preliminary-report

June 17, 2024

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

1.1 1. Introduction

1.1.1 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.

1.1.2 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.

1.1.3 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.

1.1.4 1.4 Scope and Limitations

This prototype analyses various types of residential properties in Buckinghamshire, excluding shared ownership, retirement homes, new builds, auction listings, farms/land, park homes, and properties over £650,000. 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.

1.1.5 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 in the Buckinghamshire market. To achieve this, two main data sources were utilised: - **HM Land Registry Data**: This dataset provides comprehensive information on property sales in Buckinghamshire, including sale prices and transaction dates. - **Web Scraped Data from OnTheMarket.com**: A dataset comprising current property listings in Buckinghamshire, including asking prices, property types, and other relevant details.

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 in the UK. Accessed via gov.uk, under the Open Government Licence v3.0.
- **OnTheMarket.com**: Chosen as a current and active source for property listings, offering a real-time perspective on the market. Data was scraped in compliance with the site's terms and conditions, focusing on properties listed for sale in Buckinghamshire.

1.5.3 Methodology for Data Collection and Processing

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

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.

1.1.6 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.

word count: 837

2 Chapter 2: A Literature Review

2.1 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 2.2 Housing Price Prediction Models

2.2.1 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 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 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 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 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.

Word count chapter 2: 787

2.6 2.6 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]: 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

[3]: Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization 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

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

[5]: 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

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

3 Chapter 3: A Design

3.1 3.1 Project Overview

The Personalized 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 follows "Project Idea Title 1: Deep Learning on a Public Dataset" and aims to find the most effective model for property price prediction and recommendation using deep learning techniques.

3.2 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 homebuyers 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 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 personalized 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 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.
- **Web Scraping**: Scripts to collect real-time data from OnTheMarket.
- **Data Cleaning**: Handling missing values, standardizing formats, and filtering relevant data.
- **Data Integration**: Merging datasets to create a unified data source.

3.4.2 Model Development

- **Feature Selection**: Identifying relevant features such as price, location, property type, etc.
- **Model Selection**: Experimenting with various machine learning algorithms (e.g., C4.5, RIPPER, Naïve Bayesian, AdaBoost) to identify the best-performing model.
- **Training and Validation**: Splitting the data into training and validation sets, training the model, and evaluating its performance.

3.4.3 Recommendation System

- **User Profile Creation**: Collecting user preferences and financial information.
- **Similarity Calculation**: Using content-based filtering to match properties with user profiles.
- **Property Ranking**: Ranking properties based on their relevance to the user's preferences and budget.
- **Feedback Loop**: Incorporating synthetic user feedback to continuously improve the recommendation system.

3.5 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.
- **Node.js and Express.js**: For building the backend of the web application.
- **HTML, CSS, and JavaScript**: For developing the frontend of the web application.

3.5.2 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 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 3.7 Testing and Evaluation Plan

3.7.1 Testing

- **Unit Testing**: Test individual components (e.g., data collection scripts, model training functions) to ensure they work as expected.
- **Integration Testing**: Ensure that different components (e.g., data integration, model prediction, recommendation system) work together seamlessly.
- **Synthetic User Testing**: Simulate user interactions using synthetic data to evaluate the recommendation system's usability and effectiveness.

3.7.2 Evaluation

- **Model Evaluation**: Use MAE and RMSE to evaluate the accuracy of the price prediction model.
- **Synthetic User Feedback**: Use synthetic data to simulate user feedback regarding the relevance and usefulness of the property recommendations.
- **Performance Metrics**: Track the system's performance in terms of response time, accuracy, and user satisfaction.

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

4 Chapter 4: Feature Prototype

4.1 4.1 Introduction

Purpose of the feature prototype The feature prototype aims to demonstrate the feasibility of using machine learning techniques to predict property prices within the Personalised Property Recommendation System. This initial implementation focuses on developing a content-based filtering model to predict property prices based on historical transaction data from HM Land Registry and current property listings from OnTheMarket. Accurate price prediction is a crucial component of providing personalised property suggestions that align with users' budgets and preferences.

Role in demonstrating the feasibility of the project The development and evaluation of the price prediction model play several critical roles in demonstrating the feasibility of the Personalised Property Recommendation System:

1. **Proof of Concept**: Validates the potential of applying machine learning to real estate data.
2. **Budget Matching**: Helps filter and rank properties within user budgets.
3. **Market Price Estimation**: Provides insights into likely selling prices, aiding decision-making.
4. **Enhanced Personalization**: Integrates predicted prices with user preferences for sophisticated filtering and ranking.

By demonstrating these capabilities, the prototype showcases the potential for a comprehensive system that streamlines the property search process and tailors recommendations to individual needs.

4.2 4.2 Data Collection and Preprocessing

4.2.1 4.2.1 Data Collection Scripts

Introduction to Web Scraping Methodology To complement historical data from HM Land Registry, we web-scraped current property listings from OnTheMarket.com to gain insights into real-time market trends in Buckinghamshire.

Key Steps in Web Scraping Process

1. Identify Buckinghamshire property listing URLs.
2. Retrieve web content using Python's `requests`.
3. Extract data (prices, addresses, etc.) with `BeautifulSoup`.
4. Structure the extracted data.

[45]:
```
!pip install beautifulsoup4 lxml requests
!pip install ratelimit
!pip install tqdm
!pip install tensorflow scikit-learn pandas numpy matplotlib

# Add the directory containing custom modules to Python's import path
import sys
sys.path.append('./src/data-collector/')
sys.path.append('./src/data-cleanser/')
sys.path.append('./src/data-standardiser/')
```

```
Requirement already satisfied: beautifulsoup4 in ./venv/lib/python3.9/site-
packages (4.12.2)
Requirement already satisfied: lxml in ./venv/lib/python3.9/site-packages
(4.9.4)
Requirement already satisfied: requests in ./venv/lib/python3.9/site-packages
(2.31.0)
Requirement already satisfied: soupsieve>1.2 in ./venv/lib/python3.9/site-
packages (from beautifulsoup4) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
./venv/lib/python3.9/site-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in ./venv/lib/python3.9/site-
packages (from requests) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in ./venv/lib/python3.9/site-
packages (from requests) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in ./venv/lib/python3.9/site-
packages (from requests) (2023.11.17)
Requirement already satisfied: ratelimit in ./venv/lib/python3.9/site-packages
(2.2.1)
Requirement already satisfied: tqdm in ./venv/lib/python3.9/site-packages
(4.66.1)
Requirement already satisfied: tensorflow in ./venv/lib/python3.9/site-packages
(2.16.1)
Requirement already satisfied: scikit-learn in ./venv/lib/python3.9/site-
```

```
packages (1.5.0)
Requirement already satisfied: pandas in ./venv/lib/python3.9/site-packages
(2.0.3)
Requirement already satisfied: numpy in ./venv/lib/python3.9/site-packages
(1.24.4)
Requirement already satisfied: matplotlib in ./venv/lib/python3.9/site-packages
(3.7.4)
Requirement already satisfied: absl-py>=1.0.0 in ./venv/lib/python3.9/site-
packages (from tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in ./venv/lib/python3.9/site-
packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
./venv/lib/python3.9/site-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
./venv/lib/python3.9/site-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in ./venv/lib/python3.9/site-
packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in ./venv/lib/python3.9/site-
packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in ./venv/lib/python3.9/site-
packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.3.1 in ./venv/lib/python3.9/site-
packages (from tensorflow) (0.3.2)
Requirement already satisfied: opt-einsum>=2.3.2 in ./venv/lib/python3.9/site-
packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in ./venv/lib/python3.9/site-packages
(from tensorflow) (23.2)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in ./venv/lib/python3.9/site-packages (from tensorflow) (4.25.3)
Requirement already satisfied: requests<3,>=2.21.0 in ./venv/lib/python3.9/site-
packages (from tensorflow) (2.31.0)
Requirement already satisfied: setuptools in ./venv/lib/python3.9/site-packages
(from tensorflow) (67.6.1)
Requirement already satisfied: six>=1.12.0 in ./venv/lib/python3.9/site-packages
(from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in ./venv/lib/python3.9/site-
packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
./venv/lib/python3.9/site-packages (from tensorflow) (4.9.0)
Requirement already satisfied: wrapt>=1.11.0 in ./venv/lib/python3.9/site-
packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in ./venv/lib/python3.9/site-
packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.17,>=2.16 in
./venv/lib/python3.9/site-packages (from tensorflow) (2.16.2)
Requirement already satisfied: keras>=3.0.0 in ./venv/lib/python3.9/site-
packages (from tensorflow) (3.3.3)
```

```
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
./venv/lib/python3.9/site-packages (from tensorflow) (0.37.0)
Requirement already satisfied: scipy>=1.6.0 in ./venv/lib/python3.9/site-
packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in ./venv/lib/python3.9/site-
packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
./venv/lib/python3.9/site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
./venv/lib/python3.9/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./venv/lib/python3.9/site-
packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in ./venv/lib/python3.9/site-
packages (from pandas) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in ./venv/lib/python3.9/site-
packages (from matplotlib) (1.1.1)
Requirement already satisfied: cycler>=0.10 in ./venv/lib/python3.9/site-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in ./venv/lib/python3.9/site-
packages (from matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in ./venv/lib/python3.9/site-
packages (from matplotlib) (1.4.5)
Requirement already satisfied: pillow>=6.2.0 in ./venv/lib/python3.9/site-
packages (from matplotlib) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in ./venv/lib/python3.9/site-
packages (from matplotlib) (3.1.1)
Requirement already satisfied: importlib-resources>=3.2.0 in
./venv/lib/python3.9/site-packages (from matplotlib) (6.1.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in ./venv/lib/python3.9/site-
packages (from astunparse>=1.6.0->tensorflow) (0.40.0)
Requirement already satisfied: zipp>=3.1.0 in ./venv/lib/python3.9/site-packages
(from importlib-resources>=3.2.0->matplotlib) (3.17.0)
Requirement already satisfied: rich in ./venv/lib/python3.9/site-packages (from
keras>=3.0.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in ./venv/lib/python3.9/site-packages (from
keras>=3.0.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in ./venv/lib/python3.9/site-packages
(from keras>=3.0.0->tensorflow) (0.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
./venv/lib/python3.9/site-packages (from requests<3,>=2.21.0->tensorflow)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in ./venv/lib/python3.9/site-
packages (from requests<3,>=2.21.0->tensorflow) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in ./venv/lib/python3.9/site-
packages (from requests<3,>=2.21.0->tensorflow) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in ./venv/lib/python3.9/site-
packages (from requests<3,>=2.21.0->tensorflow) (2023.11.17)
Requirement already satisfied: markdown>=2.6.8 in ./venv/lib/python3.9/site-
```

```
packages (from tensorboard<2.17,>=2.16->tensorflow) (3.6)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
./venv/lib/python3.9/site-packages (from tensorboard<2.17,>=2.16->tensorflow)
(0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in ./venv/lib/python3.9/site-
packages (from tensorboard<2.17,>=2.16->tensorflow) (3.0.3)
Requirement already satisfied: importlib-metadata>=4.4 in
./venv/lib/python3.9/site-packages (from
markdown>=2.6.8->tensorboard<2.17,>=2.16->tensorflow) (7.0.1)
Requirement already satisfied: MarkupSafe>=2.1.1 in ./venv/lib/python3.9/site-
packages (from werkzeug>=1.0.1->tensorboard<2.17,>=2.16->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in
./venv/lib/python3.9/site-packages (from rich->keras>=3.0.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
./venv/lib/python3.9/site-packages (from rich->keras>=3.0.0->tensorflow)
(2.17.2)
Requirement already satisfied: mdurl~=0.1 in ./venv/lib/python3.9/site-packages
(from markdown-it-py>=2.2.0->rich->keras>=3.0.0->tensorflow) (0.1.2)
```

Modularity and Code Structure Our web scraping is structured into four main modules, each with a specific role. This modular approach ensures a clean separation of concerns, making the code more maintainable and scalable.

1. **`robot_check.py` - Respecting Site Policies**

Ensures compliance with the website's scraping policies by parsing and interpreting the robots.txt file using urllib.robotparser. This module is used by both crawler.py and data_collector_service.py.

[46]:
```python
# src/data-collector/robot_check.py
import urllib.robotparser


class RobotCheck:
    def __init__(self, robots_txt_url):
        self.parser = urllib.robotparser.RobotFileParser()
        self.parser.set_url(robots_txt_url)
        self.parser.read()

    def is_allowed(self, url, user_agent):
        return self.parser.can_fetch(user_agent, url)

    def get_crawl_delay(self, user_agent):
        delay = self.parser.crawl_delay(user_agent)
        return delay if delay is not None else 1
```

2. **`crawler.py` - Discovering URLs**

Crawls the target website, gathering relevant page URLs using `requests` and `BeautifulSoup`. Rate limiting is implemented with `ratelimit` to avoid overloading the server. This module is used by `data_collector_service.py`.

