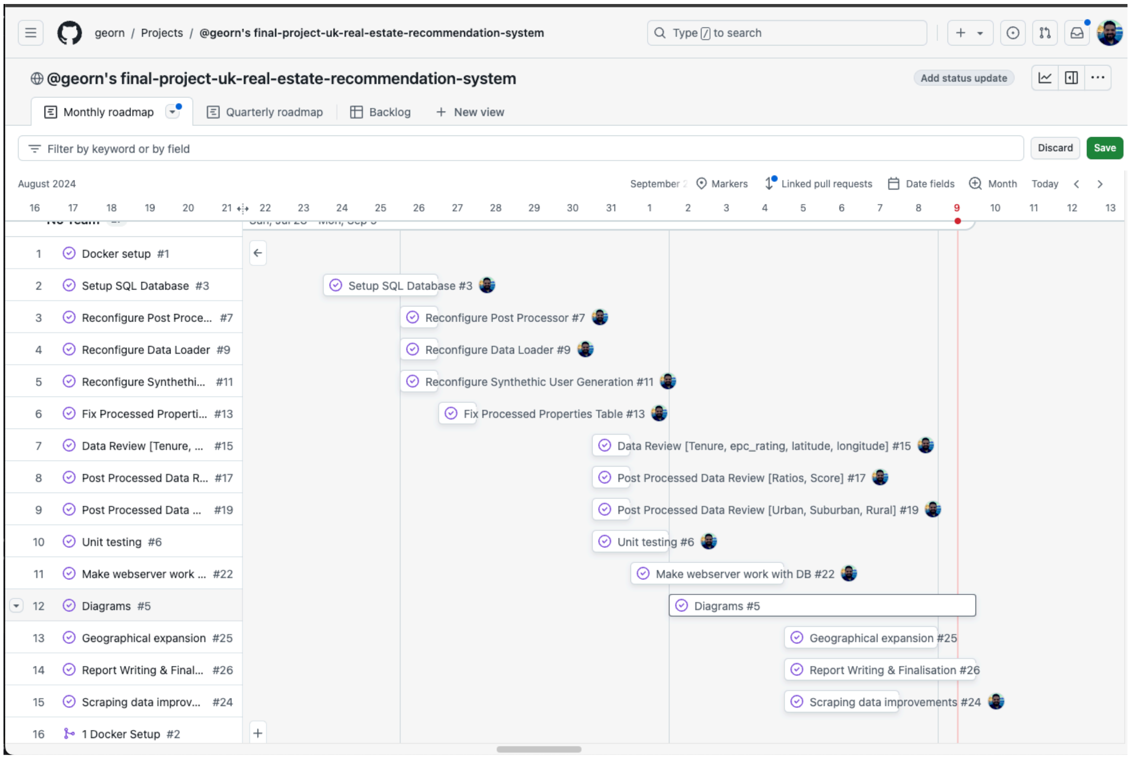


Figure 3.1: Github Project

This board included the following elements:

- **Task Cards**: Each major task was represented as a card on the board, allowing for easy tracking of progress.
- **Timelines**: Tasks were arranged on a timeline, providing a clear overview of the project schedule and deadlines.
- **Assignments**: Team members (in this case, myself) were assigned to specific tasks, ensuring accountability.
- **Status Updates**: The status of each task (e.g., To Do, In Progress, Done) was regularly updated to reflect current progress.

3.7.2 Branch-and-Merge Workflow

In parallel with the GitHub Project board, a branch-and-merge workflow was implemented to manage code changes effectively:

- **Feature Branches**: For each major feature or task, a new branch was created from the main development branch.
- **Local Development**: Work was carried out on these feature branches, allowing for isolated development and testing.
- **Regular Commits**: Changes were committed frequently with descriptive commit messages to maintain a clear history of modifications.
- **Pull Requests**: Once a feature was complete, a pull request was created to merge the changes back into the main branch.
- **Code Review**: Pull requests were reviewed to ensure code quality and consistency before merging. Merging: After approval, feature branches were merged into the main branch, and

the corresponding task on the GitHub Project board was updated.

This approach provided several benefits:

- **Traceability**: Each feature or major change could be traced back to its corresponding task and branch.
- **Version Control**: The project's history was well-documented, allowing for easy rollbacks if needed.
- **Parallel Development**: Multiple features could be developed simultaneously without conflicts.
- **Quality Assurance**: The code review process helped maintain code quality and consistency throughout the project.

3.8 Testing and Evaluation Plan

3.8.1 Model Evaluation

- **Unit Testing**: Test individual components (e.g., data collection scripts, model training functions) to ensure they work as expected. The unit testing coverage is documented with a visual report, showcasing the percentage of code covered by tests. For more details, refer to **Appendix E**, where the coverage report is provided.
- **Comprehensive Metrics**: Evaluate the model using accuracy, precision, recall, and F1 score to provide a holistic view of performance.
- **Confusion Matrix Analysis**: Analyze true positives, true negatives, false positives, and false negatives to understand the model's strengths and weaknesses
- **Learning Curve Analysis**: Monitor training and validation metrics over epochs to ensure proper convergence and absence of overfitting
- **Feature Importance Analysis**: Utilize techniques to determine the most influential features in the model's decision-making process.

3.8.2 Error Analysis

- **Misclassification Analysis**: Conduct in-depth analysis of misclassified samples to identify patterns and potential areas for improvement.
- **Confidence Analysis**: Examine the model's confidence in its predictions, particularly for misclassified samples.

3.8.3 Validation Strategies

- **Cross-Validation**: Implement k-fold cross-validation to ensure the model's robustness across different subsets of the data.
- **Synthetic Data Considerations**: Acknowledge the use of synthetic data and plan for potential challenges in transitioning to real-world data.

By following this structured approach and incorporating these components, the project aims to deliver a robust and effective personalised property recommendation system that meets user needs and leverages advanced machine learning techniques.

3.9 System Architecture Overview

The UK Real Estate Recommendation System is designed with a modular and scalable architecture to efficiently collect, process, and analyze property data for personalised recommendations. The system architecture, as illustrated in Figure 4.1, consists of several key components:

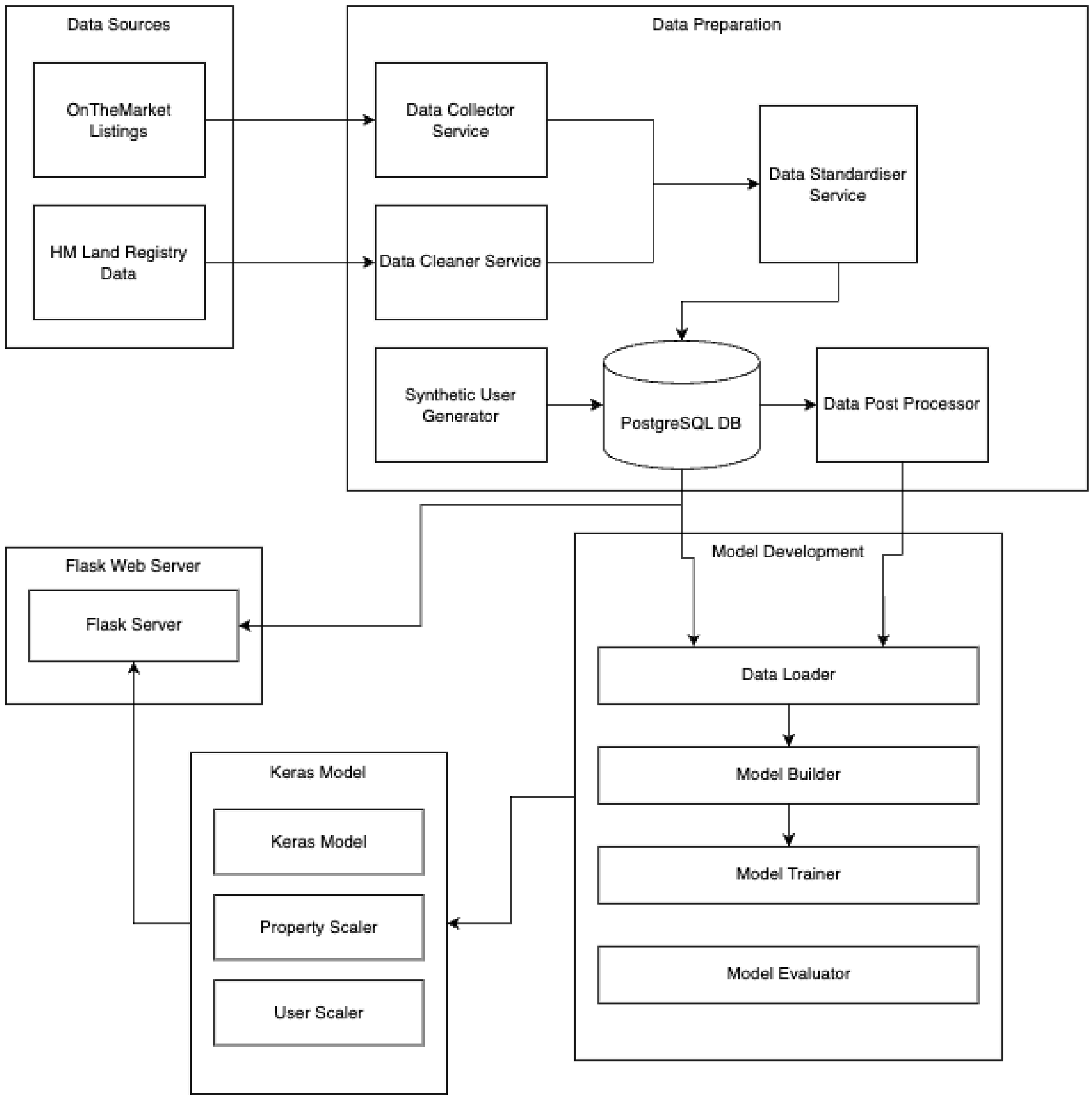


Figure 4.1: System Architecture Diagram

3.9.1 Data Sources

1. **OnTheMarket Listings:** Current property listings scraped from the OnTheMarket website, covering multiple shires including Buckinghamshire, Bedfordshire, Oxfordshire, Northamptonshire, Hertfordshire, and Berkshire.
2. **HM Land Registry Data:** Historical property transaction data from the UK government, providing comprehensive information for properties across multiple counties, including Buckinghamshire, Bedford, Oxfordshire, North Northamptonshire, West Northamptonshire,

Hertfordshire, and West Berkshire. This data includes sale prices, property types, and locations.

3.9.2 Data Preparation

1. **Data Collector Service:** Web scrapes current property listings from OnTheMarket.com for multiple shires, using price segmentation to ensure comprehensive coverage.
2. **Data Cleaner Service:** Processes and cleans the HM Land Registry data, filtering for specific shires and years, and standardizing county names.
3. **Data Standardiser Service:** Integrates and standardizes data from both sources.
4. **Synthetic User Generator:** Creates synthetic user profiles for testing and development purposes.

3.9.3 Data Storage

1. **PostgreSQL Database:** Centralized storage for all processed and standardized data.

3.9.4 Data Processing

1. **Data Post Processor:** Performs additional data transformations and feature engineering.

3.9.5 Model Development

1. **Data Loader:** Retrieves and prepares data for model training.
2. **Model Builder:** Constructs the neural network architecture.
3. **Model Trainer:** Trains the model on the prepared dataset.
4. **Model Evaluator:** Assesses the model's performance and generates evaluation metrics.

3.9.6 Web Application

1. **Flask Web Server:** Hosts the user interface and handles user requests.
2. **Keras Model:** The trained neural network model for generating property recommendations.
3. **Property Scaler and User Scaler:** Normalize input data for consistent model predictions.

This architecture ensures a streamlined flow of data from collection to recommendation, with each component designed to handle specific tasks in the pipeline. The modular design allows for easy maintenance, updates, and scalability of individual components without affecting the entire system.

Chapter 3 Word Count: 1955

4 Chapter 4: Implementation

4.1 Data Collection and Data Cleaning

4.1.1 Web Scraping Methodology

Our web scraping solution collects current property listings from OnTheMarket.com across multiple shires, complementing HM Land Registry data. The process comprises four main components:

1. `robot_check.py`: Ensures `robots.txt` compliance.
2. `crawler.py`: Collects property URLs using requests and BeautifulSoup with rate limiting.

3. `scraper.py`: Extracts specific property data from web pages.
4. `data_collector_service.py`: Orchestrates the scraping process:
   - Collects data for multiple shires (Buckinghamshire, Bedfordshire, Oxfordshire, Northamptonshire, Hertfordshire, Berkshire)
   - Implements price segmentation for comprehensive coverage
   - Saves data incrementally per shire

Ethical Considerations We adhered to ethical standards: respecting `robots.txt`, using a unique user agent, implementing rate limiting, and ensuring non-disruptive interaction with OnTheMarket.com.