[47]:
```python
# src/data-collector/crawler.py
import time
import requests
from bs4 import BeautifulSoup
from robot_check import RobotCheck
from ratelimit import limits, sleep_and_retry


# 10 requests per minute
REQUESTS_PER_MINUTE = 10


@sleep_and_retry
@limits(calls=REQUESTS_PER_MINUTE, period=60)
def make_request(url, headers):
    return requests.get(url, headers=headers)


def get_property_urls(base_url, search_url, user_agent):
    headers = {'User-Agent': user_agent}
    robot_check = RobotCheck("https://www.onthemarket.com/robots.txt")
    property_urls = set()
    page_number = 1

    while search_url:
        if robot_check.is_allowed(search_url, user_agent):
            response = make_request(search_url, headers=headers)

            if response.status_code == 200:
                soup = BeautifulSoup(response.content, 'html.parser')
                links = soup.select('div.otm-PropertyCardMedia > div > a')
                current_page_urls = set()

                for link in links:
                    href = link.get('href')
                    if href and href.startswith('/details/'):  # Filtering␣
 ↪other links
                        full_url = 'https://www.onthemarket.com' + href
                        property_urls.add(full_url)
                        current_page_urls.add(full_url)

                # print(f"{len(current_page_urls)} URLs captured on Page␣
 ↪{page_number}.")

                # Find the next page URL
                next_page = soup.select_one('a[title="Next page"]')
                if next_page:
```

```
                    search_url = base_url + next_page['href']
                    page_number += 1
                else:
                    search_url = None

            else:
                print(f"Failed to retrieve the webpage: Status Code {response.
↪status_code}")
                break

            # Sleep for the crawl delay
            time.sleep(robot_check.get_crawl_delay(user_agent))
        else:
            print("Scraping is disallowed by the website's robots.txt for this␣
↪URL.")
            break

    return list(property_urls)
```

3. **scraper.py - Extracting Data**

Extracts specific data from web pages using requests and BeautifulSoup. This module is used by data_collector_service.py.

[48]:
```
# src/data-collector/scraper.py
import requests
from bs4 import BeautifulSoup


def scrape_property_details(property_url, headers, counter):
    try:
        response = requests.get(property_url, headers=headers)
        if response.status_code == 200:
            soup = BeautifulSoup(response.content, 'html.parser')

            # Extracting data
            # Extracting the title
            title = soup.find('h1', class_='h4 md:text-xl leading-normal').
↪get_text(strip=True) if soup.find('h1', class_='h4 md:text-xl␣
↪leading-normal') else 'Title not available'

            # Extracting the address
            address_div = soup.find('div', class_='text-slate h4 font-normal␣
↪leading-none font-heading')
            address = address_div.get_text(strip=True) if address_div else␣
↪'Address not available'

            # Extracting the price
```

```
        price_div = soup.find('div', class_='otm-Price')
        if price_div:
            price = price_div.find('span', class_='price').
↪get_text(strip=True) if price_div.find('span', class_='price') else 'Price␣
↪not available'

            # Extracting the pricing qualifier if present
            qualifier_div = price_div.find('small', class_='qualifier')
            price_qualifier = qualifier_div.get_text(strip=True) if␣
↪qualifier_div else 'Price qualifier not available'
        else:
            price = 'Price not available'
            price_qualifier = 'Price qualifier not available'

        # Extracting the listing time
        listing_time_div = soup.find('div', class_='text-denim')
        if listing_time_div:
            listing_time = listing_time_div.find('small' ,␣
↪class_='font-heading').get_text(strip=True) if listing_time_div.find(
                'small') else 'Listing time not available'
        else:
            listing_time = 'Listing time not available'

        # Extracting the property type
        property_type_div = soup.find('div', class_='otm-PropertyIcon')
        property_type = property_type_div.get_text(
            strip=True) if property_type_div else 'Property type not␣
↪available'

        details_div = soup.find('div', class_='font-heading text-xs flex␣
↪flex-wrap border-t border-b mb-6 md:text-md py-3 md:py-4 md:mb-9')
        if details_div:
            # Extracting the number of bedrooms
            bedrooms = 'Bedrooms info not available'
            for div in details_div.find_all('div'):
                if 'bed' in div.get_text(strip=True).lower():
                    bedrooms = div.get_text(strip=True)
                    break

            # Extracting the number of bathrooms
            bathrooms = 'Bathrooms info not available'
            for div in details_div.find_all('div'):
                if 'bath' in div.get_text(strip=True).lower():
                    bathrooms = div.get_text(strip=True)
                    break

            # Extracting the EPC rating
```

```
                epc_rating = 'EPC rating not available'
                for div in details_div.find_all('div'):
                    if 'epc rating' in div.get_text(strip=True).lower():
                        epc_rating = div.get_text(strip=True)
                        break

                # Extracting the property size
                size = 'Size info not available'
                for div in details_div.find_all('div'):
                    text = div.get_text(strip=True).lower()
                    if 'sq ft' in text or 'sq m' in text:
                        size = div.get_text(strip=True)
                        break

            # Extracting features
            features_section = soup.find('section', class_='otm-FeaturesList')
            features = []
            if features_section:
                feature_items = features_section.find_all('li', class_='otm-ListItemOtmBullet')
                for item in feature_items:
                    feature_text = item.get_text(strip=True)
                    features.append(feature_text)

            return {
                'id': counter,
                'property_url': property_url,
                'title': title,
                'address': address,
                'price': price,
                'pricing_qualifier': price_qualifier,
                'listing_time': listing_time,
                'property_type': property_type,
                'bedrooms': bedrooms,
                'bathrooms': bathrooms,
                'epc_rating': epc_rating,
                'size': size,
                'features': features
            }
        else:
            print(f"Failed to retrieve the property page: Status Code {response.status_code}")
            return {}
    except requests.exceptions.RequestException as e:
        return {'error': f"Request failed: {e}"}
    except Exception as e:
        return {'error': f"An unexpected error occurred: {e}"}
```

4. **`data_collector_service.py` - Orchestrating the Scraping Process**

Orchestrates the entire web scraping process. It calls on `crawler.py` to get URLs and then uses `scraper.py` to extract data from them. Handles errors, saves the data (usually in JSON), and can be run from the command line or Jupyter Notebook with adjustments.

[49]:

```python
# src/data-collector/data_colletor_service.py
import json
from crawler import get_property_urls
from scraper import scrape_property_details
from robot_check import RobotCheck
from tqdm.notebook import tqdm
import time
import sys
import os

class DataCollectorService:
    def __init__(self, base_url, user_agent, max_price=120000):
        self.base_url = base_url
        self.user_agent = user_agent
        self.max_price = max_price
        self.robot_check = RobotCheck(f"{base_url}/robots.txt")

    @staticmethod
    def generate_price_segments(max_price, segment_size=100000):
        segments = []
        current_min = 0
        while current_min < max_price:
            current_max = min(current_min + segment_size, max_price)
            segments.append((current_min, current_max))
            current_min = current_max + 1
        return segments

    def collect_data_segment(self, min_price, max_price):
        search_url = f"{self.base_url}/for-sale/property/buckinghamshire/?
↪auction=false&min-price={min_price}&max-price={max_price}&new-home-flag=F&prop-types=bungal
        return get_property_urls(self.base_url, search_url, self.user_agent)

    def collect_data(self):
        price_segments = self.generate_price_segments(self.max_price)
        all_property_urls = set()
        all_property_data = []
        counter = 1

        for min_price, max_price in price_segments:
            segment_urls = self.collect_data_segment(min_price, max_price)
            all_property_urls.update(segment_urls)
```

```
            print(f"Segment {min_price}-{max_price}: Found {len(segment_urls)}␣
↪URLs.")

        print(f"Found a total of {len(all_property_urls)} property URLs from␣
↪all segments.")

        for url in tqdm(all_property_urls, desc="Scraping properties"):
            if self.robot_check.is_allowed(url, self.user_agent):
                headers = {'User-Agent': self.user_agent}
                data = scrape_property_details(url, headers, counter)
                if 'error' in data:
                    print(data['error'])
                else:
                    all_property_data.append(data)
                    counter += 1
                    # Respect the crawl delay
                    time.sleep(self.robot_check.get_crawl_delay(self.
↪user_agent))
            else:
                tqdm.write(f"Skipping {url}, disallowed by robots.txt.")

        self.save_data(all_property_data)

    def save_data(self, data):
        data_directory = './data'
        filename = f'./data/property_data_{self.max_price}.json'
        backup_filename = f'./data/property_data_{self.max_price}-backup.json'

        try:
            # Ensure data directory exists
            if not os.path.exists(data_directory):
                os.makedirs(data_directory)
                print(f"Created directory {data_directory}")

            # Rename existing files if they exist
            if os.path.exists(backup_filename):
                os.remove(backup_filename)
            if os.path.exists(filename):
                os.rename(filename, backup_filename)
                print(f"Renamed existing file to {backup_filename}")

            # Save the new data
            with open(filename, 'w') as file:
                json.dump(data, file, indent=4)
            print(f"Data saved successfully in {filename}")
        except IOError as e:
            print(f"An IOError occurred while saving data: {e}")
```

```
        except Exception as e:
            print(f"An unexpected error occurred while saving data: {e}")

if __name__ == "__main__":
    max_price_arg = 120000
    base_url = 'https://www.onthemarket.com'
    search_url = base_url + '/for-sale/property/buckinghamshire/?
 ↪auction=false&max-price=500000&min-price=10000&new-home-flag=F&prop-types=bungalows&prop-ty
    user_agent = 'StudentDataProjectScraper/1.0 (Contact: gor5@student.london.
 ↪ac.uk)'
    service = DataCollectorService(base_url, user_agent, max_price_arg)
    service.collect_data()
```

```
Segment 0-100000: Found 0 URLs.
Segment 100001-120000: Found 8 URLs.
Found a total of 8 property URLs from all segments.

Scraping properties:   0%|          | 0/8 [00:00<?, ?it/s]

Renamed existing file to ./data/property_data_120000-backup.json
Data saved successfully in ./data/property_data_120000.json
```

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

Note on Script Execution Time Due to our ethical scraping approach, the script's execution takes longer for larger datasets, particularly given the rate limits and crawl delays we adhere to. This ensures responsible scraping while avoiding potential blocking by the website.

We can run `data_collector_service.py` from the CLI or from the Jupyter Notebook using `%run` and provide the `max_price` argument to control the data collection scope. For this prototype, data was collected for properties priced up to £650,000. This data can be found in the `data` folder

[50]:
```
# %run src/data-collector/data_collector_service.py 650000
%run src/data-collector/data_collector_service.py 120000
```

```
Segment 0-100000: Found 0 URLs.
Segment 100001-120000: Found 8 URLs.
Found a total of 8 property URLs from all segments.

Scraping properties:   0%|          | 0/8 [00:00<?, ?it/s]

Renamed existing file to ./data/property_data_120000-backup.json
Data saved successfully in ./data/property_data_120000.json
```

Displaying Scraped Data Verifying the integrity and structure of the scraped data is essential to ensure it aligns with our requirements and is ready for analysis.

[51]:
```python
import pandas as pd

# Load the scraped data from the JSON file
file_path = './data/property_data_650000.json'
scraped_data = pd.read_json(file_path)

# Display the first 5 to 10 rows of the dataset
scraped_data.head(10)
```

[51]:
```
   id                                     property_url  \
0   1  https://www.onthemarket.com/details/14543582/
1   2  https://www.onthemarket.com/details/14855003/
2   3  https://www.onthemarket.com/details/15004462/
3   4  https://www.onthemarket.com/details/14302804/
4   5  https://www.onthemarket.com/details/14963433/
5   6  https://www.onthemarket.com/details/14371850/
6   7  https://www.onthemarket.com/details/13238957/
7   8  https://www.onthemarket.com/details/14214029/
8   9  https://www.onthemarket.com/details/14381566/
9  10  https://www.onthemarket.com/details/14139162/

                                      title  \
0       3 bedroom terraced house for sale
1  3 bedroom semi-detached house for sale
2            2 bedroom apartment for sale
3  3 bedroom semi-detached house for sale
4           2 bedroom maisonette for sale
5       3 bedroom terraced house for sale
6            2 bedroom apartment for sale
7       3 bedroom detached house for sale
8       4 bedroom detached house for sale
9            1 bedroom apartment for sale

                                        address     price  \
0        Green Lane, Wolverton, Milton Keynes  £385,000
1  Bernay Gardens, Bolbeck Park, Milton Keynes  £429,995
2                        Aylesbury, Aylesbury HP19  £215,000
3                              Fox Way, Buckingham  £379,995
4                  Hillary Close, Aylesbury HP21  £265,000
5                     WELLINGBOROUGH ROAD, OLNEY  £595,000
6             Grange Road, Chalfont St Peter SL9  £385,000
7                   Farmers Way, Seer Green, HP9  £620,000
8                                  FOXHILL, OLNEY  £475,000
9                        Maypole Road, Taplow SL6  £185,000

             pricing_qualifier     listing_time       property_type  \
0  Price qualifier not available  Added > 14 days      Terraced house
```

```
1  Price qualifier not available  Added > 14 days  Semi-detached house
2                    Guide price  Added < 14 days            Apartment
3  Price qualifier not available  Added > 14 days  Semi-detached house
4  Price qualifier not available  Added > 14 days           Maisonette
5                    Guide price  Added > 14 days       Terraced house
6  Price qualifier not available  Added > 14 days            Apartment
7                    Guide price  Added > 14 days       Detached house
8                    Offers over  Added > 14 days       Detached house
9            Offers in excess of  Added > 14 days            Apartment

  bedrooms bathrooms                epc_rating                    size  \
0    3 bed     2bath            EPC rating: E*  1,400 sq ft / 130 sq m
1    3 bed     2bath            EPC rating: C*     947 sq ft / 88 sq m
2    2 bed     1bath            EPC rating: D*     538 sq ft / 50 sq m
3    3 bed     1bath            EPC rating: C*     904 sq ft / 84 sq m
4    2 bed     1bath            EPC rating: C*     592 sq ft / 55 sq m
5    3 bed     2bath            EPC rating: D*  1,334 sq ft / 124 sq m
6    2 bed     1bath            EPC rating: B*     753 sq ft / 70 sq m
7    3 bed     2bath  EPC rating not available  1,142 sq ft / 106 sq m
8    4 bed     2bath            EPC rating: C*  1,194 sq ft / 111 sq m
9    1 bed     1bath            EPC rating: D*     365 sq ft / 34 sq m

                                            features
0  [THREE DOUBLE BEDROOMS, VICTORIAN TERRACED, PE…
1  [En-suite shower room, Stylish refitted kitche…
2  [Tenure: Leasehold, TWO BEDROOMS, ON SITE GYM,…
3  [Three Bedroom Semi Detached, Corner Plot, Dri…
4  [Southside Of Aylesbury, Two Double Bedrooms, …
5  [Tenure: Freehold, DOUBLE FRONTED PROPERTY, CL…
6  [Tenure: Leasehold, Beautifully Presented Thro…
7  [Tenure: Freehold, Garage, Garden, Walk to sta…
8  [Tenure: Freehold, FOUR BEDROOM DETACHED FAMIL…
9  [Tenure: Leasehold, One bedroom ground floor a…
```

4.2.2 4.2.2 Initial Data Exploration and Analysis

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 filtered it to focus on transactions in Buckinghamshire.