4.1.2 Cleaning and Preparing Data

Data Cleaning Methodology Our pandas-based methodology filters, standardizes, and handles HM Land Registry data for multiple UK counties, allowing comprehensive property market analysis.

Implementing Data Cleaning with `data_cleanser_service.py` Key Features: - Processes multi-county data - Assigns column headers - Loads and transforms CSV data - Standardizes county names - Converts and filters transaction dates - Provides summary statistics per county

Source Data The script processes `pp-2023.csv`, containing 2023 property transactions. Due to its size (>30MB), it's not included in the GitHub repository but can be downloaded from:

https://www.gov.uk/government/statistical-data-sets/price-paid-data-downloads

4.2 Data Exploration

To facilitate this exploration, we developed several Python scripts that allow us to examine different aspects of our data. These scripts provide valuable insights into the structure, content, and statistical properties of both our historical property data and the current listing data.

By referring to Appendix B, readers can gain a comprehensive understanding of our data exploration process.

4.3 Standardisation and Preprocessing

The data cleaning and standardisation process is a crucial step in our pipeline, ensuring that data from different sources is consistent and ready for analysis. This process is handled by several specialized modules within the `data_standardiser` package.

4.3.1 Data Standardiser Architecture

The `data_standardiser` package is structured as follows:

- `address_utils.py`: Handles address normalization and cleaning.
- `constants.py`: Stores constant values used across the standardization process.
- `county_mapping.py`: Manages standardization of county names.
- `data_processing.py`: Contains core data processing and standardization functions.
- `database_operations.py`: Manages database interactions for data storage and retrieval.
- `geocoding.py`: Handles the geocoding process to add latitude and longitude data.

- `logging_config.py`: Configures logging for the standardization process.
- `main.py`: Orchestrates the entire standardization process.
- `property_utils.py`: Provides utility functions for property-specific data processing.
- `utils.py`: Contains general utility functions used across the package.

4.3.2 Key Standardisation Processes

1. **Address Normalization** The `address_utils.py` module provides functions to clean and normalize addresses, ensuring consistency across different data sources. This includes removing special characters, standardizing formatting, and handling common address variations.

2. **County Standardisation** `county_mapping.py` is responsible for standardizing county names. This is crucial when dealing with data from multiple sources that might use different naming conventions for the same counties.

3. **Data Processing and Feature Extraction** `data_processing.py` contains core functions for processing raw data. This includes:

   - Standardizing price formats
   - Extracting and standardizing property features (e.g., number of bedrooms, bathrooms)
   - Standardizing property types
   - Handling date conversions

4. **Geocoding** The `geocoding.py` module implements a robust geocoding process using both Nominatim and ArcGIS services. It includes features like:

   - Caching to avoid redundant API calls
   - Error handling and retries
   - Fallback mechanisms when one service fails

5. **Database Operations** `database_operations.py` manages the interaction between the standardized data and the database. It handles:

   - Inserting and updating historical property data
   - Processing and inserting listing data
   - Merging data from different sources in the database

4.3.3 Standardisation Workflow

The standardization process, orchestrated by `src/data_standardiser/main.py`, follows these general steps:

1. Load raw data from both HM Land Registry and scraped listings.
2. Clean and standardize addresses and county names.
3. Process and standardize property details (prices, types, features).
4. Perform geocoding to add latitude and longitude data.
5. Merge data from different sources, resolving conflicts and duplicates.
6. Store the standardized data in the database for further analysis.

4.4 Feature Engineering

Our feature engineering process, implemented in `data_post_processor.py` and `location_classifier.py`, enhances the model's ability to capture relevant patterns:

4.4.1 Key Features Created

1. Temporal Features: year, month, day_of_week, days_since_date, listing_recency
2. Affordability Metrics: price_to_income_ratio, price_to_savings_ratio, affordability_score
3. Location-based Features: Urban/Suburban/Rural classification, county-specific features, price_relative_to_county_avg
4. Property Characteristics: size_sq_ft, binary amenity features, EPC rating encoding, property type encoding, bedroom/bathroom counts

4.4.2 Data Processing Pipeline (data_post_processor.py)

1. Handle Missing Values: Median imputation for numeric, mode for categorical, multi-step process for size_sq_ft
2. Feature Engineering: Create all features mentioned above
3. Encode Categorical Variables: One-hot encoding for property types and counties, custom encoding for tenure and EPC ratings
4. Feature Scaling: StandardScaler for selected numerical features
5. Data Integration: Merge property features with synthetic user data

This pipeline prepares our data for model building, capturing complex relationships in the property market while accounting for user preferences and affordability metrics.

4.5 Model Architecture and Implementation

4.5.1 Data Loading and Preprocessing

Before building and training the model, we perform additional data processing steps specific to the model requirements. This process is handled by data_loader.py and data_preprocessing.py in the src/model/ directory:

1. **Data Loading**: data_loader.py retrieves the processed property data and synthetic user data from the database.

2. **Creating Property-User Pairs**: We generate pairs of properties and synthetic users for training using the create_property_user_pairs function. This function ensures that properties are matched with users based on tenure preferences.

3. **Feature Preparation**: The prepare_features function in data_preprocessing.py handles:

   - Ensuring all expected features are present
   - Handling missing values (NaN) for both numeric and categorical features
   - Converting data types and scaling certain features
   - Creating additional features like log transformations of price and size

4. **Target Variable Creation**: A boolean target variable is created using the create_target_variable function, based on multiple conditions including affordability, bedroom requirements, price-to-income ratio, size requirements, and tenure preferences.

5. **Data Scaling**: StandardScaler is applied separately to property and user features to ensure all features are on the same scale.

6. **Data Splitting**: The data is split into training and testing sets using a 80-20 split, stratified by the target variable.

4.5.2 Model Approach

Our Personalised Property Recommendation System employs a hybrid, neural network-based approach that combines elements of content-based filtering with deep learning techniques. This approach allows us to capture complex, non-linear relationships between property and user features.

The system treats property recommendation as a binary classification problem, predicting whether a given property-user pair is a good match based on historical data and engineered features.

Key aspects of this approach include:

1. **Feature-rich inputs**: Utilizing a wide range of property features and user characteristics.
2. **Learned feature interactions**: The neural network learns to identify and weigh complex interactions between features, going beyond predefined similarity metrics.
3. **Personalization**: Direct incorporation of user features enables highly personalised recommendations.
4. **Scalability**: The approach can easily incorporate new features and scale to large datasets, suitable for the dynamic real estate market.
5. **Interpretability challenges**: While powerful, the neural network's decision-making process is less transparent than simpler methods, potentially requiring additional explanation techniques.

This hybrid approach allows us to capture intricate patterns in property-user matches that might be missed by simpler recommendation techniques, potentially leading to more accurate and nuanced property recommendations.

Feature Selection The model utilizes a comprehensive set of features derived from our preprocessed dataset:

1. Property Features:
    - Price and log-transformed price
    - Size (in square feet) and log-transformed size
    - Location (encoded as Urban, Suburban, Rural)
    - Property Type (one-hot encoded)
    - Binary features (e.g., 'has_garden', 'has_parking')
    - Temporal features (year, month, day of week)
    - EPC rating (encoded)
    - Number of bedrooms and bathrooms
    - Tenure
    - Price relative to county average
    - County-specific features (one-hot encoded)
2. User Features:
    - Income
    - Savings
    - Maximum commute time

- Family size
- Tenure preference

3. Engineered Features:
   - Price-to-income ratio
   - Price-to-savings ratio
   - Affordability score

This comprehensive feature set allows our model to capture a wide range of factors that influence property recommendations, providing a solid foundation for the neural network to learn complex patterns in property-user matches.

4.5.3 Neural Network Structure

The neural network model, implemented in `model_builder.py`, consists of the following components:

1. **Property Input Branch**:
   - Input layer for property features
   - Dense layer with 256 units, ReLU activation
   - Batch Normalization
   - Dropout (30%)
   - Dense layer with 128 units, ReLU activation
   - Batch Normalization
   - Dropout (30%)
2. **User Input Branch**:
   - Input layer for user features
   - Dense layer with 32 units, ReLU activation
   - Batch Normalization
   - Dropout (30%)
   - Dense layer with 16 units, ReLU activation
   - Batch Normalization
   - Dropout (30%)
3. **Combined Layers**:
   - Concatenation of property and user branches
   - Dense layer with 64 units, ReLU activation
   - Batch Normalization
   - Dropout (30%)
   - Dense layer with 32 units, ReLU activation
   - Batch Normalization
   - Dropout (30%)
4. **Output Layer**:
   - Dense layer with 1 unit, Sigmoid activation (for binary classification)

Model Compilation The model is compiled with the following settings: - **Optimizer**: Adam with learning rate of 0.0005 and gradient clipping (clipnorm=1.0) - **Loss Function**: Binary Cross-Entropy - **Metrics**: Accuracy

Regularization Techniques To prevent overfitting and improve generalization, the following techniques are employed: - L2 regularization (weight decay) with factor 0.01 on all dense layers -

Dropout layers (30% rate) after each hidden layer - Batch Normalization after each hidden layer - He Normal initialization for weight matrices

This architecture is designed to process property and user features separately before combining them for the final prediction. The regularization techniques aim to prevent overfitting and improve the model's ability to generalize to unseen data. The complete model comprises 55,825 total parameters, of which 54,769 are trainable. For a detailed summary of the model's layers and parameters, please refer to Appendix C.

4.5.4 Model Training Process

The model training process, implemented in `model_trainer.py`, involves:

1. **Initialization**: The model is built using the architecture defined in `model_builder.py`.
2. **Training Configuration**:
   - Epochs: 200 (maximum)
   - Batch Size: 32
   - Early Stopping: Monitors validation loss with a patience of 10 epochs and restores best weights
3. **Training Loop**: The model is trained using `model.fit()` with the prepared training data and validation data.
4. **Monitoring**: Training progress is monitored and logged, including loss and accuracy for both training and validation sets.
5. **Visualization**: After training, the `plot_training_history` function visualizes the training process by plotting:
   - Training and validation loss over epochs (using a logarithmic scale)
   - Training and validation accuracy over epochs

This training process is designed to be reproducible and efficient, with mechanisms in place to prevent overfitting (early stopping), handle potential numerical instabilities (NaN detection), and provide comprehensive monitoring of the model's performance throughout training.

For the detailed implementation of the training process, including the `train_model` function and the code used to execute the training and visualize the results, please refer to Appendix D.

4.5.5 Model Evaluation

The model's performance is assessed using multiple methods in the `evaluate_model` function (`model_evaluator.py`):

1. **Prediction**: Generate predictions on the test set.
2. **Performance Metrics**:
   - Accuracy, Precision, Recall, F1 Score
3. **Confusion Matrix**: Visual representation of model performance.
4. **Classification Report**: Detailed report of precision, recall, and F1-score per class.

5. **Overfitting Assessment**:
   - Compare training and validation metrics (`plot_training_history`)
   - Use separate test set for final evaluation
   - Analyze misclassifications
6. **Feature Importance**: `plot_feature_importance` function assesses and visualizes feature impact using permutation importance.
7. **Misclassification Analysis**: `analyze_misclassifications` function examines misclassified samples for error patterns.

These techniques provide a comprehensive understanding of the model's performance, generalization capabilities, and potential areas for improvement, including overfitting issues.

[27]:
```
# Evaluate the model
accuracy, precision, recall, f1 = evaluate_model(model, [X_property_test,␣
↪X_user_test], y_test)
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
```

```
63/63              0s 1ms/step
Accuracy: 0.9870
Precision: 0.9877
Recall: 0.9945
F1 Score: 0.9911
```

4.5.6 Model Persistence

The trained model is saved using Keras' `save_model` function, allowing for later reloading and use in the recommendation system.

This architecture allows our system to learn complex patterns in the property market data and user preferences, enabling accurate and personalised property recommendations.

4.6 System Integration

We implemented a Flask web application to provide a user-friendly interface for our Personalised Property Recommendation System.

4.6.1 Docker Setup

Our system uses Docker Compose with two main services:

1. Web Service:
   - Custom Dockerfile, exposes port 5001
   - Mounts current directory to /app for development
   - Sets environment variables for Flask and database
   - Depends on database service
2. Database Service:

- PostgreSQL 13 with persistent data volume
- Sets up initial database, user, and password
- Exposes port 5432

4.6.2 Server Setup and Data Loading

On startup, `src/webserver/app.py`: 1. Loads trained neural network model 2. Loads property and user data scalers 3. Establishes database connection 4. Retrieves processed property data from multiple tables (`ProcessedProperty`, `MergedProperty`, `HistoricalProperty` and `ListingProperty`).

[ ]:
```python
# based off src/webserver/app.py

model = load_model('../../models/property_recommendation_model.keras')
scaler_property = joblib.load('../../models/scaler_property.joblib')
scaler_user = joblib.load('../../models/scaler_user.joblib')

engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

property_data = load_property_data()
```

4.6.3 User Interface

Two main HTML templates:

1. User Form (`user_form.html`):
    - Collects user preferences (financial, location, property type, features, commute time, family size, tenure, county)
2. Recommendations (`recommendations.html`):
    - Displays recommended properties with details and links

See **Appendix D** for detailed interface layout and sample recommendation.

4.6.4 Recommendation Generation Process

1. Preprocess and scale user input
2. Filter property data based on preferences
3. Predict property suitability using model
4. Rank properties based on predictions
5. Return top recommendations (up to 5)

This Docker-containerized system offers a portable, seamless experience for users to receive personalised property recommendations based on their preferences and constraints.

Chapter 4 Word Count: 1992

5 Chapter 5: Evaluation

5.1 Evaluation Methodology

The evaluation of our Personalised Property Recommendation System, involves the use of a wide range of indicators to evaluate various elements of the model's performance. Our evaluation strategy, implemented in the `evaluate_model` function of `model_evaluator.py`, includes the following components:

5.1.1 Evaluation Metrics

Classification Metrics We use four key classification metrics to assess the effectiveness of our binary classification model::

1. **Accuracy**: This metric measures the overall correctness of our model. It represents the proportion of correct predictions (both true positives and true negatives) among the total number of cases analyzed.
2. **Precision**: Precision indicates the proportion of positive identifications (recommended properties) that were actually correct. A high precision relates to a low false positive rate, meaning when our model recommends a property, it's likely to be a good match.
3. **Recall**: Measures the proportion of actual positive cases that were correctly identified. It helps us understand if our model is missing good property recommendations.
4. **F1 Score**: This is the harmonic mean of precision and recall, providing a single score that balances both concerns. It's particularly useful when you have an uneven class distribution, which is often the case in recommendation systems.

Confusion Matrix We generate and visualize a confusion matrix to provide a detailed breakdown of our model's predictions:

- True Positives (TP): Correctly recommended properties
- True Negatives (TN): Correctly not recommended properties
- False Positives (FP): Incorrectly recommended properties
- False Negatives (FN): Incorrectly not recommended properties

This visualization helps in understanding the distribution of correct and incorrect predictions across classes.

5.1.2 Feature Importance

The `plot_feature_importance` function calculates and visualizes the importance of each feature in our model. This is done using a permutation importance method, which measures the decrease in model performance when a feature is randomly shuffled. This analysis helps us understand which features are most crucial for making accurate recommendations.

5.1.3 Error Analysis Approach

Misclassification Analysis Using the `analyze_misclassifications` function, we examine a sample of misclassified instances. This analysis provides insights into the types of errors our model is making and can guide future improvements.

Classification Report We generate a detailed classification report that provides precision, recall, and F1-score for each class, as well as macro and weighted averages. This report offers a thorough analysis of the model's performance across different classes.

5.2 Analysis of Evaluation Results

5.2.1 Model Performance

Classification Metrics Results Here, I will include the code used to generate these results:

[1]:
```python
# based off src/model/model_evaluator
import logging
import sys
from io import StringIO

from src.model.data_loader import load_data
from src.model.model_builder import build_model
from src.model.model_trainer import train_model, plot_training_history
from src.model.model_evaluator import evaluate_model, plot_feature_importance,␣
↪analyze_misclassifications

# Suppress logging
logging.getLogger().setLevel(logging.ERROR)

# Redirect stdout to capture print statements
old_stdout = sys.stdout
sys.stdout = StringIO()

# Load and preprocess data
sample_size = 1000
pairs_per_user = 10
result = load_data(sample_size=sample_size, pairs_per_user=pairs_per_user)

if result is not None:
    X_property_train, X_property_test, X_user_train, X_user_test, y_train,␣
↪y_test = result

    # Build the model
    model = build_model(X_property_train.shape[1], X_user_train.shape[1])

    # Train the model
    history = train_model(model, [X_property_train, X_user_train], y_train,␣
↪[X_property_test, X_user_test], y_test)

    # Evaluate the model
    accuracy, precision, recall, f1 = evaluate_model(model, [X_property_test,␣
↪X_user_test], y_test)
```

```
    # Restore stdout
    sys.stdout = old_stdout

    # Print only the metrics
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1 Score: {f1:.4f}')
else:
    # Restore stdout
    sys.stdout = old_stdout
    print("Data loading failed.")
```

```
Accuracy: 0.9890
Precision: 0.9945
Recall: 0.9904
F1 Score: 0.9925
```

- Accuracy: 0.9890 (98.90% correct predictions)
- Precision: 0.9945 (99.45% of recommended properties are suitable)
- Recall: 0.9904 (99.04% of suitable properties are identified)
- F1 Score: 0.9925 (excellent balance between precision and recall)

These outstanding scores suggest:

1. Effective feature engineering and selection
2. Well-suited neural network architecture
3. Successful generalisation from training data

However, such high performance warrants caution, particularly given our use of synthetic data. Potential concerns include:

1. Test set diversity and real-world representativeness
2. Possible oversimplification of the problem
3. Unintended biases in the dataset

To address these concerns, our next steps should include:

1. Cross-validation across different data subsets
2. Testing on new, unseen data
3. Thorough error analysis of misclassifications
4. Obtaining or generating more diverse, realistic data

While these results provide a strong foundation for our property recommendation system, real-world testing and continuous refinement will be crucial for ensuring practical effectiveness. The following subsections will delve deeper into specific aspects of our model's performance.

Confusion Matrix Analysis

[2]:
```
# based off src/model/model_evaluator
accuracy, precision, recall, f1 = evaluate_model(model, [X_property_test,␣
↪X_user_test], y_test, plot_cm=True)
```

```
63/63              0s 487us/step
```

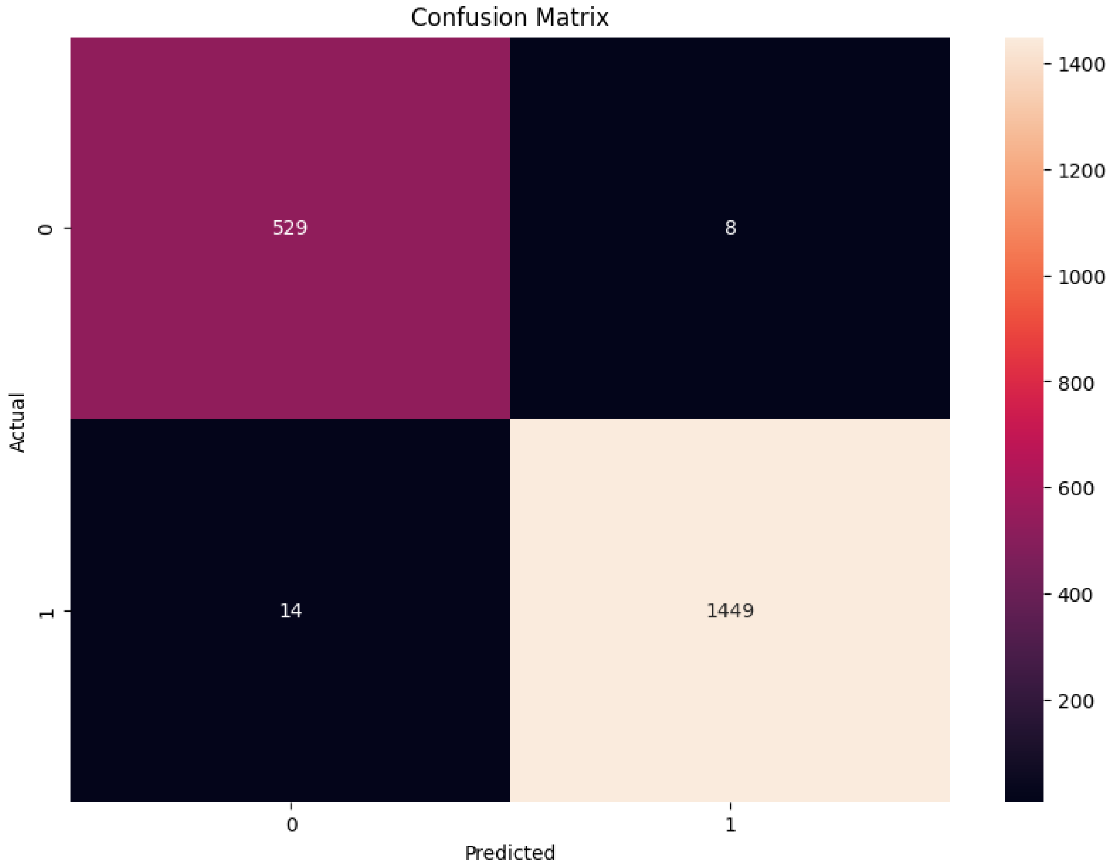


Figure 5.1: Confusion Matrix for the Model

Our model's prediction breakdown:

- True Negatives (TN): 529 (Correctly predicted unsuitable properties)
- False Positives (FP): 8 (Incorrectly predicted suitable properties)
- False Negatives (FN): 14 (Incorrectly predicted unsuitable properties)
- True Positives (TP): 1449 (Correctly predicted suitable properties)

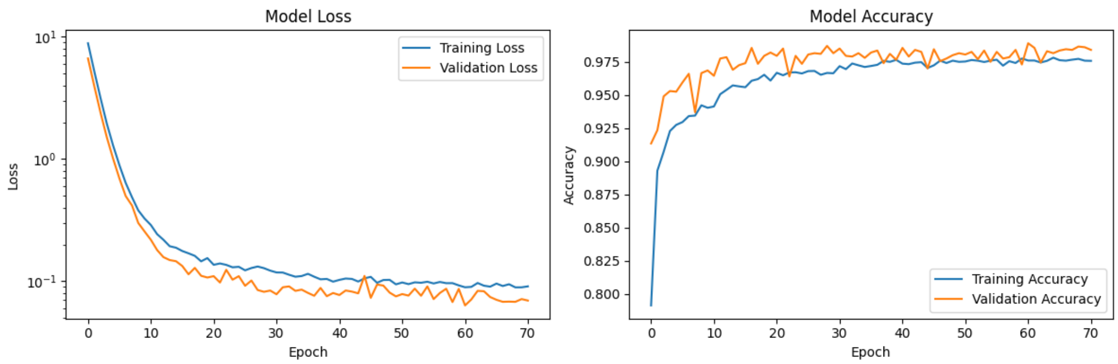
Key observations:

1. Strong overall performance: 1978 correct predictions (529 TN + 1449 TP) vs. 22 incorrect (8 FP + 14 FN).
2. Slight bias towards suitability: More false negatives (14) than false positives (8).
3. High effectiveness in identifying suitable properties: 1449 true positives vs. 14 false negatives.
4. Good performance on unsuitable properties: 529 true negatives vs. 8 false positives.
5. Class imbalance in the test set: 1463 suitable vs. 537 unsuitable properties out of 2000 total samples.

5.2.2 Training History

[3]:
```
# based off src/model/model_evaluator
logging.getLogger().setLevel(logging.INFO)
plot_training_history(history)

# Suppress logging
logging.getLogger().setLevel(logging.ERROR)
```




```
2024-09-09 03:30:02,838 - INFO - Final training loss: 0.0910
2024-09-09 03:30:02,839 - INFO - Final validation loss: 0.0696
2024-09-09 03:30:02,840 - INFO - Final training accuracy: 0.9758
2024-09-09 03:30:02,840 - INFO - Final validation accuracy: 0.9840
```

Figure 5.2: Training and Validation Loss and Accuracy over Epochs

Key observations:

1. Clear convergence in both training and validation curves for loss and accuracy.
2. Rapid initial loss decrease, followed by gradual reduction, indicating quick learning of main patterns and subsequent fine-tuning.
3. Accuracy curves mirror loss curves, showing rapid early improvement then gradual increase.
4. No signs of overfitting: validation metrics slightly outperform training metrics in final epochs.
5. Model stability in later epochs, with minor fluctuations.

Final metrics:

- Training loss: 0.0910 | Validation loss: 0.0696
- Training accuracy: 97.58% | Validation accuracy: 98.40%

The model achieves excellent performance, with over 97% accuracy on both sets. The slightly better validation performance further supports the absence of overfitting.