[52]:
```
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)
```

```python
print("First 10 rows of the data are:")
df.head(10)
```

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

```
[52]:    {09266DDB-86DB-AF90-E063-4704A8C02087}  323450  2021-11-30 00:00  BS13 0FF  \
0  {09266DDB-86EF-AF90-E063-4704A8C02087}  420000  2021-05-26 00:00   BS2 9NY
1  {09266DDB-86F5-AF90-E063-4704A8C02087}  185000  2021-10-22 00:00  BS14 0TL
2  {09266DDB-86F8-AF90-E063-4704A8C02087}  269000  2021-12-17 00:00  BS13 7DL
3  {09266DDB-8738-AF90-E063-4704A8C02087}  520000  2021-09-23 00:00   BS2 9XB
4  {09266DDB-86A8-AF90-E063-4704A8C02087}  195000  2021-09-29 00:00   BS4 5AA
5  {09266DDB-86A9-AF90-E063-4704A8C02087}  245000  2021-11-29 00:00   BS7 8BW
6  {09266DDB-86AA-AF90-E063-4704A8C02087}  204995  2021-10-08 00:00   BS4 1FN
7  {09266DDB-86AB-AF90-E063-4704A8C02087}  265000  2021-12-09 00:00   BS9 3UL
8  {09266DDB-86AD-AF90-E063-4704A8C02087}  194995  2021-12-10 00:00  BS16 2GZ
9  {09266DDB-86AE-AF90-E063-4704A8C02087}  185950  2021-05-26 00:00   BS5 9HH

   S  Y  F             23 Unnamed: 8        BUTTERFLY LANE Unnamed: 10  \
0  T  Y  F            201        NaN      NEWFOUNDLAND ROAD         NaN
1  F  Y  L  BOULEVARD VIEW        22        WHITCHURCH LANE         NaN
2  F  Y  L             84        NaN  OLD PUMP HOUSE CLOSE         NaN
3  T  N  F              7        NaN         NARROWAYS ROAD         NaN
4  F  N  L            41A        NaN           BRISTOL HILL         NaN
5  F  N  L             12     FLAT 2            BOLTON ROAD         NaN
6  F  N  L             66     FLAT 4           MANNING ROAD         NaN
7  F  N  L    LITTLE COTE     FLAT 3              COTE LANE         NaN
8  F  N  L             14        NaN       NAPOLEON AVENUE   FISHPONDS
9  F  N  L           124A        NaN            CHURCH ROAD   REDFIELD

   BRISTOL  CITY OF BRISTOL CITY OF BRISTOL.1  A A.1
0  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
1  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
2  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
3  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
4  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
5  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
6  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
7  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
8  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
9  BRISTOL  CITY OF BRISTOL   CITY OF BRISTOL  A   A
```

4.2.3 4.2.3 Cleaning and Preparing Data

Data Cleaning Methodology Our data cleaning methodology, using tools like pandas, involves filtering, standardising, and handling missing/erroneous data from HM Land Registry and OnTheMarket.com for our Buckinghamshire property market analysis.

Implementing Data Cleaning with `data_cleanser_service.py` The `data_cleanser_service.py` script is a key component in refining raw HM Land Registry data for our Buckinghamshire property market analysis.

Key Features of `data_cleanser_service.py`:

- **Header Assignment**: The script assigns column headers to the HM Land Registry dataset (`pp-monthly-update-new-version.csv`) based on definitions from their website (https://www.gov.uk/guidance/about-the-price-paid-data).
- **Loading and Structuring Data**: Loads the CSV data into a `pandas` DataFrame.
- **Date Conversion and Filtering**: Retains only properties located in Buckinghamshire.

[53]:
```python
# data-cleanser/data_cleanser_service.py
import pandas as pd

def cleanse_data(input_file, output_file):
    # Define the headers based on the provided breakdown
    headers = ["Unique Transaction Identifier", "Price", "Date of Transaction",
               "Postal Code", "Property Type", "Old/New", "Duration",
               "PAON", "SAON", "Street", "Locality", "Town/City",
               "District", "County", "PPD Category Type", "Record Status"]

    # Load the CSV file without headers
    data = pd.read_csv(input_file, header=None, names=headers)

    # Convert Date of Transaction to datetime for filtering
    print("Converting dates and filtering data...")
    data['Date of Transaction'] = pd.to_datetime(data['Date of Transaction'])

    # Filter for properties in Buckinghamshire and from the year 2023
    data['Date of Transaction'] = pd.to_datetime(data['Date of Transaction'])
    filtered_data = data[(data['County'].str.upper() == 'BUCKINGHAMSHIRE') &
                         (data['Date of Transaction'].dt.year == 2023)]

    print(f"Number of records after filtering: {len(filtered_data)}")

    print("Saving cleaned data to CSV file...")
    # Save the cleaned data to a new CSV file
    filtered_data.to_csv(output_file, index=False)
    print("Data cleansing process completed successfully.")


# File paths
input_csv = './data/historical-data/pp-monthly-update-new-version.csv'  #␣
 ↪Update with actual path
output_csv = './data/historical-data/buckinghamshire_2023_cleaned_data.csv'  #␣
 ↪Update with desired output path
```

```
cleanse_data(input_csv, output_csv)
```

```
Converting dates and filtering data…
Number of records after filtering: 438
Saving cleaned data to CSV file…
Data cleansing process completed successfully.
```

4.2.4 Enhancing Data with Geocoding and Merging using **`data_standardiser_service.py`**

`data_standardiser_service.py` enhances and merges datasets:

Key Features of `data_standardiser_service.py`: 1. **Geocoding**: Uses Nominatim and ArcGIS for rate-limited geocoding, with a fallback mechanism if one fails. 2. **Price Standardisation**: Converts various price formats into a uniform numerical format. 3. **Address Normalization**: Standardizes addresses for consistency. 4. **Dataset Merging**: Combines the processed scraped and registry data into a single DataFrame. 5. **Saving & Updating**: Saves the enriched dataset and updates existing data. 6. **Property Type Classification**: Ensures consistency across the dataset. 7. **Updating Existing Preprocessed Data**: Classifies scraped data rows according to the registry data classification system.

[54]:
```
# data-standardiser/data_standariser_service.py but with relative file path␣
↪changed
import pandas as pd
import os
from datetime import datetime
from geopy.geocoders import Nominatim, ArcGIS
from geopy.extra.rate_limiter import RateLimiter
from geopy.exc import GeocoderTimedOut, GeocoderQuotaExceeded
import time

# Initialize Nominatim API
geolocator = Nominatim(user_agent="StudentDataProjectScraper/1.0 (Contact:␣
↪gor5@student.london.ac.uk)")
geolocator_arcgis = ArcGIS(user_agent="StudentDataProjectScraper/1.0 (Contact:␣
↪gor5@student.london.ac.uk)")

# Rate limiter to avoid overloading the API
geocode = RateLimiter(geolocator.geocode, min_delay_seconds=2)
geocode_arcgis = RateLimiter(geolocator_arcgis.geocode, min_delay_seconds=2)

# Mapping for converting scraped property types to registry property types
scraped_to_registry_property_type_mapping = {
    'Apartment': 'F',
    'Barn conversion': 'O',
    'Block of apartments': 'F',
    'Bungalow': 'D',
    'Character property': 'O',
    'Cluster house': 'O',
```

```python
    'Coach house': 'F',
    'Cottage': 'D',
    'Detached bungalow': 'D',
    'Detached house': 'D',
    'Duplex': 'F',
    'End of terrace house': 'T',
    'Equestrian property': 'O',
    'Farm house': 'O',
    'Flat': 'F',
    'Ground floor flat': 'F',
    'Ground floor maisonette': 'F',
    'House': 'D',
    'Link detached house': 'D',
    'Lodge': 'O',
    'Maisonette': 'F',
    'Mews': 'O',
    'Penthouse': 'F',
    'Semi-detached bungalow': 'D',
    'Semi-detached house': 'S',
    'Studio': 'F',
    'Terraced house': 'T',
    'Townhouse': 'D'
}

def geocode_address(address):
    try:
        location = geocode(address)
        if location:
            print(f"Geocoded '{address}': Latitude {location.latitude}, 
 ↪Longitude {location.longitude}")
            return location.latitude, location.longitude
        else:
            # Fallback to ArcGIS if Nominatim fails
            location = geocode_arcgis(address)
            if location:
                print(f"Geocoded '{address}': Latitude {location.latitude}, 
 ↪Longitude {location.longitude}")
                return location.latitude, location.longitude
            else:
                print(f"No result for '{address}'")
                return None, None
    except GeocoderQuotaExceeded:
        print("Quota exceeded for geocoding API")
        return None, None
    except GeocoderTimedOut:
        print("Geocoding API timed out")
        return None, None
```

```python
    except Exception as e:
        print(f"Error geocoding {address}: {e}")
        return None, None

def check_preprocessed_file(file_path):
    """Check if the preprocessed file exists and has latitude and longitude."""
    if os.path.exists(file_path):
        df = pd.read_csv(file_path)
        if 'latitude' in df.columns and 'longitude' in df.columns:
            if df[['latitude', 'longitude']].notnull().all().all():
                # File exists and latitude and longitude are filled
                return True
    return False

def standardise_price(price):
    """
    Convert a price string to a numerical value.
    Handles strings like '£275,000' and converts them to 275000.
    """
    if not isinstance(price, str):
        return price  # If it's already a number, return as-is

    # Removing currency symbols and commas
    price = price.replace('£', '').replace(',', '').replace('€', '').strip()

    try:
        # Convert to float or int
        price_value = float(price) if '.' in price else int(price)
    except ValueError:
        # Handle cases where conversion fails
        print(f"Warning: Could not convert price '{price}' to a number.")
        price_value = None

    return price_value

def normalize_address_scraped(address):
    """
    Normalize addresses from the scraped data.
    """
    # Assuming the county is always 'Buckinghamshire' if not specified
    if 'Buckinghamshire' not in address:
        address += ', Buckinghamshire'
    return address.strip()

def normalize_address_land_registry(row):
    # Convert each component to a string to avoid TypeError
    components = [
```

```
        str(row['Street']),
        str(row['Locality']),
        str(row['Town/City']),
        str(row['District']),
        str(row['County'])
    ]
    # Join the non-empty components
    return ', '.join(filter(None, components))

# Read JSON, standardize price, normalize address, add source column

def read_and_process_scraped_data(scraped_file_path, skip_geocoding):
    # Read the scraped data
    scraped_data = pd.read_json(scraped_file_path)
    scraped_data['price'] = scraped_data['price'].apply(standardise_price)
    scraped_data['normalized_address'] = scraped_data['address'].
 ↪apply(normalize_address_scraped)
    scraped_data['source'] = 'scraped'

    if not skip_geocoding:
        lat_long = scraped_data['normalized_address'].apply(geocode_address)
        scraped_data['latitude'] = lat_long.apply(lambda x: x[0] if x else None)
        scraped_data['longitude'] = lat_long.apply(lambda x: x[1] if x else␣
 ↪None)

    return scraped_data

def read_and_process_registry_data(registry_file_path, skip_geocoding):
    registry_data = pd.read_csv(registry_file_path)
    registry_data['Price'] = registry_data['Price'].apply(standardise_price)
    registry_data['normalized_address'] = registry_data.
 ↪apply(normalize_address_land_registry, axis=1)
    registry_data.rename(columns={'Price': 'price'}, inplace=True)
    registry_data['source'] = 'registry'

    if not skip_geocoding:
        lat_long = registry_data['normalized_address'].apply(geocode_address)
        registry_data['latitude'] = lat_long.apply(lambda x: x[0] if x else␣
 ↪None)
        registry_data['longitude'] = lat_long.apply(lambda x: x[1] if x else␣
 ↪None)

    return registry_data

def update_date_column(df, source_column, new_date):
    """
    Update 'Date' column in the DataFrame based on source.
```

```python
    """
    df['Date'] = pd.NaT
    df.loc[df['source'] == 'registry', 'Date'] = pd.to_datetime(df[source_column])
    df.loc[df['source'] == 'scraped', 'Date'] = new_date
    return df

def merge_price_columns(df):
    """
    Merge 'price' and 'Price' columns and update 'Price' with 'price' values for scraped data.
    """
    # Use 'price' from scraped data if it is not NaN, else use 'Price' from registry data
    df['Price'] = df.apply(lambda x: x['price'] if pd.notna(x['price']) else x['Price'], axis=1)
    return df

def apply_property_type_mapping(df):
    """
    Apply property type mapping to the DataFrame, only if 'property_type' is not null.
    """
    # Apply mapping only where 'property_type' is not null
    mask = df['property_type'].notnull()
    df.loc[mask, 'Property Type'] = df.loc[mask, 'property_type'].map(scraped_to_registry_property_type_mapping)
    return df

def process_and_save_data(scraped_data, registry_data, output_file_path):
    """
    Process and save merged data.
    """
    # Merge datasets
    merged_data = pd.concat([scraped_data, registry_data], ignore_index=True)

    # Update the date column
    merged_data = update_date_column(merged_data, 'Date of Transaction', datetime(2023, 12, 31))

    # Apply property type mapping
    merged_data = apply_property_type_mapping(merged_data)

    # Merge 'price' and 'Price' columns
    merged_data = merge_price_columns(merged_data)
```

```python
    # Save merged data
    merged_data.to_csv(output_file_path, index=False)
    print(f"Merged data saved successfully to '{output_file_path}'.")

def main():
    scraped_file = './data/property_data_650000.json'
    registry_file = './data/historical-data/buckinghamshire_2023_cleaned_data.
 ↪csv'
    output_file = './data/preprocessed-data/preprocessed.csv'

    if check_preprocessed_file(output_file):
        # Read existing preprocessed data
        preprocessed_data = pd.read_csv(output_file)
        print(f"Using existing preprocessed data from '{output_file}'.")

        # Update the date column in the existing data
        preprocessed_data = update_date_column(preprocessed_data, 'Date of␣
 ↪Transaction', datetime(2023, 12, 31))

        # Apply property type mapping
        preprocessed_data = apply_property_type_mapping(preprocessed_data)

        # Apply property type mapping and merge price columns
        preprocessed_data = merge_price_columns(preprocessed_data)

        # Save updated data
        preprocessed_data.to_csv(output_file, index=False)
        print(f"Updated data saved successfully to '{output_file}'.")
    else:
        # Process new data
        scraped_data = read_and_process_scraped_data(scraped_file, False)
        registry_data = read_and_process_registry_data(registry_file, False)

        # Process and save merged data
        process_and_save_data(scraped_data, registry_data, output_file)

    print("Data processing completed.")

if __name__ == "__main__":
    main()
```

```
Using existing preprocessed data from './data/preprocessed-
data/preprocessed.csv'.
Updated data saved successfully to './data/preprocessed-data/preprocessed.csv'.
Data processing completed.
```

The dataset below combines web-scraped data (December 2023) and HM Land Registry data (2023) for Buckinghamshire properties. The first and last five entries are shown, summarising the dataset's structure and content.