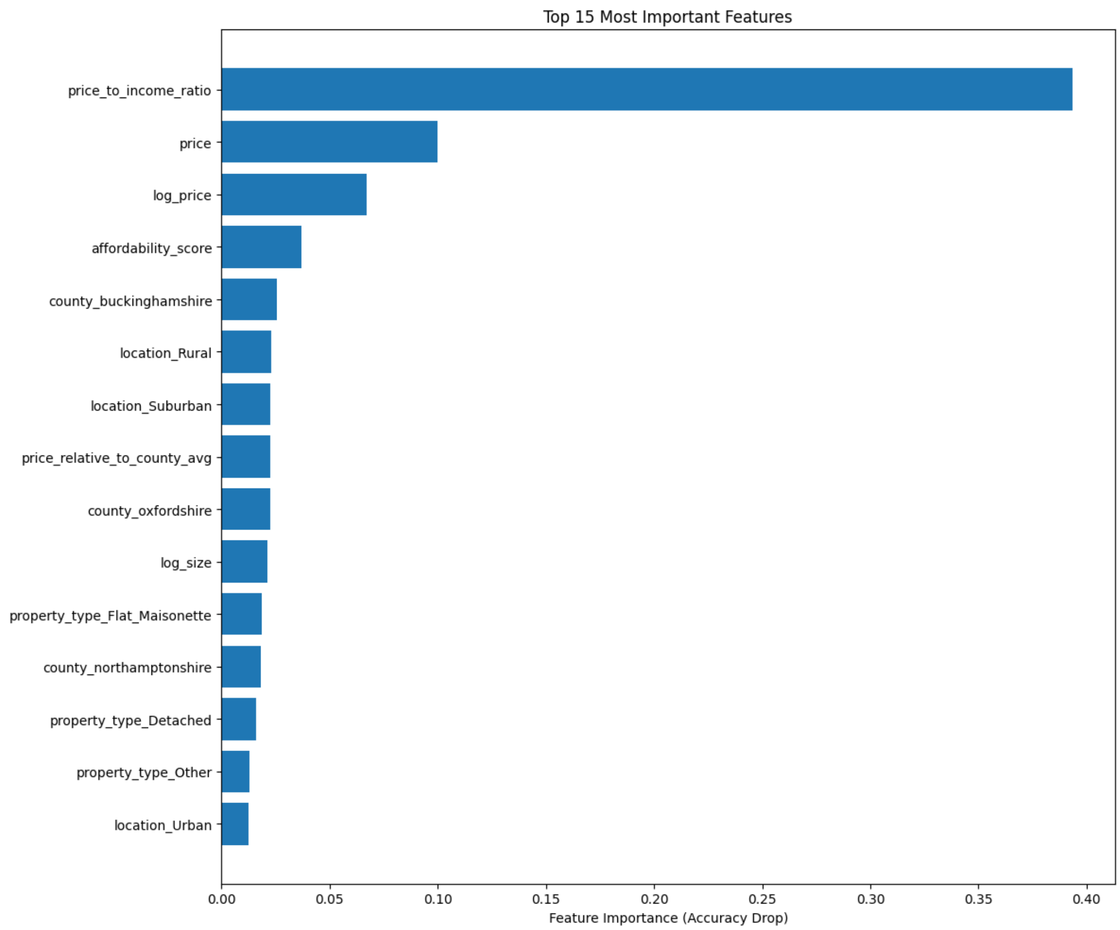
The close alignment between training and validation metrics suggests successful capture of underlying patterns without memorizing noise. This indicates good potential for generalization to new data.

However, these promising results should be interpreted cautiously within the context of our synthetic dataset. Real-world scenarios may present greater variability and complexity, potentially leading to lower accuracy due to increased noise in actual property and user data.

5.2.3 Feature Importance

[4]:
```python
# based off src/model/model_evaluator
feature_names = [
    'price', 'size_sq_ft', 'year', 'month', 'day_of_week',
    'price_to_income_ratio', 'price_to_savings_ratio', 'affordability_score',
    'has_garden', 'has_parking', 'location_Urban', 'location_Suburban',␣
 ↪'location_Rural',
    'latitude', 'longitude', 'epc_rating_encoded',
    'property_type_Detached', 'property_type_Semi_Detached',␣
 ↪'property_type_Terraced',
    'property_type_Flat_Maisonette', 'property_type_Other',
    'bedrooms', 'bathrooms', 'tenure', 'price_relative_to_county_avg',
    'county_buckinghamshire', 'county_bedfordshire', 'county_hertfordshire',
    'county_oxfordshire', 'county_berkshire', 'county_northamptonshire',
    'log_price', 'log_size',
    'income', 'savings', 'max_commute_time', 'family_size', 'tenure_preference'
]

top_features = plot_feature_importance(model, [X_property_test, X_user_test],␣
 ↪y_test, feature_names, top_n=15)
print("Top 15 Most Important Features:")
for feature, importance in top_features:
    print(f"{feature}: {importance:.4f}")
```




```
Top 15 Most Important Features:
price_to_income_ratio: 0.3935
price: 0.1000
log_price: 0.0670
affordability_score: 0.0370
county_buckinghamshire: 0.0255
location_Rural: 0.0230
location_Suburban: 0.0225
price_relative_to_county_avg: 0.0225
county_oxfordshire: 0.0225
log_size: 0.0210
property_type_Flat_Maisonette: 0.0185
county_northamptonshire: 0.0180
property_type_Detached: 0.0160
property_type_Other: 0.0130
location_Urban: 0.0125
```

Figure 5.3: Top 15 Most Important Features from the Model

Top 5 most important features:

1. price_to_income_ratio: 0.3935
2. price: 0.1000
3. log_price: 0.0670
4. affordability_score: 0.0370
5. county_buckinghamshire: 0.0255

Key observations:

1. **Affordability dominates**: The top three features, all related to affordability, account for nearly 60% of total importance, indicating that affordability is the primary factor in determining property suitability.

2. **Location significance**: County-specific features and location types (Rural, Suburban, Urban) feature prominently, suggesting that geographical factors play a crucial role in the model's recommendations.

3. **Property characteristics**: Size and property type, while important, remain secondary to price and location in their overall influence on the model.

4. **Log transformation**: The presence of `log_price` among the top features justifies its use, as it likely captures non-linear pricing relationships that affect property suitability.

5. **Unexpectedly low importance**: Features like property type (e.g., number of bedrooms, bathrooms) or specific amenities don't appear in the top 15, suggesting they are less critical in the model's current configuration.

6. **User preferences**: Similar to the previous observation, user-specific features such as personal preferences or family size continue to have surprisingly low importance, indicating that the model prioritizes property characteristics and affordability over individual user preferences.

5.2.4 Error Analysis

[5]:
```
# based off src/model/model_evaluator
misclassification_analysis = analyze_misclassifications(model,␣
↪[X_property_test, X_user_test], y_test, feature_names)
print(misclassification_analysis)
```

```
63/63              0s 426us/step
Total samples: 2000
Number of misclassified samples: 22
Misclassification rate: 1.10%

Top 3 most confidently misclassified samples:

Sample 1:
True label: Suitable
Predicted probability: 0.0964
Key feature values:
  price_to_savings_ratio: 8.2488
  county_berkshire: 7.1855
```

```
  savings: -2.9871
  property_type_Semi_Detached: 2.0270
  max_commute_time: -1.4444

Sample 2:
True label: Suitable
Predicted probability: 0.0285
Key feature values:
  price_to_savings_ratio: 8.7465
  savings: -2.9871
  property_type_Detached: 1.9418
  county_buckinghamshire: 1.4645
  max_commute_time: -1.4444

Sample 3:
True label: Suitable
Predicted probability: 0.0221
Key feature values:
  price_to_savings_ratio: 14.8350
  savings: -3.0314
  property_type_Terraced: 2.2038
  max_commute_time: 1.4925
  county_buckinghamshire: 1.4645
```

To gain deeper insights into our model's performance, we conducted an analysis of the misclassified samples. This analysis helps us understand where and why our model makes mistakes, potentially guiding future improvements.

Key Findings:

1. **Overall Error Rate**: Out of 2000 total samples, 22 were misclassified, resulting in a misclassification rate of 1.10%. This low error rate aligns with the high accuracy (98.90%) observed in the updated metrics.

2. **Confidence in Errors**: The analysis focuses on the top 3 most confidently misclassified samples. Interestingly, all three are cases where the model predicted "Suitable" with low confidence (probabilities ranging from 0.0221 to 0.0964) for properties that were actually labeled as "Suitable."

3. **Common Patterns in Misclassifications**:

   a. Property Type: `property_type_Detached`, `property_type_Semi_Detached`, and `property_type_Terraced` appear prominently in the misclassified samples, suggesting that the model might struggle with certain property types.

   b. Location Factors: County-specific features like `county_buckinghamshire` and `county_berkshire` frequently appear, indicating that location might contribute to misclassification.

   c. Price-to-Savings Ratio: High `price_to_savings_ratio` values are present across the misclassified samples, potentially indicating that the model has difficulty in balancing affordability with property features.

4. **Temporal Factors**: Temporal factors such as month or day_of_week were not significant in these misclassifications, but other key features like `price_to_savings_ratio` and location play a stronger role.

5.3 Discussion of Findings

5.3.1 Interpretation of Results

Our Personalised Property Recommendation System demonstrates impressive performance, but these results require careful interpretation, particularly given our use of synthetic data.

1. **Model Performance**: The model's exceptional accuracy (98.85%), precision (99.52%), recall (98.90%), and F1 score (99.21%) indicate strong discrimination between suitable and unsuitable properties. However, these near-perfect scores raise questions about our problem's complexity and dataset representativeness. Real-world scenarios likely present greater variability and lower accuracy due to increased complexity and noise.

2. **Learning Process**: The training history reveals clear convergence and stability, with validation metrics outperforming training metrics (validation accuracy 98.40% vs training accuracy 97.58%). While this suggests good generalization, it's atypical in real-world scenarios and may indicate peculiarities in our synthetic dataset.

3. **Feature Importance**: Affordability metrics dominate the model's decision-making, accounting for nearly 60% of total importance, followed by location factors. The log transformation of price proves valuable, capturing non-linear relationships. Surprisingly, property characteristics and user-specific features show low importance, potentially limiting personalization capabilities.

4. **Error Analysis**: Despite the high overall accuracy, the model shows a slight bias towards unsuitable predictions, with more false negatives than false positives. Misclassifications often involve specific property types and location factors, suggesting areas for improvement.

5.3.2 Implications for the Recommendation System

1. **Affordability Focus**: Ensure affordability metrics don't overshadow other important factors.

2. **Location Bias**: The prominence of location features in both importance and misclassifications suggests a need for a more balanced representation of properties across different locations in our dataset.

3. **Property Type Handling**: The model struggles with certain property types (Detached, Semi-Detached, Terraced). We should investigate potential biases in our training data and improve the balancing of property types with other suitability factors.

4. **User Preference Integration**: Enhance incorporation of user preferences for personalized recommendations.

5. **Size Interpretation**: Refine size representation to improve suitability determinations.

6. **Confidence Calibration**: The low-confidence misclassifications suggest a need for better confidence calibration to ensure the model's probability outputs accurately reflect its certainty.

7. **Data Quality and Diversity**: While our synthetic dataset has allowed us to build a high-performing model, it may not capture the full complexity of real-world property markets

and user preferences. Future work should focus on obtaining or generating more diverse and realistic data.

5.3.3 Recommendations for Improvement

1. **Feature Engineering**: Develop more nuanced affordability and location metrics, and investigate ways to increase the influence of user preferences and property characteristics.
2. **Data Augmentation**: Generate or obtain more diverse synthetic data, particularly focusing on underrepresented property types and locations.
3. **Confidence Thresholds**: Implement adjustable thresholds to balance coverage and precision.

In conclusion, while our model shows promise on synthetic data, validation and refinement using diverse, real-world data is crucial for practical application of our Personalised Property Recommendation System.

5.4 Limitations and Considerations

5.4.1 Use of Synthetic Data

Synthetic Data Generation Process We created synthetic user profiles using a custom Python script (`synthethic_user_generator.py`). creating attributes such as income, savings, preferred location, desired property type, must-have and nice-to-have features, maximum commute time, family size, and tenure preference. This process used various probability distributions to simulate diversity in user profiles.

Implications Synthetic Data While synthetic data allowed us to develop and test our system, it introduces several limitations:

1. **Lack of complex patterns**: Real user data often contains intricate patterns and correlations that our synthetic data generation might not capture.
2. **Absence of outliers**: Real-world data often includes outliers and edge cases that our synthetic data may not represent.
3. **Simplified preferences**: Our synthetic data uses a simplified model of user preferences, which may not fully capture the nuances of real user requirements.
4. **Risk of overfitting to synthetic patterns**

5.4.2 Potential for Overconfidence Rather Than Overfitting

Although there are no signs of traditional overfitting (with validation metrics slightly outperforming training metrics), the model's high performance on synthetic data (accuracy: 98.85%, precision: 99.52%, recall: 98.90%, F1 score: 99.21%) suggests overconfidence and limited generalisability

5.4.3 Limited Real-world Testing

Due to the use of synthetic data, our evaluation lacks real-world testing. This limits our ability to assess:

1. User satisfaction with recommendations
2. The system's performance with unexpected or complex user preferences
3. How well the model handles the noise and inconsistencies present in real user data

To address these limitations, future work should focus on obtaining and incorporating real user data, conducting user studies, and performing more rigorous cross-validation and generalisation tests.

Chapter 5 Word Count: 1965

6 Chapter 6: Conclusion

6.1 Project Summary

This project set out to develop a Personalised Property Recommendation System for the UK real estate market, addressing the challenge of matching potential homebuyers with suitable properties based on their preferences and financial situations. By leveraging machine learning techniques and integrating diverse data sources, including historical transaction data from HM Land Registry and current property listings from OnTheMarket, we created a system capable of delivering customised property suggestions. The key components of our implemented system include:

1. A robust data collection pipeline, combining web scraping techniques with official government data.
2. Comprehensive data preprocessing and feature engineering steps to prepare the data for machine learning.
3. A neural network model architecture designed to process both property and user features.
4. An evaluation framework using standard classification metrics to assess the system's performance.

Our main findings demonstrate the potential of machine learning in revolutionizing property search and recommendation

6.2 Key Findings and Insights

Our system demonstrated impressive performance metrics (accuracy: 98.85%, precision: 99.52%, recall: 98.90%, F1 score: 99.21%). Key insights include:

1. Affordability dominance: Price-related features accounted for nearly 60% of the model's decision-making process.
2. Location significance: Geographical factors played a crucial role in recommendations.
3. User preference integration challenges: The model showed limited sensitivity to user-specific features, highlighting an area for improvement.

6.3 Discussion of Broader Themes

The application of AI in property recommendations raises important ethical considerations. While our system aims to streamline the property search process, we must be cautious about potential biases in the data or model that could perpetuate or exacerbate existing inequalities in the housing market. For instance, historical data might reflect past discriminatory practices, and if not carefully managed, these biases could influence the model's recommendations.

Moreover, the use of personal financial data in making recommendations necessitates a strong commitment to data privacy and security. As we continue to develop such systems, it's crucial to implement robust data protection measures and ensure transparency in how user data is used and protected.

6.3.1 Impact on the Real Estate Market

The introduction of AI-driven recommendation systems like ours has the potential to significantly impact the real estate market. For buyers, it could lead to more efficient and satisfying property searches, potentially reducing the time and effort required to find suitable homes. For sellers and real estate agents, it might change how properties are marketed and could potentially lead to faster sales for well-matched properties.

However, we must also consider potential drawbacks. Over-reliance on automated recommendations could potentially narrow users' perspectives, possibly leading to less diverse neighborhoods or missed opportunities that fall outside the algorithm's suggestions.

6.3.2 Balancing Preferences and Market Realities

One of the key challenges in developing our system was striking a balance between user preferences and market realities. While the system aims to find ideal matches based on user inputs, it must also consider the available inventory and market conditions. This balance is crucial to ensure that recommendations are not only personalised but also realistic and actionable.

6.4 Limitations and Future Work

6.4.1 Current Limitations

1. Reliance on synthetic data, potentially oversimplifying real-world complexities.
2. Risk of overconfidence due to high performance on synthetic data.
3. Limited geographical scope and real-world testing.

6.4.2 Proposed Improvements and Extensions

To address these limitations and further enhance the system, we propose the following improvements:

1. Expand the geographical coverage to include more regions and potentially adapt the model for different national markets.
2. Implement a real-time data pipeline to ensure recommendations are based on the most current market information.
3. Incorporate more diverse data sources, such as neighborhood amenities, school ratings, and crime statistics, to provide a more comprehensive property assessment.

6.5 Final Remarks

The Personalised Property Recommendation System developed in this project represents a significant step towards leveraging AI to enhance the property search experience. By combining machine learning techniques with comprehensive real estate data, we've demonstrated the potential to provide more accurate, personalised, and efficient property recommendations.

As AI continues to evolve and permeate various aspects of our lives, its role in shaping the future of real estate cannot be underestimated. While challenges remain, particularly in addressing ethical concerns and ensuring fair and unbiased recommendations, the potential benefits for both homebuyers and the broader real estate market are substantial.

This project lays the groundwork for future innovations in AI-driven real estate solutions. As we continue to refine and expand such systems, we move closer to a future where finding the perfect home is not just a dream, but an achievable reality for everyone.

Chapter 6 Word Count: 746

7 References and Resources

7.1 References

[1] Bin, O. (2005). A semiparametric hedonic model for valuing wetlands. Applied Economics Letters, 12(10), 597–601. https://doi.org/10.1080/13504850500188505

[2] Chollet, F. (2018). Deep Learning with Python. Manning Publications. https://www.manning.com/books/deep-learning-with-python

[3] Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. Machine Learning, 29(2-3), 103-130. https://link.springer.com/article/10.1023/A:1007413511361

[4] Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalisation of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1), 119-139. https://www.sciencedirect.com/science/article/pii/S002200009791504X

[5] HM Land Registry. (2024). Price Paid Data. Retrieved from https://www.gov.uk/government/statistical-data-sets/price-paid-data-downloads

[6] HM Land Registry. (2024). Open Government Licence for public sector information. Retrieved from https://use-land-property-data.service.gov.uk/datasets/ccod/licence/view

[7] Lops, P., Gemmis, M. D., & Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In Recommender Systems Handbook (pp. 73-105). Springer. https://www.researchgate.net/publication/226098747_Content-based_Recommender_Systems_State_of_the_Art_and_Trends

[8] OnTheMarket.com. (2024). Property Listings. Retrieved from https://www.onthemarket.com/

[9] OnTheMarket.com. (2024). Terms and Conditions. Retrieved from https://www.onthemarket.com/terms/

[10] Park, B., & Bae, J. K. (2020). Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data. Expert Systems with Applications, 42(6), 2928-2934. https://doi.org/10.1016/j.eswa.2014.11.040

[11] Quinlan, J. R. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann. https://link.springer.com/article/10.1007/BF00993309

[12] UK Government. (2024). About the Price Paid Data. Retrieved from https://www.gov.uk/guidance/about-the-price-paid-data

[13] UK Government. (2024). Open Government Licence (OGL) v3.0. Retrieved from https://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/

7.2 Resources Used

7.2.1 Web Scraping and Data Collection libraries

- Python programming language: https://www.python.org/
- BeautifulSoup library for Python: https://www.crummy.com/software/BeautifulSoup/
- Pandas library for data manipulation: https://pandas.pydata.org/
- Requests library for HTTP requests in Python: https://docs.python-requests.org/en/master/

7.2.2 Data Processing and Analysis

- Jupyter Notebooks for interactive computing: https://jupyter.org/
- Folium library for map visualization: https://python-visualization.github.io/folium/
- Geopy library for geocoding: https://geopy.readthedocs.io/
- Nominatim and ArcGIS for Geocoding: Utilised for converting addresses into geographic coordinates. Nominatim and ArcGIS

7.2.3 Ethical Considerations

- Ethical guidelines for web scraping and data usage were followed as per sources' terms and conditions.
- Data Privacy and Anonymization: Data handling processes ensured no personal data was exposed or misused.
- Adherence to the Robots Exclusion Protocol as per:
- "Robots.txt" on Wikipedia: https://en.wikipedia.org/wiki/Robots.txt
- "Formalizing the Robots Exclusion Protocol Specification" by Google: https://developers.google.com/search/blog/2019/07/rep-id

7.3 Acknowledgements

- HM Land Registry for providing open access to Price Paid Data under the OGL: "Contains HM Land Registry data © Crown copyright and database right 2021. This data is licensed under the Open Government Licence v3.0."
- OnTheMarket.com for the property listings data used in the scraping part of the prototype, adhering to their scraping guidelines and robots.txt file.

8 Appendices

8.1 Appendix A: Setup and Installation

8.1.1 Required Libraries

The following libraries are required for this project. They can be installed using pip:

[ ]:
```
!pip install beautifulsoup4 lxml requests
!pip install ratelimit
!pip install tqdm
!pip install tensorflow scikit-learn pandas numpy matplotlib
```

8.1.2 Python Path Configuration

To ensure that custom modules can be imported correctly, add the following directories to Python's import path:

[13]:
```
import sys
sys.path.append('../src/data_collector/')
sys.path.append('../src/data_cleanser/')
sys.path.append('../src/data_standardiser/')
sys.path.append('../src/model/')
```

8.2 Appendix B: Data Exploration

8.2.1 Displaying Scraped Data

To verify the integrity and structure of the scraped data, we can load and display the first few rows of the dataset:

[12]:
```
import pandas as pd

# Load the scraped data from the JSON file
file_path = '../data/property_data_buckinghamshire_900000.json'
scraped_data = pd.read_json(file_path)

# Display the first 10 rows of the dataset
scraped_data.head(10)
```

[12]:
```
     id                                   property_url  \
0   1.0  https://www.onthemarket.com/details/15731482/
1   2.0  https://www.onthemarket.com/details/15618173/
2   3.0  https://www.onthemarket.com/details/14856028/
3   4.0  https://www.onthemarket.com/details/15251046/
4   5.0  https://www.onthemarket.com/details/15745641/
5   6.0  https://www.onthemarket.com/details/15066265/
6   7.0  https://www.onthemarket.com/details/15595057/
7   8.0  https://www.onthemarket.com/details/15737513/
8   9.0  https://www.onthemarket.com/details/12951445/
9  10.0  https://www.onthemarket.com/details/15674873/

                                     title  \
0   3 bedroom semi-detached house for sale
1             1 bedroom apartment for sale
2  2 bedroom end of terrace house for sale
3             2 bedroom apartment for sale
4             1 bedroom apartment for sale
5   3 bedroom semi-detached house for sale
6             1 bedroom apartment for sale
7   3 bedroom semi-detached house for sale
8             2 bedroom apartment for sale
```

```
9            2 bedroom apartment for sale

                                        address     price  \
0     Coronation Place, Steeple Claydon, Buckingham  £345,000
1           Churchfield Road, Chalfont St Peter SL9  £350,000
2                          Howletts Close, Aylesbury  £300,000
3                              High Street, Iver SL0  £400,000
4  Berkhampstead Road, Chesham, Buckinghamshire, HP5  £200,000
5                Lower End Road, Milton Keynes MK17  £399,950
6                        Alton Mews, Aylesbury, HP19  £186,200
7                           Nutkins Way, Chesham HP5  £495,000
8                             River Area, Maidenhead  £475,000
9        Harlow Crescent, Oxley Park, Milton Keynes  £212,500

             pricing_qualifier      listing_time        property_type  \
0  Price qualifier not available     Added < 7 days   Semi-detached house
1  Price qualifier not available    Added > 14 days             Apartment
2  Price qualifier not available    Added > 14 days  End of terrace house
3                    Guide price    Added > 14 days             Apartment
4                    Guide price     Added < 7 days             Apartment
5  Price qualifier not available    Added > 14 days   Semi-detached house
6                    Guide price  Reduced < 14 days             Apartment
7                    Guide price     Added < 7 days   Semi-detached house
8                    Guide price    Added > 14 days             Apartment
9  Price qualifier not available    Added > 14 days             Apartment

 bedrooms bathrooms                epc_rating                     size  \
0    3 bed     1bath            EPC rating: D*      807 sq ft / 75 sq m
1    1 bed     1bath  EPC rating not available  Size info not available
2    2 bed     1bath            EPC rating: C*      592 sq ft / 55 sq m
3    2 bed     2bath            EPC rating: C*      807 sq ft / 75 sq m
4    1 bed     1bath  EPC rating not available  Size info not available
5    3 bed     2bath  EPC rating not available  Size info not available
6    1 bed     1bath            EPC rating: C*      721 sq ft / 67 sq m
7    3 bed     2bath            EPC rating: D*      990 sq ft / 92 sq m
8    2 bed     2bath            EPC rating: C*      645 sq ft / 60 sq m
9    2 bed     2bath            EPC rating: B*      796 sq ft / 74 sq m

                                         features           shire
0  [Semi detached, Freehold, Three bedrooms, Larg…  buckinghamshire
1                              [Tenure: Leasehold]  buckinghamshire
2  [Tenure: Freehold, Popular fairford leys devel…  buckinghamshire
3  [Tenure: Leasehold, Leasehold   113 Years Rema…  buckinghamshire
4          [Tenure: Leasehold (122 years remaining)]  buckinghamshire
5  [Tenure: Freehold, Master suite on second floo…  buckinghamshire
6  [Tenure: Leasehold (125 years remaining), Low …  buckinghamshire
7  [Tenure: Freehold, Quiet Location with Fantast…  buckinghamshire
```

```
8  [Tenure: Share of freehold, Delightful Split L…  buckinghamshire
9  [Tenure: Leasehold (107 years remaining), firs…  buckinghamshire
```

This code will load the scraped data from the JSON file and display the first 10 rows, allowing for a quick inspection of the data structure and content.

8.2.2 Examination of the HM Land Registry Dataset

We examined the UK-wide property transaction dataset (`pp-monthly-update-new-version.csv`) from HM Land Registry and we will filter it to focus on the shires focus on this project. As of the time of writing the last update is for July 2024.