[55]:
```
import pandas as pd

# Path to your CSV file
file_path = './data/preprocessed-data/preprocessed.csv'

# Read the CSV file
df = pd.read_csv(file_path)

# Display the first 5 rows
print("First 5 rows of the data are:")
print(df.head(5))

# Display the last 5 rows
print("\nLast 5 rows of the data are:")
print(df.tail(5))

# Print the column names
print("Column names:")
print(df.columns)
```

```
First 5 rows of the data are:
    id                                  property_url  \
0  1.0  https://www.onthemarket.com/details/13262643/
1  2.0  https://www.onthemarket.com/details/14047885/
2  3.0  https://www.onthemarket.com/details/14068644/
3  4.0  https://www.onthemarket.com/details/13944473/
4  5.0  https://www.onthemarket.com/details/13422734/

                                    title  \
0           1 bedroom apartment for sale
1           2 bedroom apartment for sale
2  4 bedroom semi-detached house for sale
3                1 bedroom flat for sale
4                3 bedroom flat for sale

                                 address     price  \
0          The Green, High Wycombe HP10  275000.0
1        Aylesbury, Buckinghamshire HP21  220000.0
2     Edzell Crescent, Milton Keynes MK4  465000.0
3                   Scafell Road, Slough  195000.0
4  High Wycombe,  Buckinghamshire,  HP13  265000.0

               pricing_qualifier           listing_time        property_type  \
0  Price qualifier not available  OnTheMarket > 14 days            Apartment
1  Price qualifier not available  OnTheMarket > 14 days            Apartment
2  Price qualifier not available  OnTheMarket < 14 days  Semi-detached house
```

```
3                     Guide price  OnTheMarket > 14 days                 Flat
4  Price qualifier not available  OnTheMarket > 14 days                 Flat

  bedrooms bathrooms  … PAON SAON Street Locality Town/City  District  \
0    1 bed     1bath  …  NaN  NaN    NaN      NaN       NaN       NaN
1    2 bed     2bath  …  NaN  NaN    NaN      NaN       NaN       NaN
2    4 bed     2bath  …  NaN  NaN    NaN      NaN       NaN       NaN
3    1 bed     1bath  …  NaN  NaN    NaN      NaN       NaN       NaN
4    3 bed     1bath  …  NaN  NaN    NaN      NaN       NaN       NaN

  County PPD Category Type  Record Status        Date
0    NaN               NaN            NaN  2023-12-31
1    NaN               NaN            NaN  2023-12-31
2    NaN               NaN            NaN  2023-12-31
3    NaN               NaN            NaN  2023-12-31
4    NaN               NaN            NaN  2023-12-31

[5 rows x 34 columns]

Last 5 rows of the data are:
      id property_url title address  price pricing_qualifier listing_time  \
2595 NaN          NaN   NaN     NaN    NaN               NaN          NaN
2596 NaN          NaN   NaN     NaN    NaN               NaN          NaN
2597 NaN          NaN   NaN     NaN    NaN               NaN          NaN
2598 NaN          NaN   NaN     NaN    NaN               NaN          NaN
2599 NaN          NaN   NaN     NaN    NaN               NaN          NaN

     property_type bedrooms bathrooms  …          PAON SAON  \
2595           NaN      NaN       NaN  …            27  NaN
2596           NaN      NaN       NaN  …  PEATEY COURT  128
2597           NaN      NaN       NaN  …             2  NaN
2598           NaN      NaN       NaN  …             4  NaN
2599           NaN      NaN       NaN  …           129  NaN

                 Street          Locality     Town/City         District  \
2595          DELLFIELD               NaN       CHESHAM  BUCKINGHAMSHIRE
2596       PRINCES GATE               NaN  HIGH WYCOMBE  BUCKINGHAMSHIRE
2597  YEAT FARM COTTAGES  WOTTON UNDERWOOD     AYLESBURY  BUCKINGHAMSHIRE
2598        BADGERS RISE             STONE     AYLESBURY  BUCKINGHAMSHIRE
2599         COULSON WAY           BURNHAM        SLOUGH  BUCKINGHAMSHIRE

               County PPD Category Type  Record Status        Date
2595  BUCKINGHAMSHIRE                 A              D  2023-03-23
2596  BUCKINGHAMSHIRE                 A              D  2023-03-25
2597  BUCKINGHAMSHIRE                 A              D  2023-05-22
2598  BUCKINGHAMSHIRE                 A              D  2023-03-20
2599  BUCKINGHAMSHIRE                 B              D  2023-02-23
```

```
[5 rows x 34 columns]
Column names:
Index(['id', 'property_url', 'title', 'address', 'price', 'pricing_qualifier',
       'listing_time', 'property_type', 'bedrooms', 'bathrooms', 'epc_rating',
       'size', 'features', 'normalized_address', 'source', 'latitude',
       'longitude', 'Unique Transaction Identifier', 'Price',
       'Date of Transaction', 'Postal Code', 'Property Type', 'Old/New',
       'Duration', 'PAON', 'SAON', 'Street', 'Locality', 'Town/City',
       'District', 'County', 'PPD Category Type', 'Record Status', 'Date'],
      dtype='object')
```

4.2.4 4.3 Model Development

4.3.1 Content-Based Filtering Model

Introduction to Content-Based Filtering Content-based filtering is a recommendation system technique that uses the features of items to recommend other items with similar attributes. In the context of the Personalised Property Recommendation System, we will use property features such as price, geographical coordinates (latitude and longitude), and property type to recommend properties to users.

Relevant Dataset Features Our dataset includes the following relevant features for each property:

- Price
- Latitude
- Longitude
- Property Type: The type of the property (e.g., apartment, detached house).

These features are present in all rows of our preprocessed dataset and will be used to develop our recommendation model.

Building the Neural Network Model We will construct a neural network using TensorFlow to implement the content-based filtering model. The network will have an input layer corresponding to the features, hidden layers to capture complex patterns, and an output layer for the final recommendation score.

[56]:
```python
import pandas as pd

# Load the preprocessed data
file_path = './data/preprocessed-data/preprocessed.csv'
df = pd.read_csv(file_path)

# Select relevant features for the model
features = df[['Price', 'Property Type', 'latitude', 'longitude']]

# Handle missing values
features = features.dropna()
```

```python
# Convert categorical features to numeric
features = pd.get_dummies(features, columns=['Property Type'])

# Ensure the target variable matches the processed features dataframe
target = df.loc[features.index, 'Price']

# Check for invalid data
print("Checking for invalid data in features:")
print(features.isnull().sum())
print(features.describe())

print("Checking for invalid data in target:")
print(target.isnull().sum())
print(target.describe())

# Display the first few rows of the extracted features
print(features.head())
```

```
Checking for invalid data in features:
Price              0
latitude           0
longitude          0
Property Type_D    0
Property Type_F    0
Property Type_O    0
Property Type_S    0
Property Type_T    0
dtype: int64
              Price     latitude    longitude
count  2.600000e+03  2600.000000  2600.000000
mean   4.341690e+05    51.806702    -0.737249
std    2.698687e+05     0.194997     0.115481
min    1.000000e+03    51.481425    -1.115640
25%    2.900000e+05    51.628329    -0.803560
50%    4.000000e+05    51.811928    -0.741515
75%    5.250000e+05    52.004868    -0.658138
max    4.100000e+06    52.177271    -0.477487
Checking for invalid data in target:
0
count    2.600000e+03
mean     4.341690e+05
std      2.698687e+05
min      1.000000e+03
25%      2.900000e+05
50%      4.000000e+05
75%      5.250000e+05
max      4.100000e+06
Name: Price, dtype: float64
```

```
      Price   latitude  longitude  Property Type_D  Property Type_F  \
0  275000.0  51.587275  -0.683926            False             True
1  220000.0  51.803094  -0.817448            False             True
2  465000.0  52.004837  -0.802238            False            False
3  195000.0  51.527798  -0.634636            False             True
4  265000.0  51.628329  -0.742618            False             True

   Property Type_O  Property Type_S  Property Type_T
0            False            False            False
1            False            False            False
2            False             True            False
3            False            False            False
4            False            False            False
```

4.3.2 Model Training

Data Splitting and Preparation:

[57]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(features, target, test_size=0.
 ↪2, random_state=42)

# Normalize the features and target
scaler_features = StandardScaler()
X_train = scaler_features.fit_transform(X_train)
X_val = scaler_features.transform(X_val)

scaler_target = StandardScaler()
y_train = scaler_target.fit_transform(y_train.values.reshape(-1, 1)).flatten()
y_val = scaler_target.transform(y_val.values.reshape(-1, 1)).flatten()

# Display the shapes of the training and validation sets
print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
```

```
(2080, 8) (520, 8) (2080,) (520,)
```

Neural Network Model Structure:

[58]:
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input

# Display the shapes of the training and validation sets
print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)

# Define the model structure using the Input layer
```

```
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.
 ↪01)),
    Dropout(0.2),
    Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.
 ↪01)),
    Dropout(0.2),
    Dense(1, activation='linear')
])

# Compile the model with a lower learning rate
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),␣
 ↪loss='mean_squared_error', metrics=['mae'])

# Display the model summary
model.summary()
```

```
(2080, 8) (520, 8) (2080,) (520,)
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 128)	1,152
dropout_16 (Dropout)	(None, 128)	0
dense_25 (Dense)	(None, 64)	8,256
dropout_17 (Dropout)	(None, 64)	0
dense_26 (Dense)	(None, 1)	65

Total params: 9,473 (37.00 KB)

Trainable params: 9,473 (37.00 KB)

Non-trainable params: 0 (0.00 B)

Training Parameters and Process:

[59]:
```python
# Plot training history
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import EarlyStopping

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)

# Train the model
history = model.fit(X_train, y_train, epochs=200, validation_data=(X_val,␣
 ↪y_val), batch_size=32, callbacks=[early_stopping])

# Print the training history to verify the collected data
# print(history.history)

# Plot training history
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/200
65/65                 0s 2ms/step - loss:
1.8350 - mae: 0.5491 - val_loss: 1.4143 - val_mae: 0.3972
Epoch 2/200
65/65                 0s 690us/step -
loss: 1.7195 - mae: 0.4485 - val_loss: 1.2553 - val_mae: 0.3131
Epoch 3/200
65/65                 0s 625us/step -
loss: 1.2905 - mae: 0.3548 - val_loss: 1.1435 - val_mae: 0.2674
Epoch 4/200
65/65                 0s 613us/step -
loss: 1.2372 - mae: 0.3314 - val_loss: 1.0570 - val_mae: 0.2294
Epoch 5/200
65/65                 0s 605us/step -
loss: 1.1717 - mae: 0.3008 - val_loss: 0.9877 - val_mae: 0.1977
Epoch 6/200
65/65                 0s 593us/step -
loss: 1.0712 - mae: 0.2774 - val_loss: 0.9319 - val_mae: 0.1728
Epoch 7/200
65/65                 0s 562us/step -
loss: 1.0068 - mae: 0.2573 - val_loss: 0.8844 - val_mae: 0.1532
Epoch 8/200
65/65                 0s 555us/step -
loss: 0.9622 - mae: 0.2519 - val_loss: 0.8416 - val_mae: 0.1329
Epoch 9/200
```

```
65/65            0s 562us/step -
loss: 0.9333 - mae: 0.2312 - val_loss: 0.8045 - val_mae: 0.1147
Epoch 10/200
65/65            0s 554us/step -
loss: 0.8678 - mae: 0.2084 - val_loss: 0.7755 - val_mae: 0.1060
Epoch 11/200
65/65            0s 592us/step -
loss: 0.8211 - mae: 0.1959 - val_loss: 0.7477 - val_mae: 0.0955
Epoch 12/200
65/65            0s 589us/step -
loss: 0.7985 - mae: 0.1943 - val_loss: 0.7220 - val_mae: 0.0893
Epoch 13/200
65/65            0s 593us/step -
loss: 0.7681 - mae: 0.1852 - val_loss: 0.6981 - val_mae: 0.0835
Epoch 14/200
65/65            0s 631us/step -
loss: 0.7621 - mae: 0.1963 - val_loss: 0.6766 - val_mae: 0.0852
Epoch 15/200
65/65            0s 579us/step -
loss: 0.7310 - mae: 0.1834 - val_loss: 0.6539 - val_mae: 0.0794
Epoch 16/200
65/65            0s 609us/step -
loss: 0.7233 - mae: 0.1810 - val_loss: 0.6325 - val_mae: 0.0756
Epoch 17/200
65/65            0s 583us/step -
loss: 0.6933 - mae: 0.1816 - val_loss: 0.6119 - val_mae: 0.0690
Epoch 18/200
65/65            0s 577us/step -
loss: 0.6652 - mae: 0.1745 - val_loss: 0.5924 - val_mae: 0.0689
Epoch 19/200
65/65            0s 543us/step -
loss: 0.6400 - mae: 0.1689 - val_loss: 0.5740 - val_mae: 0.0710
Epoch 20/200
65/65            0s 582us/step -
loss: 0.6115 - mae: 0.1578 - val_loss: 0.5554 - val_mae: 0.0668
Epoch 21/200
65/65            0s 571us/step -
loss: 0.5916 - mae: 0.1578 - val_loss: 0.5376 - val_mae: 0.0643
Epoch 22/200
65/65            0s 573us/step -
loss: 0.5910 - mae: 0.1619 - val_loss: 0.5225 - val_mae: 0.0722
Epoch 23/200
65/65            0s 560us/step -
loss: 0.5791 - mae: 0.1626 - val_loss: 0.5044 - val_mae: 0.0613
Epoch 24/200
65/65            0s 554us/step -
loss: 0.5491 - mae: 0.1540 - val_loss: 0.4890 - val_mae: 0.0629
Epoch 25/200
```

```
65/65            0s 584us/step -
loss: 0.5330 - mae: 0.1478 - val_loss: 0.4732 - val_mae: 0.0585
Epoch 26/200
65/65            0s 554us/step -
loss: 0.5210 - mae: 0.1507 - val_loss: 0.4595 - val_mae: 0.0631
Epoch 27/200
65/65            0s 559us/step -
loss: 0.4942 - mae: 0.1404 - val_loss: 0.4445 - val_mae: 0.0570
Epoch 28/200
65/65            0s 570us/step -
loss: 0.4738 - mae: 0.1356 - val_loss: 0.4308 - val_mae: 0.0562
Epoch 29/200
65/65            0s 583us/step -
loss: 0.4546 - mae: 0.1302 - val_loss: 0.4172 - val_mae: 0.0521
Epoch 30/200
65/65            0s 554us/step -
loss: 0.4509 - mae: 0.1322 - val_loss: 0.4043 - val_mae: 0.0502
Epoch 31/200
65/65            0s 583us/step -
loss: 0.4302 - mae: 0.1296 - val_loss: 0.3925 - val_mae: 0.0532
Epoch 32/200
65/65            0s 574us/step -
loss: 0.4223 - mae: 0.1266 - val_loss: 0.3805 - val_mae: 0.0519
Epoch 33/200
65/65            0s 558us/step -
loss: 0.4088 - mae: 0.1315 - val_loss: 0.3689 - val_mae: 0.0501
Epoch 34/200
65/65            0s 571us/step -
loss: 0.4306 - mae: 0.1330 - val_loss: 0.3582 - val_mae: 0.0515
Epoch 35/200
65/65            0s 590us/step -
loss: 0.3993 - mae: 0.1283 - val_loss: 0.3469 - val_mae: 0.0482
Epoch 36/200
65/65            0s 593us/step -
loss: 0.3864 - mae: 0.1219 - val_loss: 0.3364 - val_mae: 0.0462
Epoch 37/200
65/65            0s 599us/step -
loss: 0.3781 - mae: 0.1245 - val_loss: 0.3264 - val_mae: 0.0467
Epoch 38/200
65/65            0s 550us/step -
loss: 0.3615 - mae: 0.1220 - val_loss: 0.3165 - val_mae: 0.0455
Epoch 39/200
65/65            0s 568us/step -
loss: 0.3354 - mae: 0.1126 - val_loss: 0.3071 - val_mae: 0.0441
Epoch 40/200
65/65            0s 580us/step -
loss: 0.3342 - mae: 0.1145 - val_loss: 0.2986 - val_mae: 0.0481
Epoch 41/200
```

```
65/65              0s 562us/step -
loss: 0.3290 - mae: 0.1130 - val_loss: 0.2891 - val_mae: 0.0396
Epoch 42/200
65/65              0s 572us/step -
loss: 0.3110 - mae: 0.1069 - val_loss: 0.2807 - val_mae: 0.0409
Epoch 43/200
65/65              0s 560us/step -
loss: 0.3042 - mae: 0.1073 - val_loss: 0.2733 - val_mae: 0.0469
Epoch 44/200
65/65              0s 674us/step -
loss: 0.2916 - mae: 0.1047 - val_loss: 0.2670 - val_mae: 0.0535
Epoch 45/200
65/65              0s 573us/step -
loss: 0.2994 - mae: 0.1149 - val_loss: 0.2573 - val_mae: 0.0421
Epoch 46/200
65/65              0s 567us/step -
loss: 0.2834 - mae: 0.1087 - val_loss: 0.2495 - val_mae: 0.0393
Epoch 47/200
65/65              0s 555us/step -
loss: 0.2765 - mae: 0.1043 - val_loss: 0.2426 - val_mae: 0.0390
Epoch 48/200
65/65              0s 590us/step -
loss: 0.2685 - mae: 0.1054 - val_loss: 0.2360 - val_mae: 0.0412
Epoch 49/200
65/65              0s 559us/step -
loss: 0.2532 - mae: 0.1026 - val_loss: 0.2290 - val_mae: 0.0390
Epoch 50/200
65/65              0s 604us/step -
loss: 0.2547 - mae: 0.1031 - val_loss: 0.2230 - val_mae: 0.0438
Epoch 51/200
65/65              0s 575us/step -
loss: 0.2572 - mae: 0.1064 - val_loss: 0.2165 - val_mae: 0.0408
Epoch 52/200
65/65              0s 594us/step -
loss: 0.2400 - mae: 0.0990 - val_loss: 0.2101 - val_mae: 0.0365
Epoch 53/200
65/65              0s 547us/step -
loss: 0.2384 - mae: 0.1066 - val_loss: 0.2047 - val_mae: 0.0378
Epoch 54/200
65/65              0s 563us/step -
loss: 0.2327 - mae: 0.0989 - val_loss: 0.1993 - val_mae: 0.0394
Epoch 55/200
65/65              0s 545us/step -
loss: 0.2209 - mae: 0.0977 - val_loss: 0.1946 - val_mae: 0.0425
Epoch 56/200
65/65              0s 580us/step -
loss: 0.2103 - mae: 0.0936 - val_loss: 0.1895 - val_mae: 0.0401
Epoch 57/200
```

```
65/65            0s 583us/step -
loss: 0.2028 - mae: 0.0950 - val_loss: 0.1833 - val_mae: 0.0362
Epoch 58/200
65/65            0s 576us/step -
loss: 0.2000 - mae: 0.0921 - val_loss: 0.1789 - val_mae: 0.0380
Epoch 59/200
65/65            0s 581us/step -
loss: 0.2189 - mae: 0.1024 - val_loss: 0.1737 - val_mae: 0.0364
Epoch 60/200
65/65            0s 599us/step -
loss: 0.1910 - mae: 0.0877 - val_loss: 0.1697 - val_mae: 0.0403
Epoch 61/200
65/65            0s 616us/step -
loss: 0.1883 - mae: 0.0917 - val_loss: 0.1652 - val_mae: 0.0382
Epoch 62/200
65/65            0s 589us/step -
loss: 0.1799 - mae: 0.0887 - val_loss: 0.1605 - val_mae: 0.0361
Epoch 63/200
65/65            0s 549us/step -
loss: 0.1917 - mae: 0.0950 - val_loss: 0.1567 - val_mae: 0.0370
Epoch 64/200
65/65            0s 577us/step -
loss: 0.1874 - mae: 0.0979 - val_loss: 0.1527 - val_mae: 0.0350
Epoch 65/200
65/65            0s 574us/step -
loss: 0.1744 - mae: 0.0926 - val_loss: 0.1491 - val_mae: 0.0340
Epoch 66/200
65/65            0s 572us/step -
loss: 0.1825 - mae: 0.1006 - val_loss: 0.1456 - val_mae: 0.0368
Epoch 67/200
65/65            0s 588us/step -
loss: 0.1737 - mae: 0.0925 - val_loss: 0.1419 - val_mae: 0.0396
Epoch 68/200
65/65            0s 567us/step -
loss: 0.1562 - mae: 0.0866 - val_loss: 0.1385 - val_mae: 0.0376
Epoch 69/200
65/65            0s 560us/step -
loss: 0.1560 - mae: 0.0870 - val_loss: 0.1348 - val_mae: 0.0350
Epoch 70/200
65/65            0s 567us/step -
loss: 0.1547 - mae: 0.0929 - val_loss: 0.1318 - val_mae: 0.0342
Epoch 71/200
65/65            0s 590us/step -
loss: 0.1477 - mae: 0.0836 - val_loss: 0.1287 - val_mae: 0.0358
Epoch 72/200
65/65            0s 576us/step -
loss: 0.1550 - mae: 0.0917 - val_loss: 0.1256 - val_mae: 0.0356
Epoch 73/200
```

```
65/65             0s 568us/step -
loss: 0.1445 - mae: 0.0850 - val_loss: 0.1224 - val_mae: 0.0338
Epoch 74/200
65/65             0s 535us/step -
loss: 0.1429 - mae: 0.0915 - val_loss: 0.1200 - val_mae: 0.0369
Epoch 75/200
65/65             0s 560us/step -
loss: 0.1462 - mae: 0.0906 - val_loss: 0.1175 - val_mae: 0.0407
Epoch 76/200
65/65             0s 538us/step -
loss: 0.1371 - mae: 0.0898 - val_loss: 0.1142 - val_mae: 0.0335
Epoch 77/200
65/65             0s 583us/step -
loss: 0.1284 - mae: 0.0826 - val_loss: 0.1115 - val_mae: 0.0337
Epoch 78/200
65/65             0s 562us/step -
loss: 0.1284 - mae: 0.0832 - val_loss: 0.1093 - val_mae: 0.0358
Epoch 79/200
65/65             0s 561us/step -
loss: 0.1282 - mae: 0.0829 - val_loss: 0.1068 - val_mae: 0.0348
Epoch 80/200
65/65             0s 574us/step -
loss: 0.1413 - mae: 0.0926 - val_loss: 0.1043 - val_mae: 0.0332
Epoch 81/200
65/65             0s 572us/step -
loss: 0.1171 - mae: 0.0795 - val_loss: 0.1020 - val_mae: 0.0320
Epoch 82/200
65/65             0s 600us/step -
loss: 0.1186 - mae: 0.0822 - val_loss: 0.1007 - val_mae: 0.0351
Epoch 83/200
65/65             0s 591us/step -
loss: 0.1114 - mae: 0.0793 - val_loss: 0.0980 - val_mae: 0.0332
Epoch 84/200
65/65             0s 603us/step -
loss: 0.1125 - mae: 0.0783 - val_loss: 0.0959 - val_mae: 0.0339
Epoch 85/200
65/65             0s 586us/step -
loss: 0.1132 - mae: 0.0820 - val_loss: 0.0937 - val_mae: 0.0369
Epoch 86/200
65/65             0s 603us/step -
loss: 0.1131 - mae: 0.0839 - val_loss: 0.0922 - val_mae: 0.0327
Epoch 87/200
65/65             0s 590us/step -
loss: 0.1055 - mae: 0.0780 - val_loss: 0.0901 - val_mae: 0.0342
Epoch 88/200
65/65             0s 593us/step -
loss: 0.1094 - mae: 0.0824 - val_loss: 0.0880 - val_mae: 0.0315
Epoch 89/200
```

```
65/65             0s 594us/step -
loss: 0.1064 - mae: 0.0841 - val_loss: 0.0864 - val_mae: 0.0340
Epoch 90/200
65/65             0s 683us/step -
loss: 0.1298 - mae: 0.0852 - val_loss: 0.0846 - val_mae: 0.0284
Epoch 91/200
65/65             0s 589us/step -
loss: 0.1009 - mae: 0.0811 - val_loss: 0.0830 - val_mae: 0.0308
Epoch 92/200
65/65             0s 596us/step -
loss: 0.1040 - mae: 0.0816 - val_loss: 0.0821 - val_mae: 0.0321
Epoch 93/200
65/65             0s 574us/step -
loss: 0.0963 - mae: 0.0752 - val_loss: 0.0799 - val_mae: 0.0317
Epoch 94/200
65/65             0s 599us/step -
loss: 0.0925 - mae: 0.0774 - val_loss: 0.0787 - val_mae: 0.0318
Epoch 95/200
65/65             0s 567us/step -
loss: 0.0930 - mae: 0.0781 - val_loss: 0.0769 - val_mae: 0.0303
Epoch 96/200
65/65             0s 569us/step -
loss: 0.0908 - mae: 0.0790 - val_loss: 0.0760 - val_mae: 0.0306
Epoch 97/200
65/65             0s 560us/step -
loss: 0.1079 - mae: 0.0853 - val_loss: 0.0762 - val_mae: 0.0401
Epoch 98/200
65/65             0s 552us/step -
loss: 0.0940 - mae: 0.0814 - val_loss: 0.0735 - val_mae: 0.0328
Epoch 99/200
65/65             0s 576us/step -
loss: 0.0923 - mae: 0.0793 - val_loss: 0.0717 - val_mae: 0.0302
Epoch 100/200
65/65             0s 593us/step -
loss: 0.0864 - mae: 0.0764 - val_loss: 0.0708 - val_mae: 0.0313
Epoch 101/200
65/65             0s 610us/step -
loss: 0.0833 - mae: 0.0774 - val_loss: 0.0694 - val_mae: 0.0350
Epoch 102/200
65/65             0s 573us/step -
loss: 0.0827 - mae: 0.0770 - val_loss: 0.0680 - val_mae: 0.0280
Epoch 103/200
65/65             0s 613us/step -
loss: 0.0873 - mae: 0.0769 - val_loss: 0.0673 - val_mae: 0.0298
Epoch 104/200
65/65             0s 574us/step -
loss: 0.0825 - mae: 0.0807 - val_loss: 0.0663 - val_mae: 0.0303
Epoch 105/200
```