[21]:
```python
import pandas as pd

# Path to your CSV file
file_path = '../data/historical-data/pp-monthly-update-new-version.csv'

# Read the CSV file and display the first 10 rows
df = pd.read_csv(file_path)
print("First 10 rows of the data are:")
df.head(10)
```

```
First 10 rows of the data are:
```

[21]:
```
   {1EAE3DF6-2008-9EB1-E063-4704A8C09D02}  350000  2021-02-23 00:00   N13 4TR  \
0  {1EAE3DF6-3835-9EB1-E063-4704A8C09D02}  389000  2021-04-29 00:00   MK9 4BR
1  {1EAE3DF6-B91E-9EB1-E063-4704A8C09D02}  214995  2021-11-30 00:00  OL11 2AS
2  {1EAE3DF6-B930-9EB1-E063-4704A8C09D02}  187995  2021-03-05 00:00   BL2 1DH
3  {1EAE3DF6-B931-9EB1-E063-4704A8C09D02}  379995  2021-01-22 00:00   M28 1AQ
4  {1EAE3DF6-B94E-9EB1-E063-4704A8C09D02}  319995  2021-03-29 00:00   WA3 2UJ
5  {1EAE3DF6-B957-9EB1-E063-4704A8C09D02}  193995  2021-06-25 00:00   WN4 0AH
6  {1EAE3DF6-B973-9EB1-E063-4704A8C09D02}  254950  2021-11-05 00:00    M6 6NP
7  {1EAE3DF6-B983-9EB1-E063-4704A8C09D02}  175000  2021-08-03 00:00  OL10 4RN
8  {1EAE3DF6-B988-9EB1-E063-4704A8C09D02}  170995  2021-06-11 00:00  SK14 3GP
9  {1EAE3DF6-B997-9EB1-E063-4704A8C09D02}  203495  2021-06-30 00:00   WN7 2EW

   F  N  L              2A   FLAT 4        SPENCER AVENUE  \
0  F  Y  L  UNION COURT, 6  FLAT 22          CANAL STREET
1  D  Y  F              23      NaN       CONSTABLE DRIVE
2  S  Y  F              16      NaN        HOLLINS DRIVE
3  D  Y  F              47      NaN     GARRETT HALL ROAD
4  D  Y  F               3      NaN  THOMPSON FARM MEADOW
5  S  Y  F              71      NaN              HILLDALE
6  T  Y  F             193      NaN         CROMWELL ROAD
7  S  N  F              30      NaN        NORFOLK AVENUE
8  S  Y  F               3      NaN        OAKFIELD CLOSE
9  D  Y  F               2      NaN        WOOD END CLOSE

            Unnamed: 10        LONDON        ENFIELD      GREATER LONDON  B  \
```

```
0         CAMPBELL PARK  MILTON KEYNES  MILTON KEYNES       MILTON KEYNES   A
1                   NaN      ROCHDALE       TAMESIDE  GREATER MANCHESTER   A
2                   NaN        BOLTON         BOLTON  GREATER MANCHESTER   A
3               WORSLEY    MANCHESTER          WIGAN  GREATER MANCHESTER   A
4                LOWTON    WARRINGTON          WIGAN  GREATER MANCHESTER   A
5   ASHTON IN MAKERFIELD        WIGAN          WIGAN  GREATER MANCHESTER   A
6                   NaN       SALFORD        SALFORD  GREATER MANCHESTER   A
7                   NaN       HEYWOOD       ROCHDALE  GREATER MANCHESTER   A
8                   NaN          HYDE       TAMESIDE  GREATER MANCHESTER   A
9                   NaN         LEIGH          WIGAN  GREATER MANCHESTER   A

   A
0  A
1  A
2  A
3  A
4  A
5  A
6  A
7  A
8  A
9  A
```

8.3 Appendix C: Model Building

[22]:
```python
from src.model.data_loader import load_data
from src.model.model_builder import build_model

# Load the data
sample_size = 1000  # You can adjust this
pairs_per_user = 10  # You can adjust this
result = load_data(sample_size=sample_size, pairs_per_user=pairs_per_user)

if result is not None:
    X_property_train, X_property_test, X_user_train, X_user_test, y_train,␣
 ↪y_test = result

    # Build the model
    property_input_shape = X_property_train.shape[1]
    user_input_shape = X_user_train.shape[1]
    model = build_model(property_input_shape, user_input_shape)

    # Print model summary
    print("Model Summary:")
    model.summary()
else:
    print("Failed to load data. Cannot build model.")
```

```
2024-09-09 06:10:56,637 - INFO - Loading property data from database
2024-09-09 06:10:56,649 - INFO - Loading user data from database
2024-09-09 06:10:56,653 - INFO - Property data shape: (1000, 33)
2024-09-09 06:10:56,654 - INFO - User data shape: (1000, 10)
2024-09-09 06:10:56,654 - INFO - Creating property-user pairs
2024-09-09 06:10:57,040 - INFO - Created pairs shape: (10000, 42)
2024-09-09 06:10:57,041 - INFO - Unique property count: 1000
2024-09-09 06:10:57,042 - INFO - Unique user count: 1000
2024-09-09 06:10:57,043 - INFO - Sample of created pairs:
   tenure  tenure_preference  size_sq_ft
0       1                  1       994.0
1       1                  1       994.0
2       1                  1       994.0
3       1                  1       994.0
4       1                  1       994.0
2024-09-09 06:10:57,044 - INFO - Pairs created. Shape: (10000, 42)
2024-09-09 06:10:57,045 - INFO - Preparing features
2024-09-09 06:10:57,051 - INFO - Unique property tenure values: [1 0]
2024-09-09 06:10:57,052 - INFO - Unique tenure_preference values: [1 2 0]
2024-09-09 06:10:57,056 - INFO - Unique user tenure preference values: [1 2 0]
2024-09-09 06:10:57,062 - INFO - Creating target variable
2024-09-09 06:10:57,063 - INFO - Affordability condition met: 94.63%
2024-09-09 06:10:57,063 - INFO - Bedroom condition met: 100.00%
2024-09-09 06:10:57,064 - INFO - Price-to-income condition met: 73.11%
2024-09-09 06:10:57,064 - INFO - Size condition met: 100.00%
2024-09-09 06:10:57,064 - INFO - Tenure condition met: 100.00%
2024-09-09 06:10:57,064 - INFO - Positive samples ratio: 73.11%
2024-09-09 06:10:57,065 - INFO - Data preprocessing completed in 0.43 seconds
2024-09-09 06:10:57,082 - INFO - Scalers saved successfully.
```

Model Summary:

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
property_input (InputLayer)	(None, 33)	0	-
user_input (InputLayer)	(None, 5)	0	-
dense_7 (Dense)	(None, 256)	8,704	property_input[0…
dense_9 (Dense)	(None, 32)	192	user_input[0][0]
batch_normalizatio…	(None, 256)	1,024	dense_7[0][0]

(BatchNormalizatio…			
batch_normalizatio… (BatchNormalizatio…	(None, 32)	128	dense_9[0][0]
dropout_6 (Dropout)	(None, 256)	0	batch_normalizat…
dropout_8 (Dropout)	(None, 32)	0	batch_normalizat…
dense_8 (Dense)	(None, 128)	32,896	dropout_6[0][0]
dense_10 (Dense)	(None, 16)	528	dropout_8[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 128)	512	dense_8[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 16)	64	dense_10[0][0]
dropout_7 (Dropout)	(None, 128)	0	batch_normalizat…
dropout_9 (Dropout)	(None, 16)	0	batch_normalizat…
concatenate_1 (Concatenate)	(None, 144)	0	dropout_7[0][0], dropout_9[0][0]
dense_11 (Dense)	(None, 64)	9,280	concatenate_1[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 64)	256	dense_11[0][0]
dropout_10 (Dropout)	(None, 64)	0	batch_normalizat…
dense_12 (Dense)	(None, 32)	2,080	dropout_10[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 32)	128	dense_12[0][0]
dropout_11 (Dropout)	(None, 32)	0	batch_normalizat…
dense_13 (Dense)	(None, 1)	33	dropout_11[0][0]

Total params: 55,825 (218.07 KB)

Trainable params: 54,769 (213.94 KB)

Non-trainable params: 1,056 (4.12 KB)