```
65/65             0s 598us/step -
loss: 0.0816 - mae: 0.0805 - val_loss: 0.0665 - val_mae: 0.0386
Epoch 106/200
65/65             0s 572us/step -
loss: 0.0783 - mae: 0.0773 - val_loss: 0.0641 - val_mae: 0.0313
Epoch 107/200
65/65             0s 572us/step -
loss: 0.0794 - mae: 0.0773 - val_loss: 0.0629 - val_mae: 0.0267
Epoch 108/200
65/65             0s 609us/step -
loss: 0.0763 - mae: 0.0765 - val_loss: 0.0621 - val_mae: 0.0279
Epoch 109/200
65/65             0s 587us/step -
loss: 0.0763 - mae: 0.0744 - val_loss: 0.0617 - val_mae: 0.0303
Epoch 110/200
65/65             0s 571us/step -
loss: 0.0782 - mae: 0.0781 - val_loss: 0.0602 - val_mae: 0.0304
Epoch 111/200
65/65             0s 589us/step -
loss: 0.0730 - mae: 0.0754 - val_loss: 0.0599 - val_mae: 0.0318
Epoch 112/200
65/65             0s 562us/step -
loss: 0.0760 - mae: 0.0796 - val_loss: 0.0606 - val_mae: 0.0310
Epoch 113/200
65/65             0s 563us/step -
loss: 0.0730 - mae: 0.0773 - val_loss: 0.0577 - val_mae: 0.0286
Epoch 114/200
65/65             0s 581us/step -
loss: 0.0711 - mae: 0.0782 - val_loss: 0.0571 - val_mae: 0.0312
Epoch 115/200
65/65             0s 593us/step -
loss: 0.0693 - mae: 0.0766 - val_loss: 0.0571 - val_mae: 0.0334
Epoch 116/200
65/65             0s 598us/step -
loss: 0.0665 - mae: 0.0721 - val_loss: 0.0566 - val_mae: 0.0305
Epoch 117/200
65/65             0s 588us/step -
loss: 0.0714 - mae: 0.0722 - val_loss: 0.0556 - val_mae: 0.0306
Epoch 118/200
65/65             0s 600us/step -
loss: 0.0725 - mae: 0.0807 - val_loss: 0.0538 - val_mae: 0.0300
Epoch 119/200
65/65             0s 595us/step -
loss: 0.0742 - mae: 0.0796 - val_loss: 0.0527 - val_mae: 0.0303
Epoch 120/200
65/65             0s 577us/step -
loss: 0.0689 - mae: 0.0788 - val_loss: 0.0529 - val_mae: 0.0350
Epoch 121/200
```

```
65/65            0s 596us/step -
loss: 0.0721 - mae: 0.0771 - val_loss: 0.0514 - val_mae: 0.0258
Epoch 122/200
65/65            0s 594us/step -
loss: 0.0647 - mae: 0.0749 - val_loss: 0.0509 - val_mae: 0.0278
Epoch 123/200
65/65            0s 570us/step -
loss: 0.0807 - mae: 0.0794 - val_loss: 0.0502 - val_mae: 0.0272
Epoch 124/200
65/65            0s 570us/step -
loss: 0.0658 - mae: 0.0767 - val_loss: 0.0502 - val_mae: 0.0302
Epoch 125/200
65/65            0s 560us/step -
loss: 0.0614 - mae: 0.0754 - val_loss: 0.0492 - val_mae: 0.0270
Epoch 126/200
65/65            0s 583us/step -
loss: 0.0699 - mae: 0.0788 - val_loss: 0.0485 - val_mae: 0.0280
Epoch 127/200
65/65            0s 576us/step -
loss: 0.0617 - mae: 0.0747 - val_loss: 0.0486 - val_mae: 0.0273
Epoch 128/200
65/65            0s 612us/step -
loss: 0.0660 - mae: 0.0782 - val_loss: 0.0482 - val_mae: 0.0293
Epoch 129/200
65/65            0s 573us/step -
loss: 0.0686 - mae: 0.0767 - val_loss: 0.0469 - val_mae: 0.0273
Epoch 130/200
65/65            0s 578us/step -
loss: 0.0615 - mae: 0.0739 - val_loss: 0.0465 - val_mae: 0.0268
Epoch 131/200
65/65            0s 590us/step -
loss: 0.0586 - mae: 0.0745 - val_loss: 0.0458 - val_mae: 0.0278
Epoch 132/200
65/65            0s 574us/step -
loss: 0.0592 - mae: 0.0745 - val_loss: 0.0449 - val_mae: 0.0268
Epoch 133/200
65/65            0s 592us/step -
loss: 0.0647 - mae: 0.0817 - val_loss: 0.0446 - val_mae: 0.0279
Epoch 134/200
65/65            0s 591us/step -
loss: 0.0566 - mae: 0.0737 - val_loss: 0.0446 - val_mae: 0.0301
Epoch 135/200
65/65            0s 693us/step -
loss: 0.0652 - mae: 0.0782 - val_loss: 0.0442 - val_mae: 0.0278
Epoch 136/200
65/65            0s 587us/step -
loss: 0.0614 - mae: 0.0777 - val_loss: 0.0431 - val_mae: 0.0280
Epoch 137/200
```

```
65/65            0s 557us/step -
loss: 0.0587 - mae: 0.0788 - val_loss: 0.0439 - val_mae: 0.0280
Epoch 138/200
65/65            0s 589us/step -
loss: 0.0554 - mae: 0.0732 - val_loss: 0.0427 - val_mae: 0.0261
Epoch 139/200
65/65            0s 584us/step -
loss: 0.0531 - mae: 0.0709 - val_loss: 0.0421 - val_mae: 0.0297
Epoch 140/200
65/65            0s 597us/step -
loss: 0.0616 - mae: 0.0801 - val_loss: 0.0418 - val_mae: 0.0261
Epoch 141/200
65/65            0s 566us/step -
loss: 0.0775 - mae: 0.0853 - val_loss: 0.0426 - val_mae: 0.0311
Epoch 142/200
65/65            0s 596us/step -
loss: 0.0514 - mae: 0.0724 - val_loss: 0.0426 - val_mae: 0.0327
Epoch 143/200
65/65            0s 566us/step -
loss: 0.0548 - mae: 0.0753 - val_loss: 0.0414 - val_mae: 0.0263
Epoch 144/200
65/65            0s 600us/step -
loss: 0.0528 - mae: 0.0753 - val_loss: 0.0411 - val_mae: 0.0295
Epoch 145/200
65/65            0s 547us/step -
loss: 0.0585 - mae: 0.0781 - val_loss: 0.0398 - val_mae: 0.0301
Epoch 146/200
65/65            0s 582us/step -
loss: 0.0554 - mae: 0.0786 - val_loss: 0.0390 - val_mae: 0.0246
Epoch 147/200
65/65            0s 597us/step -
loss: 0.0642 - mae: 0.0735 - val_loss: 0.0395 - val_mae: 0.0253
Epoch 148/200
65/65            0s 561us/step -
loss: 0.0543 - mae: 0.0758 - val_loss: 0.0387 - val_mae: 0.0246
Epoch 149/200
65/65            0s 567us/step -
loss: 0.0609 - mae: 0.0804 - val_loss: 0.0382 - val_mae: 0.0256
Epoch 150/200
65/65            0s 596us/step -
loss: 0.0494 - mae: 0.0706 - val_loss: 0.0378 - val_mae: 0.0228
Epoch 151/200
65/65            0s 573us/step -
loss: 0.0530 - mae: 0.0757 - val_loss: 0.0380 - val_mae: 0.0240
Epoch 152/200
65/65            0s 582us/step -
loss: 0.0497 - mae: 0.0729 - val_loss: 0.0372 - val_mae: 0.0271
Epoch 153/200
```

```
65/65            0s 571us/step -
loss: 0.0490 - mae: 0.0719 - val_loss: 0.0371 - val_mae: 0.0254
Epoch 154/200
65/65            0s 581us/step -
loss: 0.0647 - mae: 0.0809 - val_loss: 0.0367 - val_mae: 0.0261
Epoch 155/200
65/65            0s 584us/step -
loss: 0.0468 - mae: 0.0714 - val_loss: 0.0367 - val_mae: 0.0322
Epoch 156/200
65/65            0s 589us/step -
loss: 0.0556 - mae: 0.0777 - val_loss: 0.0361 - val_mae: 0.0233
Epoch 157/200
65/65            0s 549us/step -
loss: 0.0505 - mae: 0.0726 - val_loss: 0.0359 - val_mae: 0.0217
Epoch 158/200
65/65            0s 545us/step -
loss: 0.0505 - mae: 0.0725 - val_loss: 0.0362 - val_mae: 0.0264
Epoch 159/200
65/65            0s 570us/step -
loss: 0.0653 - mae: 0.0791 - val_loss: 0.0354 - val_mae: 0.0214
Epoch 160/200
65/65            0s 547us/step -
loss: 0.0471 - mae: 0.0726 - val_loss: 0.0355 - val_mae: 0.0235
Epoch 161/200
65/65            0s 596us/step -
loss: 0.0556 - mae: 0.0802 - val_loss: 0.0348 - val_mae: 0.0246
Epoch 162/200
65/65            0s 559us/step -
loss: 0.0443 - mae: 0.0704 - val_loss: 0.0349 - val_mae: 0.0256
Epoch 163/200
65/65            0s 546us/step -
loss: 0.0564 - mae: 0.0775 - val_loss: 0.0349 - val_mae: 0.0252
Epoch 164/200
65/65            0s 532us/step -
loss: 0.0433 - mae: 0.0715 - val_loss: 0.0350 - val_mae: 0.0241
Epoch 165/200
65/65            0s 544us/step -
loss: 0.0469 - mae: 0.0714 - val_loss: 0.0346 - val_mae: 0.0286
Epoch 166/200
65/65            0s 570us/step -
loss: 0.0491 - mae: 0.0733 - val_loss: 0.0355 - val_mae: 0.0257
Epoch 167/200
65/65            0s 549us/step -
loss: 0.0527 - mae: 0.0728 - val_loss: 0.0337 - val_mae: 0.0300
Epoch 168/200
65/65            0s 557us/step -
loss: 0.0513 - mae: 0.0756 - val_loss: 0.0338 - val_mae: 0.0262
Epoch 169/200
```

```
65/65            0s 593us/step -
loss: 0.0526 - mae: 0.0755 - val_loss: 0.0334 - val_mae: 0.0236
Epoch 170/200
65/65            0s 586us/step -
loss: 0.0449 - mae: 0.0726 - val_loss: 0.0347 - val_mae: 0.0280
Epoch 171/200
65/65            0s 583us/step -
loss: 0.0503 - mae: 0.0724 - val_loss: 0.0328 - val_mae: 0.0223
Epoch 172/200
65/65            0s 554us/step -
loss: 0.0487 - mae: 0.0715 - val_loss: 0.0321 - val_mae: 0.0212
Epoch 173/200
65/65            0s 580us/step -
loss: 0.0498 - mae: 0.0728 - val_loss: 0.0323 - val_mae: 0.0225
Epoch 174/200
65/65            0s 560us/step -
loss: 0.0447 - mae: 0.0765 - val_loss: 0.0321 - val_mae: 0.0231
Epoch 175/200
65/65            0s 553us/step -
loss: 0.0459 - mae: 0.0738 - val_loss: 0.0320 - val_mae: 0.0260
Epoch 176/200
65/65            0s 557us/step -
loss: 0.0524 - mae: 0.0752 - val_loss: 0.0314 - val_mae: 0.0231
Epoch 177/200
65/65            0s 549us/step -
loss: 0.0438 - mae: 0.0712 - val_loss: 0.0349 - val_mae: 0.0399
Epoch 178/200
65/65            0s 531us/step -
loss: 0.0518 - mae: 0.0759 - val_loss: 0.0311 - val_mae: 0.0221
Epoch 179/200
65/65            0s 550us/step -
loss: 0.0429 - mae: 0.0707 - val_loss: 0.0309 - val_mae: 0.0223
Epoch 180/200
65/65            0s 571us/step -
loss: 0.0478 - mae: 0.0783 - val_loss: 0.0308 - val_mae: 0.0242
Epoch 181/200
65/65            0s 663us/step -
loss: 0.0435 - mae: 0.0717 - val_loss: 0.0315 - val_mae: 0.0234
Epoch 182/200
65/65            0s 539us/step -
loss: 0.0418 - mae: 0.0715 - val_loss: 0.0304 - val_mae: 0.0231
Epoch 183/200
65/65            0s 556us/step -
loss: 0.0436 - mae: 0.0726 - val_loss: 0.0306 - val_mae: 0.0257
Epoch 184/200
65/65            0s 576us/step -
loss: 0.0436 - mae: 0.0684 - val_loss: 0.0297 - val_mae: 0.0236
Epoch 185/200
```