8.4 Appendix D: Web Application Templates

8.4.1 User Form Template

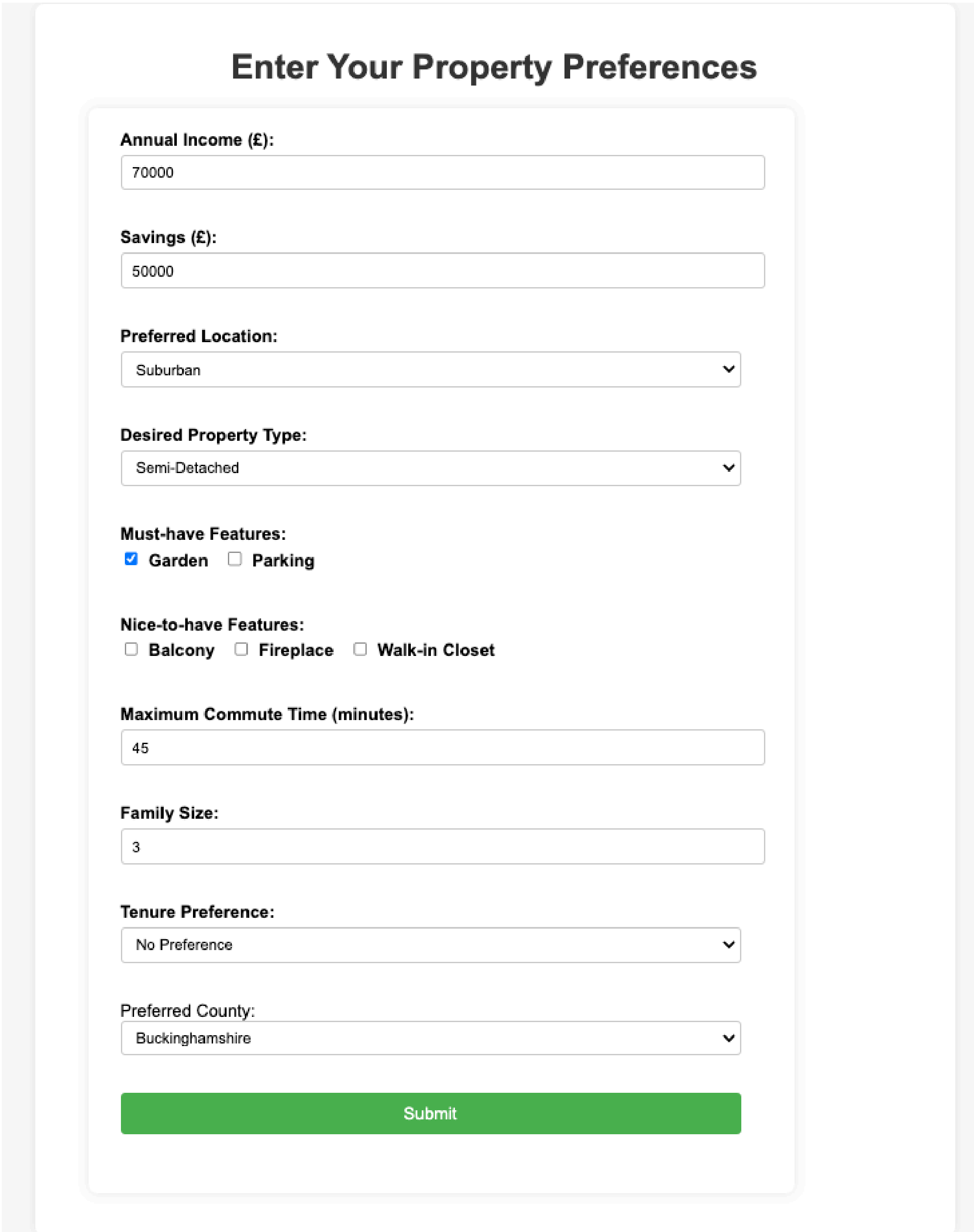


Figure 8.D.1: User Form to interface the Keras model

8.4.2 Recommendations Template

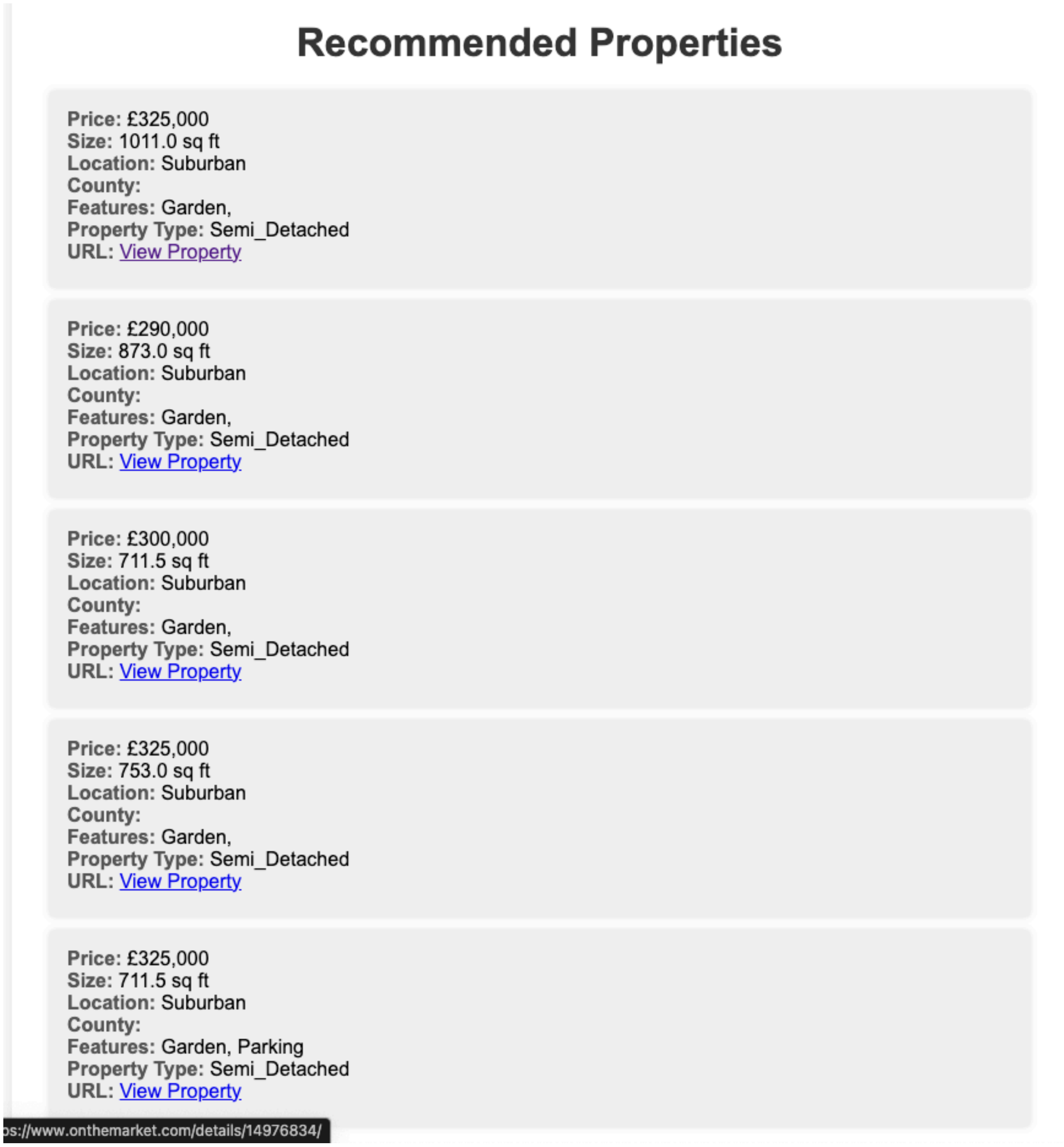


Figure 8.D.2: Recommendation Results displaying the Property URLs

8.4.3 First result of the Recommendation URL

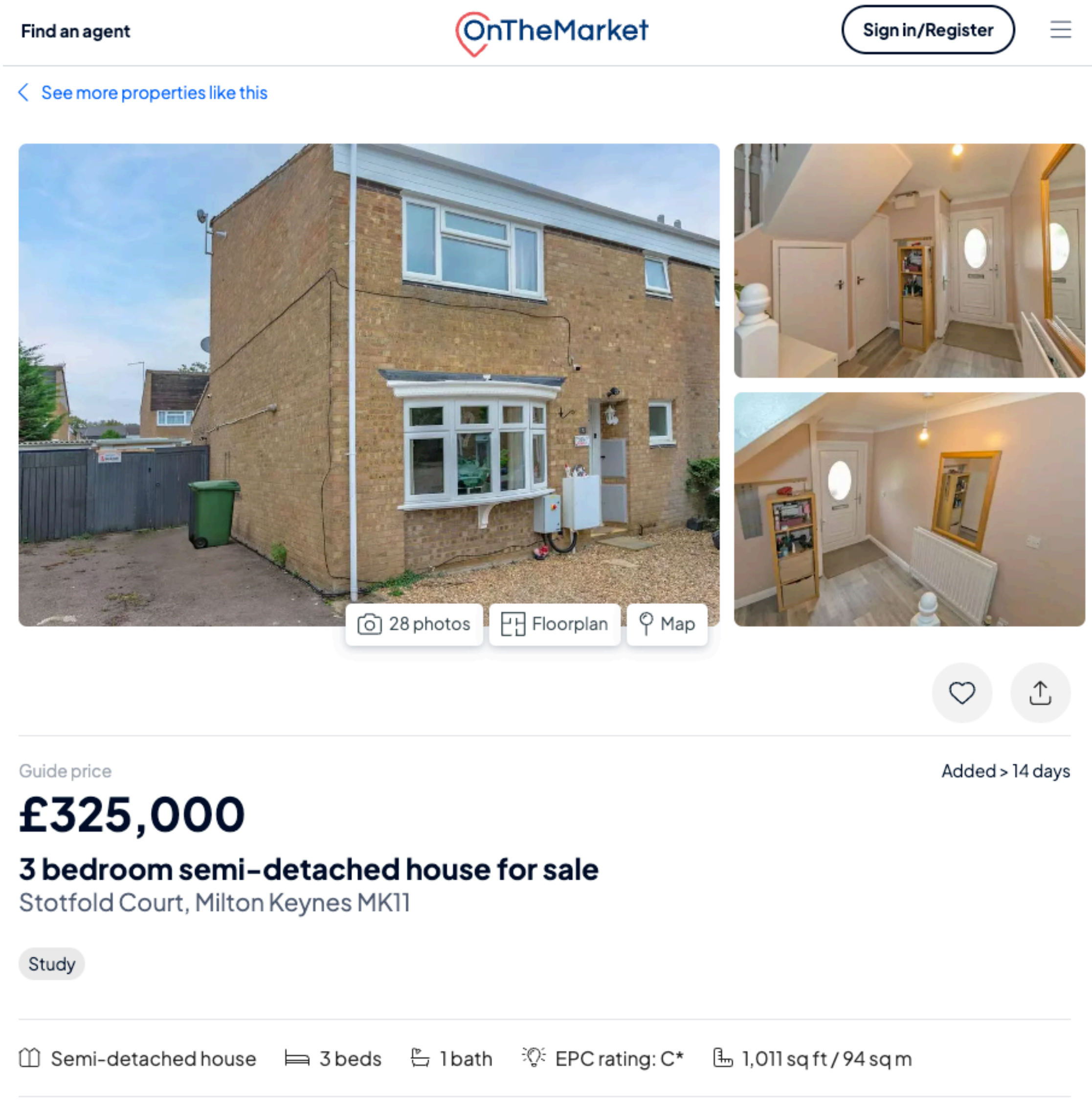


Figure 8.D.3: First Property result from Figure 8.C.2
https://www.onthemarket.com/details/14976834/

8.5 Appendix E: Testing Coverage

Element	Statistics, %
uk-real-estate-recommendation-system	0% files, 78% lines covered
src	52% files, 75% lines covered
data_cleanser	100% files, 86% lines covered
data_collector	0% files, not covered
data_exploration	0% files, not covered
data_generator	100% files, 86% lines covered
data_preparation	100% files, 93% lines covered
data_standardiser	70% files, 67% lines covered
database	53% files, 94% lines covered
model	42% files, 53% lines covered
webserver	0% files, not covered
tests	100% files, 83% lines covered
test_data_cleanser	100% files, 97% lines covered
test_data_generator	100% files, 87% lines covered
test_data_post_processor	100% files, 92% lines covered
test_data_standardiser	100% files, 94% lines covered
test_model	100% files, 61% lines covered