```
65/65              0s 557us/step -
loss: 0.0517 - mae: 0.0713 - val_loss: 0.0299 - val_mae: 0.0221
Epoch 186/200
65/65              0s 563us/step -
loss: 0.0415 - mae: 0.0722 - val_loss: 0.0293 - val_mae: 0.0198
Epoch 187/200
65/65              0s 545us/step -
loss: 0.0491 - mae: 0.0733 - val_loss: 0.0293 - val_mae: 0.0221
Epoch 188/200
65/65              0s 566us/step -
loss: 0.0457 - mae: 0.0695 - val_loss: 0.0298 - val_mae: 0.0265
Epoch 189/200
65/65              0s 571us/step -
loss: 0.0478 - mae: 0.0747 - val_loss: 0.0289 - val_mae: 0.0241
Epoch 190/200
65/65              0s 560us/step -
loss: 0.0528 - mae: 0.0776 - val_loss: 0.0298 - val_mae: 0.0254
Epoch 191/200
65/65              0s 558us/step -
loss: 0.0390 - mae: 0.0694 - val_loss: 0.0291 - val_mae: 0.0264
Epoch 192/200
65/65              0s 572us/step -
loss: 0.0435 - mae: 0.0735 - val_loss: 0.0284 - val_mae: 0.0180
Epoch 193/200
65/65              0s 541us/step -
loss: 0.0408 - mae: 0.0722 - val_loss: 0.0283 - val_mae: 0.0217
Epoch 194/200
65/65              0s 565us/step -
loss: 0.0463 - mae: 0.0747 - val_loss: 0.0280 - val_mae: 0.0210
Epoch 195/200
65/65              0s 544us/step -
loss: 0.0424 - mae: 0.0701 - val_loss: 0.0295 - val_mae: 0.0314
Epoch 196/200
65/65              0s 582us/step -
loss: 0.0382 - mae: 0.0690 - val_loss: 0.0285 - val_mae: 0.0215
Epoch 197/200
65/65              0s 569us/step -
loss: 0.0427 - mae: 0.0715 - val_loss: 0.0277 - val_mae: 0.0232
Epoch 198/200
65/65              0s 542us/step -
loss: 0.0385 - mae: 0.0727 - val_loss: 0.0276 - val_mae: 0.0223
Epoch 199/200
65/65              0s 554us/step -
loss: 0.0378 - mae: 0.0682 - val_loss: 0.0276 - val_mae: 0.0189
Epoch 200/200
65/65              0s 550us/step -
loss: 0.0393 - mae: 0.0700 - val_loss: 0.0273 - val_mae: 0.0208
{'loss': [1.825515866279602, 1.5521342754364014, 1.3274273872375488,
```

1.208950161933899, 1.1324560642242432, 1.0724793672561646, 0.9943206906318665,
0.9631871581077576, 0.9310721755027771, 0.8577185869216919, 0.8282926678657532,
0.7999303936958313, 0.7650788426399231, 0.7569543719291687, 0.7189301252365112,
0.7069898843765259, 0.682901918888092, 0.6547801494598389, 0.6432111263275146,
0.6072601675987244, 0.587548553943634, 0.5841178297996521, 0.5702789425849915,
0.5418058633804321, 0.5392463207244873, 0.5155147910118103, 0.4955413341522217,
0.4710766077041626, 0.4528196156024933, 0.45021823048591614,
0.43038949370384216, 0.42456871271133423, 0.4077240824699402,
0.4051710069179535, 0.39168429374694824, 0.3779614269733429,
0.36216941475868225, 0.3598874807357788, 0.33639076352119446,
0.3327787220478058, 0.33949339389801025, 0.3111379146575928,
0.30564814805984497, 0.29194021224975586, 0.2913252115249634,
0.2804591655731201, 0.28024765849113464, 0.2719748318195343, 0.2556295394897461,
0.25263261795043945, 0.24828775227069855, 0.23908698558807373,
0.2382689118385315, 0.23540353775024414, 0.2225879728794098, 0.217087984085083,
0.20271922647953033, 0.20273157954216003, 0.20399843156337738,
0.19343508780002594, 0.19033478200435638, 0.1894349604845047,
0.1862175613641739, 0.18061590194702148, 0.17273230850696564,
0.18215133249759674, 0.1684826910495758, 0.1586548388004303,
0.15437079966068268, 0.15407276153564453, 0.15237650275230408,
0.14584697782993317, 0.14613120257854462, 0.14005301892757416,
0.1438787579536438, 0.13966646790504456, 0.1286073923110962,
0.12710456550121307, 0.13007110357284546, 0.12897463142871857,
0.12439420819282532, 0.12024719268083572, 0.11278538405895233,
0.11449646949768066, 0.1146785244345665, 0.11156732589006424,
0.10625024139881134, 0.10914794355630875, 0.10173255950212479,
0.11619604378938675, 0.09926439076662064, 0.10072379559278488,
0.10360387712717056, 0.09696944057941437, 0.09401272237300873,
0.0912514328956604, 0.10122072696685791, 0.09343263506889343,
0.09576845914125443, 0.0923842117190361, 0.08392839133739471,
0.08577796816825867, 0.08849748224020004, 0.08335911482572556,
0.0827121362090108, 0.07960288971662521, 0.08330889791250229,
0.07506965845823288, 0.07697037607431412, 0.07892586290836334,
0.07612844556570053, 0.07538973540067673, 0.07451928406953812,
0.07255961000919342, 0.07051248848438263, 0.06681882590055466,
0.0745912566781044, 0.06811970472335815, 0.07079141587018967,
0.0667465552687645, 0.07015632092952728, 0.07231957465410233,
0.07636374235153198, 0.06591146439313889, 0.06327321380376816,
0.0677023524045944, 0.06178074702620506, 0.06423106789588928,
0.06488629430532455, 0.0632317066192627, 0.059481676667928696,
0.06070757284760475, 0.061631642282009125, 0.060413211584091187,
0.0631561912298203, 0.06081046536564827, 0.059178225696086884,
0.06171834096312523, 0.057219624519348145, 0.05646739527583122,
0.06270045787096024, 0.0518820919096469, 0.05546141788363457,
0.05598027631640434, 0.06144438683986664, 0.06179182976484299,
0.06768579035997391, 0.06072428822517395, 0.052293285727500916,
0.052717674523591995, 0.05397320166230202, 0.05087874084711075,
0.0556741692125797 3, 0.05709041282534599, 0.049688201397657394,

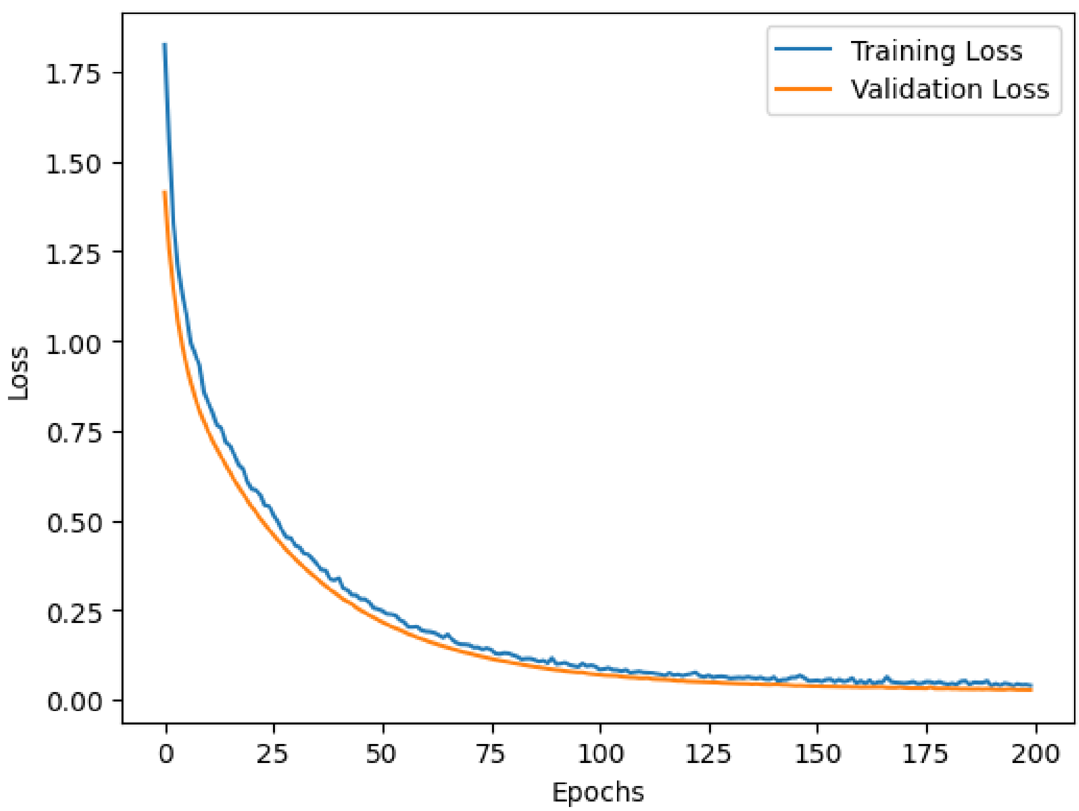
0.056136175990104675, 0.05065878480672836, 0.052990205585956573,
0.05987878888845444, 0.045899391174316406, 0.05256681889295578,
0.045291196554899216, 0.054158009588718414, 0.044111527502536774,
0.04978049546480179, 0.04922322928905487, 0.06323596090078354,
0.05001801624894142, 0.04817194491624832, 0.04693671688437462,
0.045858096331357956, 0.047239694744348526, 0.05033918097615242,
0.04472764953970909, 0.04560314118862152, 0.050148773938417435,
0.049684103578329086, 0.046002406626939774, 0.050027549266815186,
0.045720845460891724, 0.04288311302661896, 0.0452786423265934,
0.042446013540029526, 0.05110311880707741, 0.052742935717105865,
0.042610201984643936, 0.048589516431093216, 0.047721587121486664,
0.04759136214852333, 0.05243462324142456, 0.038798220455646515,
0.044737398624420166, 0.04002050310373306, 0.04590121656656265,
0.04376724734902382, 0.03885500133037567, 0.044114917516708374,
0.041392870247364044, 0.04221904277801514, 0.039890095591545105], 'mae':
[0.5236052870750427, 0.41548171639442444, 0.3504365384578705,
0.3169167935848236, 0.29192161560058594, 0.27434319257736206,
0.25598573684692383, 0.24483856558799744, 0.2314072698354721,
0.21095991134643555, 0.20234595239162445, 0.19957230985164642,
0.19059805572032928, 0.19300055503845215, 0.18113628029823303,
0.1770687699317932, 0.17182393372058868, 0.17008204758167267,
0.17036239802837372, 0.15879902243614197, 0.15699242055416107,
0.16024647653102875, 0.1555853635072708, 0.15008427202701569,
0.1499571055173874, 0.1460249423980713, 0.14211545884609222, 0.135373055934906,
0.13120244443416595, 0.1351381540298462, 0.13011862337589264,
0.12716475129127502, 0.13048559427261353, 0.1280742883682251,
0.12619049847126007, 0.12095242738723755, 0.11880916357040405,
0.12100853770971298, 0.11333852261304855, 0.11543251574039459,
0.11388099938631058, 0.1092705950140953, 0.11207222193479538,
0.1060250774025917, 0.10983821004629135, 0.10846029222011566,
0.1073911190032959, 0.10580842196941376, 0.10234547406435013,
0.1025761291384697, 0.10321526229381561, 0.09978364408016205,
0.10429241508245468, 0.10069988667964935, 0.09793692827224731,
0.09674803167581558, 0.09410776942968369, 0.09493856132030487,
0.09685910493135452, 0.09175620973110199, 0.09503670781850815,
0.09300613403320312, 0.09584532678127289, 0.09485333412885666,
0.09226317703723907, 0.09894514083862305, 0.09051281213760376,
0.08888090401887894, 0.08648768067359924, 0.0899091362953186,
0.0893077701330185, 0.08690045773983002, 0.08643922954797745,
0.08710763603448868, 0.08916634321212769, 0.0892007052898407,
0.08476044237613678, 0.08470732718706131, 0.08558832854032516,
0.08680426329374313, 0.08265159279108047, 0.08555617928504944,
0.08156260848045349, 0.08187456429004669, 0.0847250446677208,
0.08332858234643936, 0.0784435123205185, 0.08360850811004639,
0.07980725914239883, 0.0817413181066513, 0.07991333305835724,
0.08111970126628876, 0.07985377311706543, 0.08038493990898132,
0.07948102056980133, 0.07885891199111938, 0.08143619447946548,
0.08065131306648254, 0.08177465945482254, 0.08111214637756348,

0.07915503531694412, 0.07899682223796844, 0.07928748428821564,
0.08107022196054459, 0.08073510974645615, 0.07761102169752121,
0.08027292042970657, 0.07576552033424377, 0.07587068527936935,
0.0793316662311554, 0.07632984220981598, 0.0793338194489479,
0.07783262431621552, 0.07702115178108215, 0.07635830342769623,
0.0743454024195671, 0.0769047886133194, 0.07681168615818024,
0.07754102349281311, 0.07644131779670715, 0.07444462180137634,
0.07892237603664398, 0.07949631661176682, 0.07599670439958572,
0.07518921792507172, 0.0784917101264, 0.0732257142663002, 0.07829703390598297,
0.07597038149833679, 0.0741250067949295, 0.07499866187572479,
0.07630563527345657, 0.07853443920612335, 0.07442804425954819,
0.07659614086151123, 0.07727987319231033, 0.07795453816652298,
0.07741698622703552, 0.07415343821048737, 0.07553839683532715,
0.07843122631311417, 0.07241300493478775, 0.07555609941482544,
0.07650258392095566, 0.07602944225072861, 0.07901545614004135,
0.0764797180891037, 0.0773397758603096, 0.07483021169900894,
0.07409574836492538, 0.07703899592161179, 0.07413967698812485,
0.07679666578769684, 0.07751286029815674, 0.0731402263045311,
0.07725394517183304, 0.07297830283641815, 0.0736132264137268,
0.07845885306596756, 0.07130026072263718, 0.0767732635140419,
0.07099021971225739, 0.07479016482830048, 0.07255673408508301,
0.07383453100919724, 0.07230090349912643, 0.07925284653902054,
0.07325638085603714, 0.07303617149591446, 0.07301820069551468,
0.0722828209400177, 0.0723155066370964, 0.07451298087835312,
0.07513204216957092, 0.07414237409830093, 0.07431397587060928,
0.07327131181955338, 0.0727747455239296, 0.0735969692468643,
0.07462701201438904, 0.07084706425666809, 0.07186666131019592,
0.07307460904121399, 0.07404603809118271, 0.07319138199090958,
0.07170368731021881, 0.0732947364449501, 0.07135351747274399,
0.0755353718996048, 0.07687194645404816, 0.0701783075928688,
0.07437694817781448, 0.06953031569719315, 0.07462451606988907,
0.0719367042183876, 0.0678849071264267, 0.07310346513986588,
0.07330557703971863, 0.07291031628847122, 0.07107178121805191], 'val_loss':
[1.4142568111419678, 1.255345106124878, 1.1434577703475952, 1.0570182800292969,
0.9876598715782166, 0.9319351315498352, 0.8843593001365662, 0.8415622711181641,
0.8044754266738892, 0.7755414247512817, 0.7476881742477417, 0.7220491766929626,
0.6981146335601807, 0.6765899658203125, 0.6538541913032532, 0.6325360536575317,
0.6118853092193604, 0.5923954248428345, 0.5740025043487549, 0.5554223656654358,
0.5376431345939636, 0.5225244164466858, 0.5043587684631348, 0.48898953199386597,
0.47319507598876953, 0.45948097109794617, 0.4445427656173706,
0.4307772219181061, 0.4171591103076935, 0.4042914807796478, 0.39245885610580444,
0.38051894307136536, 0.3689241111278534, 0.3582379221916199, 0.3469008505344391,
0.3364117741584778, 0.32638630270957947, 0.31652525067329407,
0.3070545196533203, 0.2985743582248688, 0.2891300618648529, 0.28068456053733826,
0.2732679843902588, 0.2669769525527954, 0.25734350085258484, 0.2494577020406723,
0.2425810992717743, 0.23602823913097382, 0.229015052318573, 0.2230094820261016,
0.21645747125148773, 0.21010708808898926, 0.2046583890914917, 0.199302077293396,
0.19464655220508575, 0.18954767286777496, 0.18325527012348175,

0.1788652539253235, 0.17372974753379822, 0.1697317361831665, 0.1652068942785263,
0.1605309247970581, 0.15666697919368744, 0.15271806716918945,
0.14906081557273865, 0.145598366856575, 0.1418919563293457, 0.13845014572143555,
0.13484804332256317, 0.13176842033863068, 0.12871655821800232,
0.12564197182655334, 0.12235742062330246, 0.12004334479570389,
0.11745979636907578, 0.11422594636678696, 0.11150725930929184,
0.10927415639162064, 0.10675183683633804, 0.10430485010147095,
0.10201577097177505, 0.10066715627908707, 0.09796740114688873,
0.095912642776961, 0.09367289394140244, 0.0921682491898536 7,
0.09012196958065033, 0.08804463595151901, 0.08644668012857437,
0.08456779271364212, 0.0829591751098 6328, 0.0820549950003624,
0.0799093097448349, 0.07868330925703049, 0.07690020650625229,
0.07595832645893097, 0.07619686424732208, 0.07350285351276398,
0.0716954916715622, 0.07075468450784683, 0.06944628059864044,
0.06801136583089828, 0.06734343618154526, 0.06629501283168793,
0.06645571440458298, 0.0641227513551712, 0.06288204342126846,
0.06209086626768112, 0.06170171499252319, 0.06019992381334305,
0.05988300591707 2296, 0.06060271710157 3944, 0.057671919465065,
0.05712909996509552, 0.0571388602256774 9, 0.05658664181828499,
0.05560004711151123, 0.053835030645132065, 0.052738361060619354,
0.05293845757842064, 0.05142738297581673, 0.05088392272591591,
0.05023927614092827, 0.05015582591295242, 0.04923008382320404,
0.04845862090587616, 0.04857160523533821, 0.0482192263007164,
0.04687173292040825, 0.04646948724985123, 0.04581551253795624,
0.044928841292858124, 0.0445998907089233 4, 0.04459115117788315,
0.04416554421186447, 0.04307800158858299, 0.04388055577874184,
0.04265214502811432, 0.04206033796072006, 0.04175757244229317,
0.04258140176534653, 0.04258767515420914, 0.04136422649025917,
0.04109777510166168, 0.039809633046388626, 0.0390402227640152,
0.03946177288889885, 0.038670849055051804, 0.03821897506713867,
0.0377715528011322, 0.037961602210998535, 0.037221092730760574,
0.037094201892614365, 0.0367241017520427 7, 0.03665253147482872,
0.03606833145022392, 0.035903096199035645, 0.036174435168504715,
0.035427212715148926, 0.035500381141901016, 0.0348332300782203 7,
0.034907326102256775, 0.034944236278533936, 0.03498629108071327,
0.03461186960339546, 0.035533688962459564, 0.033682987093925476,
0.0337754487991333, 0.03339718282222748, 0.03469222038984299,
0.03283999115228653, 0.03213920444250107, 0.03226634860038757,
0.0320700258016586 3, 0.0320291742682457, 0.031391337513923645,
0.034854140132665634, 0.031108027324080467, 0.03087291680276394,
0.030828189104795456, 0.031483873724937 44, 0.030402906239032745,
0.0306166671216487 9, 0.029668046161532402, 0.029891949146986008,
0.029319878667593002, 0.029304547235369682, 0.029819360002875328,
0.028940675780177116, 0.029794486239552498, 0.029062708839774132,
0.0284261591732502, 0.028276784345507622, 0.02802421897649765,
0.029462046921253204, 0.02845059707760811, 0.0276802890002727 5,
0.0276102200150489 8, 0.027630414813756943, 0.027273468673229218], 'val_mae':
[0.39719533920288086, 0.31313952803611755, 0.2673645615577698,

0.22936487197875977, 0.19767315685749054, 0.1728171706199646,
0.15316826105117798, 0.13290950655937195, 0.11465547233819962,
0.10601060837507248, 0.0954974192029953, 0.0893271267414093,
0.08349821716547012, 0.0851670652627945, 0.07940475642681122,
0.0756433755159378, 0.06900888681411743, 0.0688982903957367,
0.07103598862886429, 0.06677111983299255, 0.06434018909931183,
0.07219052314758301, 0.061301834881305695, 0.06285104155540466,
0.058549121022224426, 0.06307173520326614, 0.0570429228246212,
0.056200165301561356, 0.05209790915250778, 0.050215158611536026,
0.05320621281862259, 0.05186739191412926, 0.05008956044912338,
0.05145297944545746, 0.048248324543237686, 0.04615636542439461,
0.04673102870583534, 0.04545668140053749, 0.04414771497249603,
0.048066578805446625, 0.03961506858468056, 0.04094470292329788,
0.046866510063409805, 0.05345696583390236, 0.04210786148905754,
0.03930836543440819, 0.03902165591716766 4, 0.04120510071516037,
0.03898533061146736, 0.0437841713428497 3, 0.04075906425714493,
0.03649966046214104, 0.03777295723557472, 0.03943339362740517,
0.04248640686273575, 0.04014148935675621, 0.0361689031124115,
0.03801014646887779, 0.03635344281792641, 0.04033553600311279,
0.03820914775133133, 0.036099664866924286, 0.0369691327214241,
0.034986913204193115, 0.03402014076709747, 0.036769039928913116,
0.03961866721510887, 0.03755779564380646, 0.03498384729027748,
0.03421564772725105, 0.035808440297842026, 0.035622965544462204,
0.03379141539335251, 0.03685861453413963, 0.04068201407790184,
0.03351347893476486, 0.033731963485479355, 0.03579849377274513,
0.03482713922858238, 0.03321627154946327, 0.032011859118938446,
0.035099878907203674, 0.033189788460731506, 0.03392953425645828,
0.03685884177684784, 0.03270762041211128, 0.03424544632434845,
0.0315256230533123, 0.033984262496232986, 0.02835928089916706,
0.030806371942162514, 0.03213811293244362, 0.0317285917699337,
0.031814802438020706, 0.030274799093604088, 0.030565282329916954,
0.04014255851507187, 0.03283395618200302, 0.03021472506225109,
0.03131398186087608, 0.035035934299230576, 0.028005536645650864,
0.029774030670523643, 0.030289962887763977, 0.038595765829086304,
0.031251683831214905, 0.026722880080342293, 0.027920924127101898,
0.030309142544865608, 0.03042670525610447, 0.0318068265914917,
0.03095402754843235, 0.028583098202943802, 0.031246375292539597,
0.033421289175748825, 0.03051002509891987, 0.030564893037080765,
0.029964711517095566, 0.03030427359044552, 0.034958213567733765,
0.02579914592206478, 0.027826812118291855, 0.027199655771255493,
0.030193470418453217, 0.02701934240758419, 0.02799343317747116,
0.027294689789414406, 0.029276994988322258, 0.027342503890395164,
0.026763880625367165, 0.027785681188106537, 0.026783078908920288,
0.02791774831712246, 0.03012077882885933, 0.027798481285572052,
0.028047464787960052, 0.02804156392812729, 0.026081634685397148,
0.029684927314519882, 0.026136387139558792, 0.03105814941227436,
0.03265158459544182, 0.026265790686011314, 0.029524121433496475,
0.030085723847150803, 0.024577755481004715, 0.025331635028123856,

0.024603774771094322, 0.025594042614102364, 0.022833894938230515,
0.024009326472878456, 0.02708231657743454, 0.025377830490469933,
0.02614901214838028, 0.0322042778134346, 0.023341044783592224,
0.02174481377005577, 0.0263819582760334, 0.021433208137750626,
0.023538624867796898, 0.024600671604275703, 0.025602564215660095,
0.025227230042219162, 0.02408701553940773, 0.028553325682878494,
0.025725992396473885, 0.03002922609448433, 0.026199227198958397,
0.023566409945487976, 0.028005806729197502, 0.02230706252157688,
0.02120509184896946, 0.022541919723153114, 0.02311699092388153,
0.026010993868112564, 0.023115992546081543, 0.03989480435848236,
0.022084921598434448, 0.022345881909132004, 0.02423708885908127,
0.023382054641842842, 0.023079900071024895, 0.025722505524754524,
0.023577813059091568, 0.02208773046731949, 0.019819103181362152,
0.02209433540701866, 0.02647118642926216, 0.024096660315990448,
0.025425324216485023, 0.026438869535923004, 0.018005594611167908,
0.02172842063009739, 0.021015306934714317, 0.031449124217033386,
0.021497849375009537, 0.023168379440903664, 0.022290833294391632,
0.018908333033323288, 0.020769694820046425]}



4.2.5 4.4 Evaluation and Improvements

4.4.1 Model Evaluation After training the content-based filtering model, we can evaluate its performance using appropriate metrics. We use Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) as our evaluation metrics.

Evaluation Metrics:

- **Mean Absolute Error (MAE)**: Measures the average magnitude of errors in a set of predictions. It is calculated as the average of the absolute differences between predicted and actual values.
- **Root Mean Squared Error (RMSE)**: The square root of the average of squared differences between predicted and actual values.

[60]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Predict on the validation set
predictions = model.predict(X_val)

# Rescale the predictions and true values back to the original scale
y_val_rescaled = scaler_target.inverse_transform(y_val.reshape(-1, 1)).flatten()
predictions_rescaled = scaler_target.inverse_transform(predictions).flatten()

# Calculate evaluation metrics
mae = mean_absolute_error(y_val_rescaled, predictions_rescaled)
rmse = np.sqrt(mean_squared_error(y_val_rescaled, predictions_rescaled))

print(f'Mean Absolute Error: {mae}')
print(f'Root Mean Squared Error: {rmse}')
```

```
17/17              0s 2ms/step
Mean Absolute Error: 5745.838701923077
Root Mean Squared Error: 11915.382990344848
```