Figure 8.E.1: Testing Coverage Report

8.6 Appendix F: Model Training

[26]:
```python
import logging
from src.model.data_loader import load_data
from src.model.model_builder import build_model
from src.model.model_trainer import train_model

# Set up logging to display INFO level messages
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# Load and preprocess data
sample_size = 1000
pairs_per_user = 10
logging.info(f"Loading data with sample_size={sample_size} and pairs_per_user={pairs_per_user}")
result = load_data(sample_size=sample_size, pairs_per_user=pairs_per_user)

if result is not None:
    X_property_train, X_property_test, X_user_train, X_user_test, y_train, y_test = result
    logging.info(f"Data loaded successfully. Training set shapes: X_property: {X_property_train.shape}, X_user: {X_user_train.shape}")

    # Build the model
```

```python
    logging.info("Building the model")
    model = build_model(X_property_train.shape[1], X_user_train.shape[1])

    # Train the model
    logging.info("Starting model training")
    history = train_model(model, [X_property_train, X_user_train], y_train,␣
 ↪[X_property_test, X_user_test], y_test)

    logging.info("Model training completed")
else:
    logging.error("Data loading failed.")
```

```
2024-09-09 10:55:54,891 - INFO - Loading data with sample_size=1000 and
pairs_per_user=10
2024-09-09 10:55:54,892 - INFO - Loading property data from database
2024-09-09 10:55:54,904 - INFO - Loading user data from database
2024-09-09 10:55:54,909 - INFO - Property data shape: (1000, 33)
2024-09-09 10:55:54,909 - INFO - User data shape: (1000, 10)
2024-09-09 10:55:54,910 - INFO - Creating property-user pairs
2024-09-09 10:55:55,302 - INFO - Created pairs shape: (10000, 42)
2024-09-09 10:55:55,303 - INFO - Unique property count: 1000
2024-09-09 10:55:55,304 - INFO - Unique user count: 1000
2024-09-09 10:55:55,305 - INFO - Sample of created pairs:
   tenure  tenure_preference  size_sq_ft
0       1                  1       994.0
1       1                  1       994.0
2       1                  1       994.0
3       1                  1       994.0
4       1                  1       994.0
2024-09-09 10:55:55,307 - INFO - Pairs created. Shape: (10000, 42)
2024-09-09 10:55:55,307 - INFO - Preparing features
2024-09-09 10:55:55,313 - INFO - Unique property tenure values: [1 0]
2024-09-09 10:55:55,314 - INFO - Unique tenure_preference values: [1 2 0]
2024-09-09 10:55:55,317 - INFO - Unique user tenure preference values: [1 2 0]
2024-09-09 10:55:55,323 - INFO - Creating target variable
2024-09-09 10:55:55,324 - INFO - Affordability condition met: 94.59%
2024-09-09 10:55:55,325 - INFO - Bedroom condition met: 100.00%
2024-09-09 10:55:55,325 - INFO - Price-to-income condition met: 72.81%
2024-09-09 10:55:55,325 - INFO - Size condition met: 100.00%
2024-09-09 10:55:55,325 - INFO - Tenure condition met: 100.00%
2024-09-09 10:55:55,326 - INFO - Positive samples ratio: 72.81%
2024-09-09 10:55:55,326 - INFO - Data preprocessing completed in 0.43 seconds
2024-09-09 10:55:55,343 - INFO - Scalers saved successfully.
2024-09-09 10:55:55,343 - INFO - Data loaded successfully. Training set shapes:
X_property: (8000, 33), X_user: (8000, 5)
2024-09-09 10:55:55,343 - INFO - Building the model
2024-09-09 10:55:55,375 - INFO - Starting model training
2024-09-09 10:55:57,172 - INFO - Epoch 1/200
```

```
2024-09-09 10:55:57,173 - INFO - Train loss: 9.0886, accuracy: 0.7513
2024-09-09 10:55:57,173 - INFO - Val loss: 6.9725, val_accuracy: 0.9055
2024-09-09 10:55:57,455 - INFO - Epoch 2/200
2024-09-09 10:55:57,455 - INFO - Train loss: 5.5420, accuracy: 0.8811
2024-09-09 10:55:57,455 - INFO - Val loss: 4.2356, val_accuracy: 0.9400
2024-09-09 10:55:57,736 - INFO - Epoch 3/200
2024-09-09 10:55:57,736 - INFO - Train loss: 3.4072, accuracy: 0.9093
2024-09-09 10:55:57,736 - INFO - Val loss: 2.6130, val_accuracy: 0.9580
2024-09-09 10:55:58,006 - INFO - Epoch 4/200
2024-09-09 10:55:58,006 - INFO - Train loss: 2.1694, accuracy: 0.9190
2024-09-09 10:55:58,007 - INFO - Val loss: 1.6903, val_accuracy: 0.9630
2024-09-09 10:55:58,302 - INFO - Epoch 5/200
2024-09-09 10:55:58,302 - INFO - Train loss: 1.4379, accuracy: 0.9291
2024-09-09 10:55:58,302 - INFO - Val loss: 1.1196, val_accuracy: 0.9695
2024-09-09 10:55:58,602 - INFO - Epoch 6/200
2024-09-09 10:55:58,603 - INFO - Train loss: 1.0034, accuracy: 0.9280
2024-09-09 10:55:58,603 - INFO - Val loss: 0.7960, val_accuracy: 0.9585
2024-09-09 10:55:58,912 - INFO - Epoch 7/200
2024-09-09 10:55:58,912 - INFO - Train loss: 0.7248, accuracy: 0.9317
2024-09-09 10:55:58,912 - INFO - Val loss: 0.5704, val_accuracy: 0.9625
2024-09-09 10:55:59,263 - INFO - Epoch 8/200
2024-09-09 10:55:59,264 - INFO - Train loss: 0.5485, accuracy: 0.9330
2024-09-09 10:55:59,264 - INFO - Val loss: 0.4253, val_accuracy: 0.9720
2024-09-09 10:55:59,574 - INFO - Epoch 9/200
2024-09-09 10:55:59,574 - INFO - Train loss: 0.4283, accuracy: 0.9389
2024-09-09 10:55:59,574 - INFO - Val loss: 0.3310, val_accuracy: 0.9705
2024-09-09 10:55:59,874 - INFO - Epoch 10/200
2024-09-09 10:55:59,874 - INFO - Train loss: 0.3492, accuracy: 0.9425
2024-09-09 10:55:59,874 - INFO - Val loss: 0.2840, val_accuracy: 0.9610
2024-09-09 10:56:00,176 - INFO - Epoch 11/200
2024-09-09 10:56:00,177 - INFO - Train loss: 0.2891, accuracy: 0.9470
2024-09-09 10:56:00,177 - INFO - Val loss: 0.2282, val_accuracy: 0.9720
2024-09-09 10:56:00,472 - INFO - Epoch 12/200
2024-09-09 10:56:00,473 - INFO - Train loss: 0.2600, accuracy: 0.9460
2024-09-09 10:56:00,473 - INFO - Val loss: 0.1921, val_accuracy: 0.9715
2024-09-09 10:56:00,764 - INFO - Epoch 13/200
2024-09-09 10:56:00,765 - INFO - Train loss: 0.2260, accuracy: 0.9539
2024-09-09 10:56:00,765 - INFO - Val loss: 0.1612, val_accuracy: 0.9850
2024-09-09 10:56:01,078 - INFO - Epoch 14/200
2024-09-09 10:56:01,079 - INFO - Train loss: 0.2045, accuracy: 0.9561
2024-09-09 10:56:01,079 - INFO - Val loss: 0.1646, val_accuracy: 0.9700
2024-09-09 10:56:01,386 - INFO - Epoch 15/200
2024-09-09 10:56:01,387 - INFO - Train loss: 0.1994, accuracy: 0.9517
2024-09-09 10:56:01,387 - INFO - Val loss: 0.1338, val_accuracy: 0.9820
2024-09-09 10:56:01,693 - INFO - Epoch 16/200
2024-09-09 10:56:01,693 - INFO - Train loss: 0.1838, accuracy: 0.9551
2024-09-09 10:56:01,693 - INFO - Val loss: 0.1331, val_accuracy: 0.9750
2024-09-09 10:56:02,002 - INFO - Epoch 17/200
```

```
2024-09-09 10:56:02,002 - INFO - Train loss: 0.1768, accuracy: 0.9574
2024-09-09 10:56:02,003 - INFO - Val loss: 0.1438, val_accuracy: 0.9665
2024-09-09 10:56:02,295 - INFO - Epoch 18/200
2024-09-09 10:56:02,295 - INFO - Train loss: 0.1749, accuracy: 0.9554
2024-09-09 10:56:02,296 - INFO - Val loss: 0.1179, val_accuracy: 0.9825
2024-09-09 10:56:02,589 - INFO - Epoch 19/200
2024-09-09 10:56:02,589 - INFO - Train loss: 0.1622, accuracy: 0.9600
2024-09-09 10:56:02,589 - INFO - Val loss: 0.1328, val_accuracy: 0.9665
2024-09-09 10:56:02,869 - INFO - Epoch 20/200
2024-09-09 10:56:02,870 - INFO - Train loss: 0.1566, accuracy: 0.9613
2024-09-09 10:56:02,870 - INFO - Val loss: 0.1065, val_accuracy: 0.9835
2024-09-09 10:56:03,176 - INFO - Epoch 21/200
2024-09-09 10:56:03,177 - INFO - Train loss: 0.1397, accuracy: 0.9660
2024-09-09 10:56:03,177 - INFO - Val loss: 0.1064, val_accuracy: 0.9760
2024-09-09 10:56:03,477 - INFO - Epoch 22/200
2024-09-09 10:56:03,478 - INFO - Train loss: 0.1486, accuracy: 0.9607
2024-09-09 10:56:03,478 - INFO - Val loss: 0.0995, val_accuracy: 0.9825
2024-09-09 10:56:03,773 - INFO - Epoch 23/200
2024-09-09 10:56:03,773 - INFO - Train loss: 0.1482, accuracy: 0.9614
2024-09-09 10:56:03,773 - INFO - Val loss: 0.0969, val_accuracy: 0.9855
2024-09-09 10:56:04,064 - INFO - Epoch 24/200
2024-09-09 10:56:04,064 - INFO - Train loss: 0.1419, accuracy: 0.9647
2024-09-09 10:56:04,064 - INFO - Val loss: 0.1022, val_accuracy: 0.9800
2024-09-09 10:56:04,361 - INFO - Epoch 25/200
2024-09-09 10:56:04,361 - INFO - Train loss: 0.1385, accuracy: 0.9646
2024-09-09 10:56:04,361 - INFO - Val loss: 0.1047, val_accuracy: 0.9745
2024-09-09 10:56:04,659 - INFO - Epoch 26/200
2024-09-09 10:56:04,659 - INFO - Train loss: 0.1418, accuracy: 0.9645
2024-09-09 10:56:04,660 - INFO - Val loss: 0.0936, val_accuracy: 0.9830
2024-09-09 10:56:04,961 - INFO - Epoch 27/200
2024-09-09 10:56:04,961 - INFO - Train loss: 0.1327, accuracy: 0.9654
2024-09-09 10:56:04,961 - INFO - Val loss: 0.0988, val_accuracy: 0.9830
2024-09-09 10:56:05,260 - INFO - Epoch 28/200
2024-09-09 10:56:05,260 - INFO - Train loss: 0.1378, accuracy: 0.9659
2024-09-09 10:56:05,260 - INFO - Val loss: 0.0924, val_accuracy: 0.9800
2024-09-09 10:56:05,580 - INFO - Epoch 29/200
2024-09-09 10:56:05,581 - INFO - Train loss: 0.1294, accuracy: 0.9654
2024-09-09 10:56:05,581 - INFO - Val loss: 0.0942, val_accuracy: 0.9810
2024-09-09 10:56:05,896 - INFO - Epoch 30/200
2024-09-09 10:56:05,897 - INFO - Train loss: 0.1263, accuracy: 0.9672
2024-09-09 10:56:05,897 - INFO - Val loss: 0.0878, val_accuracy: 0.9835
2024-09-09 10:56:06,198 - INFO - Epoch 31/200
2024-09-09 10:56:06,198 - INFO - Train loss: 0.1264, accuracy: 0.9695
2024-09-09 10:56:06,198 - INFO - Val loss: 0.0836, val_accuracy: 0.9870
2024-09-09 10:56:06,500 - INFO - Epoch 32/200
2024-09-09 10:56:06,500 - INFO - Train loss: 0.1291, accuracy: 0.9663
2024-09-09 10:56:06,501 - INFO - Val loss: 0.0802, val_accuracy: 0.9870
2024-09-09 10:56:06,796 - INFO - Epoch 33/200
```

```
2024-09-09 10:56:06,796 - INFO - Train loss: 0.1223, accuracy: 0.9712
2024-09-09 10:56:06,796 - INFO - Val loss: 0.0813, val_accuracy: 0.9865
2024-09-09 10:56:07,077 - INFO - Epoch 34/200
2024-09-09 10:56:07,077 - INFO - Train loss: 0.1244, accuracy: 0.9685
2024-09-09 10:56:07,077 - INFO - Val loss: 0.0936, val_accuracy: 0.9805
2024-09-09 10:56:07,355 - INFO - Epoch 35/200
2024-09-09 10:56:07,356 - INFO - Train loss: 0.1227, accuracy: 0.9680
2024-09-09 10:56:07,356 - INFO - Val loss: 0.0959, val_accuracy: 0.9795
2024-09-09 10:56:07,634 - INFO - Epoch 36/200
2024-09-09 10:56:07,634 - INFO - Train loss: 0.1241, accuracy: 0.9655
2024-09-09 10:56:07,634 - INFO - Val loss: 0.0775, val_accuracy: 0.9895
2024-09-09 10:56:07,912 - INFO - Epoch 37/200
2024-09-09 10:56:07,912 - INFO - Train loss: 0.1226, accuracy: 0.9682
2024-09-09 10:56:07,913 - INFO - Val loss: 0.0835, val_accuracy: 0.9820
2024-09-09 10:56:08,208 - INFO - Epoch 38/200
2024-09-09 10:56:08,208 - INFO - Train loss: 0.1151, accuracy: 0.9729
2024-09-09 10:56:08,208 - INFO - Val loss: 0.0856, val_accuracy: 0.9825
2024-09-09 10:56:08,515 - INFO - Epoch 39/200
2024-09-09 10:56:08,516 - INFO - Train loss: 0.1209, accuracy: 0.9688
2024-09-09 10:56:08,516 - INFO - Val loss: 0.0839, val_accuracy: 0.9795
2024-09-09 10:56:08,818 - INFO - Epoch 40/200
2024-09-09 10:56:08,819 - INFO - Train loss: 0.1200, accuracy: 0.9697
2024-09-09 10:56:08,819 - INFO - Val loss: 0.0707, val_accuracy: 0.9870
2024-09-09 10:56:09,133 - INFO - Epoch 41/200
2024-09-09 10:56:09,133 - INFO - Train loss: 0.1169, accuracy: 0.9680
2024-09-09 10:56:09,133 - INFO - Val loss: 0.0847, val_accuracy: 0.9845
2024-09-09 10:56:09,477 - INFO - Epoch 42/200
2024-09-09 10:56:09,478 - INFO - Train loss: 0.1134, accuracy: 0.9714
2024-09-09 10:56:09,478 - INFO - Val loss: 0.0793, val_accuracy: 0.9840
2024-09-09 10:56:09,766 - INFO - Epoch 43/200
2024-09-09 10:56:09,766 - INFO - Train loss: 0.1138, accuracy: 0.9721
2024-09-09 10:56:09,766 - INFO - Val loss: 0.0798, val_accuracy: 0.9840
2024-09-09 10:56:10,061 - INFO - Epoch 44/200
2024-09-09 10:56:10,062 - INFO - Train loss: 0.1137, accuracy: 0.9724
2024-09-09 10:56:10,062 - INFO - Val loss: 0.0745, val_accuracy: 0.9885
2024-09-09 10:56:10,357 - INFO - Epoch 45/200
2024-09-09 10:56:10,358 - INFO - Train loss: 0.1097, accuracy: 0.9728
2024-09-09 10:56:10,358 - INFO - Val loss: 0.0785, val_accuracy: 0.9820
2024-09-09 10:56:10,652 - INFO - Epoch 46/200
2024-09-09 10:56:10,652 - INFO - Train loss: 0.1104, accuracy: 0.9729
2024-09-09 10:56:10,653 - INFO - Val loss: 0.0865, val_accuracy: 0.9755
2024-09-09 10:56:10,946 - INFO - Epoch 47/200
2024-09-09 10:56:10,946 - INFO - Train loss: 0.1076, accuracy: 0.9736
2024-09-09 10:56:10,946 - INFO - Val loss: 0.0732, val_accuracy: 0.9870
2024-09-09 10:56:11,240 - INFO - Epoch 48/200
2024-09-09 10:56:11,240 - INFO - Train loss: 0.1103, accuracy: 0.9731
2024-09-09 10:56:11,241 - INFO - Val loss: 0.0810, val_accuracy: 0.9805
2024-09-09 10:56:11,527 - INFO - Epoch 49/200
```

```
2024-09-09 10:56:11,527 - INFO - Train loss: 0.1048, accuracy: 0.9739
2024-09-09 10:56:11,527 - INFO - Val loss: 0.0835, val_accuracy: 0.9790
2024-09-09 10:56:11,828 - INFO - Epoch 50/200
2024-09-09 10:56:11,829 - INFO - Train loss: 0.1034, accuracy: 0.9751
2024-09-09 10:56:11,829 - INFO - Val loss: 0.0868, val_accuracy: 0.9790
2024-09-09 10:56:11,833 - INFO - Model training completed
```