The MAE and RMSE values indicate the average and squared differences between the predicted and actual property prices, respectively. The values indicate that the model's property price predictions are reasonably accurate, deviating by roughly £5,745 on average.

4.4.2 Discussion and Improvements

Integrating Price Prediction into the Recommendation System The present model primarily emphasises the forecast of property prices, which plays a crucial role in the recommendation system. The ability to determine the projected price of a property enables the system to categorise and prioritise properties based on a user's financial constraints, which is a key component of tailored suggestions.

Transitioning to a Full Recommendation System To transform the price prediction model into a comprehensive recommendation system, we can take the following steps:

- User Profile Creation: Collect user preferences (property style, location, budget) and specific requirements (e.g., number of bedrooms).
- Similarity Calculation: Compare properties based on predicted price and other features (content-based filtering).
- Property Ranking: Rank properties based on alignment with user preferences and budget.
- User Feedback Loop: Gather user feedback to refine the recommendations.

4.5 Conclusion The content-based filtering model demonstrates promising results in predicting property prices based on the available features. This is crucial for the Personalised Property Recommendation System, as it allows for tailored suggestions based on both price and user preferences.

The evaluation metrics indicate that the model reliably predicts property prices. Future improvements include refining features, tuning the model, and incorporating additional data, which will enhance the system's accuracy and reliability. The successful implementation functions as a prototype, showing that current listings from OnTheMarket and historical transaction data from HM Land Registry may be used to construct strong models.

Accurate price prediction is a cornerstone of a comprehensive recommendation system. By integrating this with user preferences, the system can streamline property searches and cater to the needs of buyers, especially first-time buyers. This highlights the potential of machine learning in the real estate domain, paving the way for a more personalised and efficient property search experience.

word count: 1428

4.3 5. References and Resources

4.3.1 5.1 References

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

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

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

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

[5]: HM Land Registry Price Paid Data License. "Open Government Licence for public sector information." Available at: https://use-land-property-data.service.gov.uk/datasets/ccod/licence/view.

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

4.3.2 5.2 Resources Used

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/

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

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

4.3.3 5.3 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.